JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 2, March-April 2003

Quality Characteristics for Software Architecture*

Francisca Losavio and Ledis Chirinos, Central University of Venezuela, Caracas, Venezuela
Nicole Lévy and Amar Ramdane-Cherif, Université de Versailles St.-Quentin, France

Abstract

It is of general agreement that quality issues should be considered very early in the software development process, to mitigate risks and to facilitate the achievement of the overall software system. Moreover, the architecture of the system drives the whole development process. The fulfillment of nonfunctional quality requirements by a candidate architecture is crucial to select the convenient architecture on which the whole system will be articulated. This issue is very important in the construction of reliable evolutionary applications. Software development methods do not give many details on this important stage. This work deals with the specification of quality requirements for software architecture, introducing a technique based on the ISO 9126-1 standard. The quality characteristics of the ISO quality model are refined into attributes, which can be measured to enrich the information about the architecture. Our technique is used to help selecting a suitable architecture among a set of candidates, by comparing the values of the respective quality attributes. A case study illustrates the application of the technique on a monitoring system. Our approach facilitates the choice of the right decisions during the architecture analysis process. It could be easily integrated into a general software development process or into specific architectural design methods.

1 INTRODUCTION

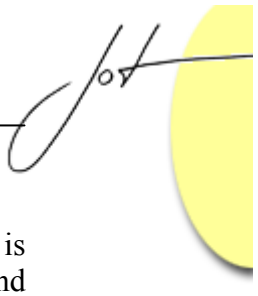
Quality requirements, captured as nonfunctional requirements in the early steps of software development, influence greatly the software system's architecture. However, also the system's core abstractions which are functional requirements, play an important role in the definition of the initial architecture. On the other hand, the quality requirements have to be "balanced" during the design process [Kazman et al. 2000]. Only

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

Cite this column as follows: Francisca Losavio et al: "Quality Characteristics for Software Architecture", in *Journal of Object Technology*, vol. 2, no. 2, March-April 2003, pp. 133-150.
http://www.jot.fm/issues/issue_2003_03/article2

recently the importance of a precise design of software architecture, not limited to graphical notations of boxes and lines, has grown up considerably for the construction of reliable evolutionary systems [Bachmann et al. 1996], [Bosch 2000], [Krutchen 2000]. Modern applications involving distribution, adaptability, interoperability, component reusability and real-time issues require an early definition of the system architecture in order to fulfill quality requirements, such as maintainability and reliability. They are crucial for the achievement of the overall functional purpose of the software system under construction. In particular, new information systems using Internet services, like Web-based e-commerce applications, are developed very rapidly for marketing needs, without much care about software engineering practices. Moreover, the quality of such products is not discussed. However, when an HTML page is displayed on a browser, we are immediately aware if we are in presence of a "good" or "bad" Web application. Factors such as usability, reliability (robustness) or efficiency (time or resource behavior) are involved in this quick evaluation. In fact the problem is that software developers do not have a clear description of the quality characteristics of a Web application, since in the development of these systems, as we have already pointed out, software engineering paradigms are often neglected. For example, even when the separation between application data semantics and presentation is an accepted paradigm, HTML is normally used regardless of this issue and only recently XML is being adopted. An interesting problem is then the specification of quality requirements. They may appear implicitly during functional requirements specification, as for example in a textual use-case description or in a scenario, but in standard object-oriented methods there are no explicit guidelines or explicit modeling elements on how to capture or specify them. Moreover, we feel that the design of software architecture should not be considered as an independent activity, but a step further in the development and evolutionary process of software products. Architectures should be considered as a main concern (Krutchen, 2000) to establish more clearly reusable frameworks, for guaranteeing to a certain extent, the overall quality of the resulting software products.

The main goal of this work is to propose an ISO 9126-1 [ISO/IEC 1998] based technique to specify the relevant quality characteristics, refined until the attribute level or measurable items, involved in the architectural design process. The specification and evaluation of these attributes, as steps of the architectural design, is the basis of the architectural transformation process, allowing the incremental adaptation of the initial architecture. This candidate, often selected on some key functional requirements of the system, is adapted or transformed during the design process to accomplish the established quality goals, which are the values that the system should attain in fulfilling a quality requirement. In this process, quality requirements are often transformed into implicit functional requirements for the final system [Bosch 2000], expressing them as the introduction of additional mechanisms, for example. However, in commonly used software analysis and design methods, the specification and evaluation of quality attributes is only performed on the basis of the designer's experience. The ATAM (Attribute-based Tradeoff Analysis Method) [Kazman et al. 2000] has some common points with our approach. It uses one level of quality characteristics (attributes) refinement, called utility (system goodness) tree, for prioritizing scenarios based on a



particular quality characteristic. The information on the architecture and attribute is captured into the ABAS (Attribute-Based Architectural Style) framework [Klein and Kazman 1999]. However, how to arrive to the utility tree, which quality view is it expressing and why only one level refinement, is not clear and the definitions of the quality attributes are not standards. The utility tree is used to give priorities to scenarios to identify sensitive points, from which a set of "test" cases for the architecture can be derived. The measures for the attributes are given in terms of stimuli, parameters and responses. Our approach considers the specification of the quality requirements using a quality model according to the ISO 9126-1 standard. This hierarchical model, which is structurally similar to the ATAM quality tree, is adapted to software architecture. The ISO quality model is now a software industry standard and it is defined at a high abstraction level, in terms of external/internal and quality in use views of quality characteristics. The quality characteristics (attributes for ATAM) are defined precisely in the standard, and the measures for the attributes are quite general and could be refined further for a particular application. The quality of the environment, where the software is running, is considered by the quality in use model, which is the user's view of quality in a specific context of use. In this work we are concerned only with the external/internal quality model, representing the user and developer views respectively. In order to achieve the quality in use, the system must have reached external and internal quality goals. The quality characteristics are refined into sub-characteristics manifested externally when the software is used as a part of a computer system and they are also a result of the evaluation of internal software attributes or measurable properties of an entity, appearing during the software development process. In our case, we have to "transfer" or "translate" these properties to the software architecture, which is an intermediate software product. The values obtained for the attributes during the development process can be used to verify internal quality goals, contributing to the validation of the external quality goals, required by the final software system [SO/IEC 1998]. The fact of having a precise specification of the quality attributes adds more information to the architectural specification, facilitating the analysis process for the selection of the architecture to solve a particular design problem.

Besides the introduction and conclusion, the main sections of the paper are the following: - The description of a general quality model, based on ISO 9126-1, for specifying the quality characteristics of software architecture is given. - A case study, where the general quality model obtained is used for selecting two different architectures for a soft real-time monitoring system using Internet facilities.

2 ADAPTING ISO 9126-1 QUALITY MODEL TO SOFTWARE ARCHITECTURE

ISO 9126-1 Quality Model

According to ISO 9126-1 [ISO/IEC, 1998], *quality* is defined as a set of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Different perspectives of quality can be considered: - user view, as the quality of the final product; - developer view, as the quality of the intermediate products generated by the different stakeholders during the development process; - end-user manager view, as the marketing requirements. The overall quality of a product can be the expressed by a combination of the different views. In our context, the user and developer (architect) views will be used. The work of McCall [McCall et al. 1977] distinguishes between two levels of quality features: factors and criteria. The former cannot be measured directly, while the latter can be subjectively measured. It inspired the ISO 9126-1 model. On this basis, ISO 9126-1 simplifies further the McCall's model, into the ISO 9126-1 quality model, now commonly accepted in the state-of-the-art of product quality specification. It 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. In practice, some influence could appear among the characteristics, however, in this work they will be considered independent to simplify our presentation. The quality characteristics are used as the targets for validation (external quality) and verification (internal quality) at the various stages of development. They are refined (see Figure 1) into sub-characteristics, until the *attributes or measurable properties* are obtained. In this context, *metric* or *measure* is a defined as a measurement method and *measurement* means to use a metric or measure to assign a value. Figure 1 shows these relations:

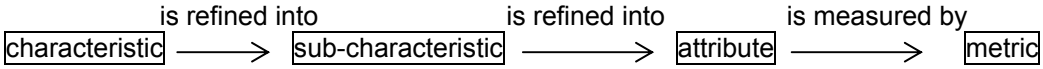
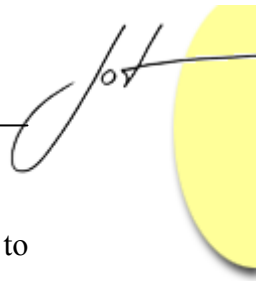


Fig. 1: Relations among the quality model elements

In order to monitor and control software quality during the development process, the external quality requirements are translated or transferred into the requirements of intermediate products, obtained from development activities. The translation and selection of the attributes is a non-trivial activity, depending on the stakeholder personal experience, unless the organization provides an infrastructure to collect and to analyze previous experience on completed projects. The definition of the main quality characteristics of the ISO 9126-1 standard for software quality measurement is shown in Table 1. The model should be adapted or customized to the specific application or product domain. In this sense, for a particular software product we could have a subset of



the six characteristics. In the ISO 9126-1 standard, no guidelines are given on how to customize the quality model.

Characteristics	**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 1. Characteristics of ISO 9126-1 Quality Model

The sub-characteristics are shown in Figure 2.

Notice that *compliance* means to adhere to standards, conventions or regulations and it is presented in [ISO/IEC 1998] as sub-characteristic of all the characteristics. We will consider it here only for functionality in order to abridge this presentation. Notice that the presence of the compliance sub-characteristic means that the remaining properties within the characteristic are assumed to be fulfilled by the particular standard chosen.

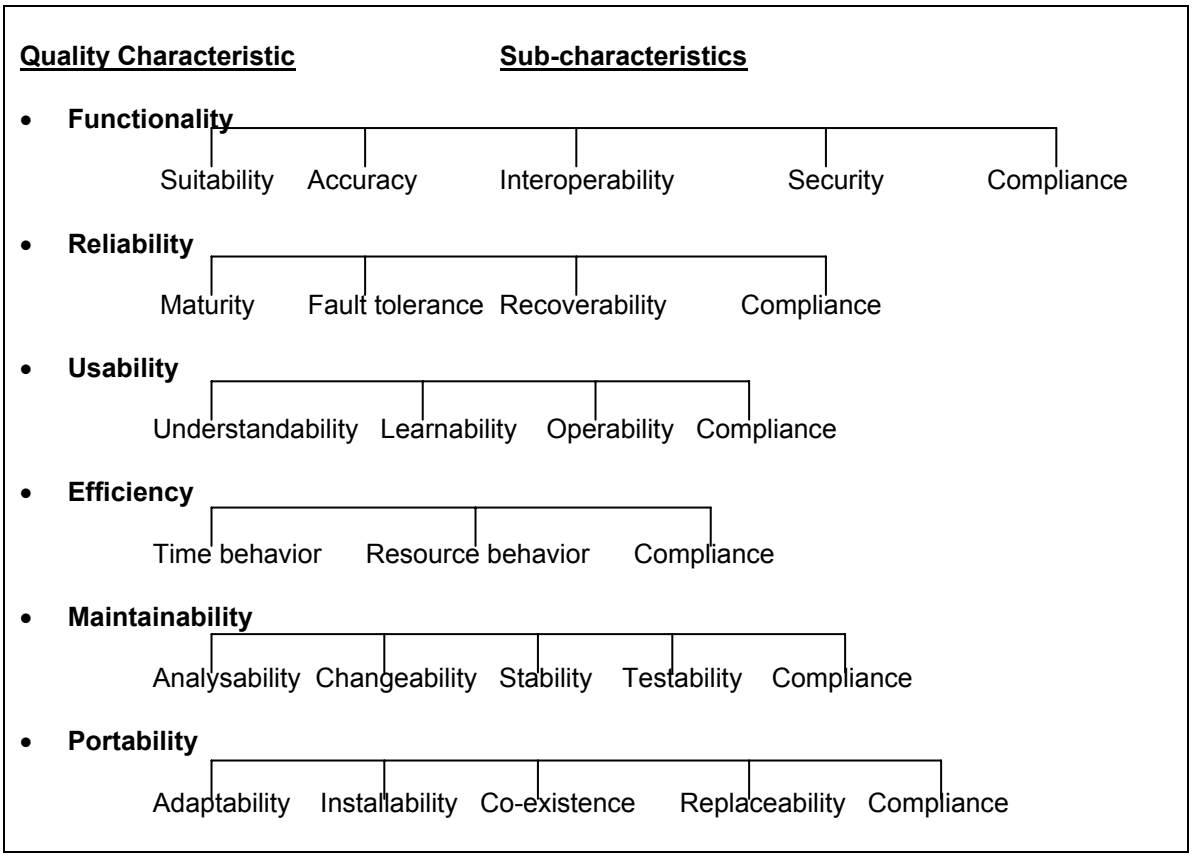


Fig. 2: Sub-characteristics of ISO 9126-1 Quality Model

Customization of the standard quality model for software architecture

In order to customize the ISO quality model, we should be aware of the properties that are expected from the architecture or generic framework (baseline) on which the software system must be built, considering it an intermediate product of the software development process. Hence a particular architecture, expressed at a high-level by the components, the connectors connecting them and a configuration or topology, must "satisfy" the six ISO 9126 characteristics or a subset of them. Each characteristic will be associated to attributes to be valued. These attributes will be associated globally to the architecture and/or to each component and connector. Measures will be used to quantify the quality attributes. These are defined as symbolic expressions at first and then could be defined more precisely using a formal language [Marcano et al. 2000]. During the architecture stepwise definition, it is possible to evaluate if the refinement of the architecture enhance the quality attributes. However this issue will not be discussed in this work.

For the final product there are quality goal values that must be reached or surpassed for each attribute. When the values are reached or surpassed then the architecture is said

to satisfy the quality characteristics required. The goal values are established in the requirements definition. In what follows we will explain how the quality requirements are refined into the corresponding sub-characteristics and attributes and how they are adapted to software architectures, giving the corresponding metrics for each attribute. Notice that characteristics and sub-characteristics are considered independent.

- **Characteristic Functionality**
  - **Sub-characteristic Suitability**: to have the adequate functions for the required tasks.
    It involves two aspects:
    - Presence: the tasks are specified, for example by means of use cases. For each task there must exist a functionality to accomplish it.
    - Appropriateness: the specification of the task is correctly refined, for example the sequence diagrams (see Figure 3) must be satisfied.

    At architectural level:
    1. The system's functionalities are identified. In this case, the sub-characteristic is refined into an attribute whose value is yes (1) or not (0). Notice that there are attributes whose values belong to the interval of integers [n..m], for example [0..1], meaning absence or presence. The metric is a scale to obtain rating levels.
    2. The sequence diagrams obtained from the functional requirements are refined. In case of having an architecture specification, the specified functionality is decomposed into functions associated to components and whose composition will meet the functional requirements of the system.

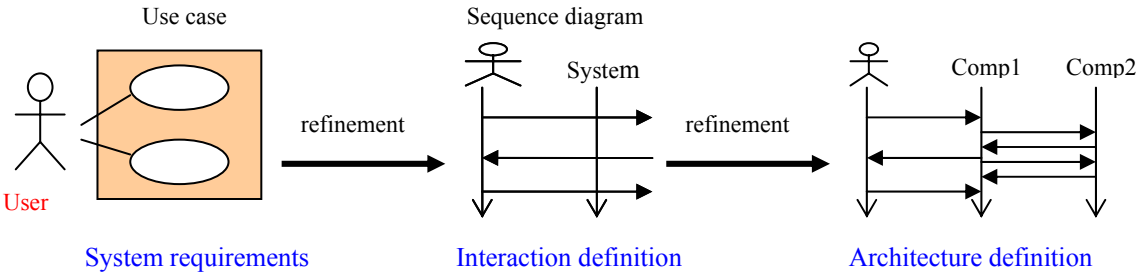


Fig. 3: Translation of system requirements to software architecture

- **Sub-characteristic Accuracy:** to provide the right or agreed results or effects with the needed degree of precision. It can be measured by an attribute on the source code. Hence it is delegated to the components in which will be defined the functions that will compute the values.
  At architectural level:

1. Identification of the components with the functions responsible of the computations (functional components).
2. The attribute is computed by the following metric:

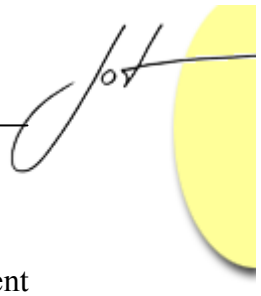
$$\Pi_i \, (Accuracy \, (functional \, components_i)).$$

- **Sub-characteristic Interoperability:** the ability to interact with on or more specified systems. Notice that it is used in place of compatibility to avoid ambiguity with replaceability.
  At architectural level:
  1. Identification of the connectors communicating with external specified systems. For example, to require CORBA compatibility implies the existence of CORBA components.
  2. It is refined into an attribute whose value is yes or not, depending on the presence or not of corresponding middleware components.
- **Sub-characteristic Security:** the ability to prevent unauthorized access to programs or data.
  At architectural level:
  1. It means to have a mechanism or device (software or hardware) to perform explicitly this task. It may be a component (for example, a service provided by the middleware) or a functionality integrated into a component
  2. It is refined into an attribute whose value is yes or not, depending on the presence or not of the mechanism or device
- **Sub-characteristic Compliance:** to adhere to standards, conventions or regulations. It is related to the development process.
  At architectural level:
  1. It is a very general property that cannot be directly applied to architectural design
  2. It is refined into an attribute whose value is yes or not, depending on the application of the required standard.
  3. The compliance to an architectural style can be defined as the satisfaction to the architectural constraints associated to it.

- **Characteristic Reliability**
  - **Sub-characteristic 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 the 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.
  - **Sub-characteristic 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.

- **Sub-characteristic 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.

  Remark: **availability** depends on the above three sub-characteristics of reliability and it is used in [Marcano et al. 2000]. 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.

- **Characteristic Usability**
  - **Sub-characteristic Understandability:** the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
  - **Sub-characteristic Learnability:** the capability of the software product to enable the user to learn its application
  - **Sub-characteristic Operability:** the capability of the software product to enable the user to operate and control it.

  These sub-characteristics can be refined into attributes translated to the GUI components .

  At the architectural level, they are independent from the architecture, which is transparent to the users, so they will not be considered here.

- **Characteristic Efficiency**
  - **Sub-characteristic Time behavior (performance):** the capability of the software product to provide appropriate response time, processing time and

throughput rates when performing its function under stated conditions. It is an attribute that can be measured for each functionality of the system.
At architectural level:
It is measured for each functionality and each user of the functionality by means of attributes computed by the following metric:
Σ_i *Performance (Component Functionality* $_i$*)* + Σj *Performance (Connector* $_j$*)*
Affected by the data flow to a functionality. The performance depends on:
- the stimulus/event/functionality;
- the path taken in the architecture in order to answer to a stimulus for a given functionality ;
- each component traversed, containing the executed functionality.

- **Sub-characteristic Resource utilization:** amount and type of resources used and the duration of such use in performing its function. It involves the attribute *complexity* that is computed by a metric involving size (*space* for the resources used and *time* spent using the resources).
  At the architectural level:
  The attributes can be defined and measured for each functionality and they are a characteristic of the style. Space and time are associated to the components. The values are associated to each component and/or connector for each functionality.

- **Characteristic Maintainability**
  - **Sub-characteristic Analyzability:** the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified, to be identified.
  - **Sub-characteristic Changeability:** the capability of the software product to enable a specified modification to be implemented.
  - **Sub-characteristic Stability:** the capability of the software product to avoid unexpected effects from modifications of the software (the risk of unexpected effect of modifications)
  - **Sub-characteristic Testability:** the capability of the software product to be validated.
    They are refined into the attribute *complexity* of the source code, computed by metrics involving, in particular, size.
    At the architectural level:
  - **Sub-characteristic Coupling** is a global property of the architecture relative to the exchanges between components; the attributes can be measured for each component using fan-in, fan-out metrics. It is a system attribute.
  - **Sub-characteristic Modularity** expresses the topology of the architecture, as the number of components depending on one component. It is an attribute computed for each component by metrics involving size.

- **Characteristic Portability**
  - **Sub-characteristic Adaptability:** the capability of the software product to be adapted to different specified environments using only its own functionality.
    At architectural level:
    1. The presence of mechanisms for adaptation, for example genericity or parameterization
    2. This sub-characteristic is refined into an attribute whose value is yes or not, depending on the presence or not of the mechanism
  - **Sub-characteristic Installability:** the capability of the software product to be installed in a specified environment.
    At architectural level:
    1. The presence of an install mechanism
    2. This sub-characteristic is refined into an attribute whose value is yes or not, depending on the presence or not of the mechanism
  - **Sub-characteristic Co-existence:** the capability of the software product to co-exist with other independent software in a common environment, sharing common resources
    At architectural level:
    1. The presence of a mechanism facilitating the co-existence
    2. This sub-characteristic is refined into an attribute whose value is yes or not, depending on the presence or not of the mechanism
  - **Sub-characteristic Replaceability:** the capability of the software product to be used in place of another specified software product for the same purpose in the same environment.
    It involves adaptability and installability.
    At architectural level:
    The attribute is expressed by a list (name) of replaceable components, for each component

3 APPLICATION OF THE CUSTOMIZED QUALITY MODEL TO A CASE STUDY

The quality model defined in the previous section will be used to compare two architectures based on two different architectural patterns: publisher/subscriber with push model [Buschman 1996] and repository [Shaw and Garlan 1996]. Notice that publisher/subscriber is also known as subject/observer [Gamma et al. 1995]. The architectures are used to implement a market stock exchange monitoring system. In what follows, the system requirements are briefly presented. More details on this application are given in [Ordaz Jr. 2000].

Requirements for a Stock Exchange Monitoring System

The primary goal of a real-time monitoring system is to capture, to analyze and to broadcast events (data) in real-time. It is a *soft* real-time system, where some of the events may miss their deadline, without affecting the whole system's behavior. The system is a real-time *data provider*, for monitoring in real-time small and medium size stock exchanges for brokers and independent investors. An antenna (*feed server*), external to the system, provides the data (*feed*) to the data server. A feed contains the relevant information of a stock exchange transaction. Feeds are supposed to be reliable and available. The clients (*brokers*), distributed in different geographical locations, subscribe with the data server. When a change on the feed to which a client has subscribed occurs, the feed is broadcasted to him by the data server, according to a strict time delay. The time delay will depend on the network structure used to send the information to the clients. The type of service offered depends on this delay. Requirements for the system are high security, availability, platforms heterogeneity, distribution of clients, reliable information with strict deadlines. It is known that these characteristics are not independent, and there must be a tradeoff to determine priorities. Internet facilities through commercial browsers are required for the system.

Architectures proposed for the monitoring system

The proposed architectures based on two different architectural patterns, publisher/subscriber (push model) and repository are shown in Figures 4 and 5 respectively.

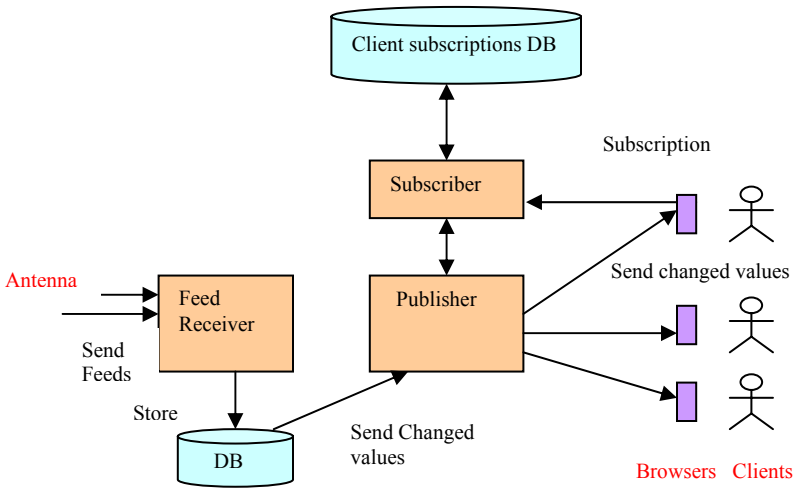


Fig. 4: Architecture based on the publisher/subscriber pattern

The publisher/subscriber memorizes the client subscriptions and the actual values in the Client Subscription DB and the DB respectively.

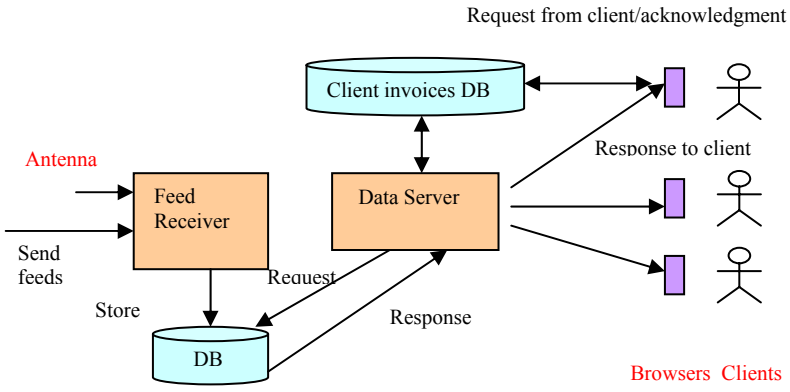


Fig. 5: Architecture based on the repository pattern

The repository memorizes the actual values in the DB and the client requests in the Client invoices DB, for invoicing purposes.

Comparison of the architectures

The results presented in Table 2 show that publisher/subscriber is better than repository with respect to security and efficiency in time behavior. However, repository is winner for maturity (there are fewer components in the architecture) and for efficiency in time (the Browser displays only on request). Hence only the resource utilization in time would favor repository. The analysis now consists in prioritizing the characteristics, i.e. decide which are the most important for the software system. This step corresponds to customize the ISO 9126-1 quality model to the problem domain [Losavio et al. 2002]. According to the requirements for the stock exchanges monitoring system, security and efficiency in time behavior are more important than reliability (maturity) and efficiency in resource utilization (space and time), according to the initial nonfunctional requirements on the problem domain. Hence the architecture based on the publisher/subscriber architectural pattern is selected as the initial candidate architecture. Notice that external characteristics, such as the volume of client requests, mostly related with the user's behavior affecting components and/or connectors, cannot be considered in our evaluation and taken into account for comparison. Once this broad and quick selection has been performed on the basis of the analysis of the table, the final decision could be further corroborated executing and evaluating scenarios related only with the quality characteristics relevant to the application. In this case, the profile approach suggested by [Bosch 2000] or the scenarios approach proposed by [Kazman et al. 2000] could be applied.

Characteristics	Sub-characteristics	Publisher/Subscriber	Repository	Comments and results
Functionality	Suitability	yes	yes	
	Accuracy	=	=	No special computation required
	Interoperability	yes	yes	Communication through browsers
	Security	Mechanism for a subscription	Mechanism for each client request	**Publisher/subscriber is better**
Reliability	Maturity	Maturity (Reception) + Maturity (DB) + Maturity (Publisher) + Maturity (Subscriber) + Maturity (subscriptionDB)	Maturity (Reception) + Maturity (DB) + Maturity (Data Server) + Maturity (Client invoices DB)	**Repository is better**
	Fault Tolerance (availability)	=	=	Depends on additional mechanisms
	Recoverability	=	=	Depends on additional mechanisms
Usability		=	=	Depends on the browser's GUI
Efficiency	Time behavior (time spent from the data reception to the data delivery)	time (Reception)+ time (store in DB)+ time (send changes)+ time (Publisher)+ time (send changed values)	time (Client Request)+ time (Client invoices DB)+ time (Data Server)+ time (request)+time(DB)+ time (reponse Data Server)+time (Response to Client)	**Publisher/subscriber is better**
	Resource utilization (time)	Browser displays always	Browser displays on request	**Repository is better**
	Resource utilization (space)	Size (subscription DB)	Size (invoices DB)	Depends on external issues, such as the volume of client requests
Maintainability		=	=	Depends on the code in modules
Portability		=	=	Depends on additional mechanisms

Table 2. Comparison of publisher/subscriber and repository with respect to quality attributes

Summary of the technique

Summarizing our approach, the activities performed to accomplish the process for evaluating and comparing architectures, on the basis of the quality characteristics specification, are listed in Table 3, as follows:

Activities:
1. Analyze the main functional requirements and nonfunctional requirements for the system, to establish the quality requirements and quality goals
2. Use the customized ISO 9126-1 quality model defined in Section 2.2 for the architecture as a framework. Some of the metrics could be further specified, according to specific components and/or connectors
3. Present the initial candidate architectures
4. Construct the comparison table for the candidate architectures
5. Prioritize the quality characteristics taking into account the system's quality requirements and quality goals. The customization of ISO 9126-1 to the problem domain can be used to organize hierarchically the characteristics.
6. Analyze the results summarized in the table, according to the given priorities obtained in step 5
7. Select the initial architecture, among the evaluated candidates, on the basis of the previous analysis
8. If a finer analysis is required, scenarios or profile-based approaches could be used, considering only the quality characteristics relevant to the problem domain, obtained in step 5.

Table 3. Method for comparing architectures, based on the ISO-9126-1 quality attributes specification

4 CONCLUSION

An approach for specifying the quality requirements of a software system has been presented as a repeatable technique. The ISO/IEC 9126-1 standard has been used to define a quality model for software architecture. This framework has been applied to a simple case study for comparing two architectures and selecting the best suited to the problem, on the basis of the initial nonfunctional requirements. Finally, the activities involved in the application of the technique have been summarized. The specification of the quality attributes using a quality model based on international standards offers a global and broad view of the quality characteristics and attributes for software architecture, form the user and architect points of view. We consider that our approach is similar to the ATAM analysis technique, differing, however from this in several important aspects: the definition of the quality characteristics conforming to an industrial standard and a more general definition of the measures for the attributes that could be further refined, according to the particular application. Moreover, the comparison table produced can be used to derive the scenarios, in the sense of ATAM. However, both approaches could be easily integrated, with multiple benefits. Our technique can be easily integrated in generic development process frameworks, such as the Rational Unified Process [Krutchen 2000] or in specific architectural design methods, such as the J. Bosch method [Bosch 2000], or the Architecture Based Design (ABD) Method [Bachmann et al. 1996].

We feel that this work is a step forward towards quality requirements specification, systematization and improvement of the architectural design process, with built-in quality issues. Other undergoing research issues are the integration of this technique to the Unified Process and the formal specification of architectural styles and patterns [Marcano et al. 2000], [Meyer and Souquières 2000], taking account of quality attributes [Losavio and Levy 2001].

REFERENCES

[Bachm00] Bachmann F., Bass L., Chastek G., Donohoe P., Peruzzi F.: *The Architectural Based Design Method*. CMU/SEI-2000-TR-001, ESC-TR-2000-001, 2000

[Bosch00] Bosch J.: "Design and Use of Software Architecture", *ACM Press*, Harlow, England, 2000.

[Busch96] Buschman F., Meunier R., Rohnert H., Sommerlad P., Stal, M.: *Pattern-Oriented Software Architecture. A System of Patterns*, John Wiley & Sons Inc., New York, 1996

[Gamma95] Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns – Element of Reusable Object-Oriented Software*. Addison Wesley, New York, 1995.

[ISO/IEC98] ISO/IEC: FCD 9126-1.2: Information Technology - Software Product Quality. Part 1: Quality Model, 1998.

[Klein99] Klein M., Kazman R.: *Attribute-Based Architectural Styles*, CMU/SEI-99-TR-022, ESC-TR-99-022, 1999.

[Krutch00] Krutchen P.: *The Rational Unified Process. An Introduction*, 2nd. Edition, Addison Wesley, Reading, Massachussets, 2000.

[Kazm00] Kazman R., Klein M., Clements P.: *ATAM: Method for Architecture Evaluation*. CMU/SEI-2000-TR-004, ESC-TR-2000-004, 2000.

[McCall77] McCall J.A., Richards P.K., Walters G.F.: *Factors in Software Quality*. Vol. 1, 2, 3, AD/A-049-015/055. Springfield, 1977.

[Losav01] Losavio F., Lévy N. : "Specification of Attribute-based Architectural Styles". *Proceedings CD-Rom, http://www.umag.cl/ec of XI Encuentro Chileno de Computación, Jornadas Chilenas de Computación 2001*, Punta Arenas, Chile, 2001

[Losavi02] Losavio F., Chirinos L., Pérez M.: Attribute-based techniques to evaluate architectural styles. Case Study for interactive systems. Acta Científica Venezolana, Vol. 53, 2, 2002.

[Marca00] Marcano R., Levy N., Losavio F. : Spécification et Spécialisation de Patterns en UML et B. Proceedings of LMO'2000 – Langages et Modèles à Objets, Ed. Hermès, Montréal (Ca), 245-260, 2000.

[Meyer99] Meyer E. and Souquières J.: A systematic approach to transform OMT diagrams to a B specification. Proceedings of FM'99 : World Congress on Formal Methods in the Development of Computing Systems. Toulouse (F), September 1999.

[Ordaz00] Ordaz Jr. O. : Aplicaciones Tempo Real en Internet: Arquitecturas, Lenguajes y un Caso de Estudio, License Thesis, Universidad Central de Venezuela, Caracas, 2000.

[Shaw96] Shaw M., Garlan D.: Software Architecture – Perspective of an Emerging Discipline, Prentice Hall, Upper Saddle River, New Jersey, 1996.

About the authors

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

Ledis Chirinos obtained her MSc. degree in Computer Science from the Central University of Venezuela in 1999, where she continued with a PhD program. She works at the LaTecS laboratory, of the ISYS research center, Faculty of Scence, Central University of Venezuela. Her main interests are software quality, software measurement and software architecture. E-mail: lchirinos@cantv.net

Nicole Lévy is a professor at the University of Versailles, Saint-Quentin en Yvelines, France. She holds a doctoral degree from the Nancy University. She is Director of the ISTY and a research staff of the PRISM Laboratory, Versailles, where she coordinates the SFAL (Spécification Formelle et Architecture Logicielle) research group. Her main research interests are formal and semiformal development methods, formalization of styles and architectural patterns. E-mail: Nicole.Levy@prism.uvsq.fr

Amar Ramdane-Cherif received his Ph.D. degree from Pierre and Marie university of Paris in 1998 in neural networks and IA optimization for robotic applications. Since 2000, he has been associate Professor in the laboratory PRISM, University of Versailles, Saint-Quentin en Yvelines, France. His main current research interests include: Software architecture and formal specification, dynamic architecture, architectural quality attributes, architectural styles and design patterns, pervasive computing and communications, dependable systems architecture, E-mail: amar.ramdane-cherif@prism.uvsq.fr