

QUALITY MODELS TO DESIGN SOFTWARE ARCHITECTURES¹

Losavio F., Chirinos L.

Laboratorio de Tecnología del Software (LaTecS)
Centro ISYS, Facultad de Ciencias,
Universidad Central de Venezuela
Apartado 47567, Los Chaguaramos 1041-A, Caracas, Venezuela
flosavio@isys.ciens.ucv.ve, lchirino@isys.ciens.ucv.ve

Pérez M.

Laboratorio LISI
Departamento de Procesos y Sistemas
Universidad Simón Bolívar
Apartado 89000, Caracas, Venezuela
movalles@usb.ve

Abstract

This paper addresses the problems related with the construction of quality software systems. A generally accepted claim is that the evaluation and control of each stage of development in a well-defined process, will improve the overall quality of the final software product. In particular, we are interested in the early analysis and design stages, where an intermediate product is the architecture of the system. Several approaches will be discussed here, where quality models are established in order to determine the influence of the system non-functional characteristics as part of the software initial requirements, on the final software system. These characteristics affect mostly the software system structure or architecture. The main goal of this paper is to discuss three important approaches based on quality models: ISO 9126, Dromey and ABAS (Attribute-Based Architectural Styles), and establish several criteria or points of comparison. These approaches have been selected in this study because they share the fact of considering quality models related to the product or products obtained in the early stages of the software development process. In this sense, they are good candidates to be used in an architectural design process. The problem of establishing the quality of software architecture is, in general, not an easy one. But the selection of an architecture that guarantees, for example, a portable and extensible system, is crucial in component-based and object-oriented development. At present there are no general methods for evaluating architectures to assure these issues, only heuristics based on experience and practice. We believe that the study of the above approaches will be useful to introduce design quality issues in a development process that does not consider them explicitly, which is actually a usual practice. New techniques and methods must be defined in order to deal with the quality of design, for an architecture-based development. As a consequence, recent trends in software development processes consider the introduction of quality requirements from the earliest stages of development. On the other hand, even if the semantics of the terminology is similar in the different approaches studied, the names of the terms used are quite different; we think that our discussion will help to unify the terminology in the interesting field of quality architectural design of software systems.

Keywords: architectural design, quality model, reusable design component, software architecture, software design

¹ This work is a result of the CEE INCO SQUAD project #EP 962019 and the CDCH ARCAS project #03.13.4584.00 of the Universidad Central de Venezuela

INTRODUCTION

The growing complexity of software systems increases the need of more rigorous approaches for driving early design decisions. These decisions will affect the structural and behavioral issues of the final system. Hence, it is nowadays of crucial importance to fulfill not only the *functional requirements* or services that the system has to accomplish, but also other characteristics which are *non-functional requirements*, such as portability and extensibility. These characteristics may affect the *quality* of the global architecture of the system. The selection of an adequate architecture or a combination of different architectural styles for a software system is an open research problem that has been treated extensively in the literature [SG 96], [BCK 98]. Moreover, it is a key issue for component-based and object-oriented software development. A wrong early decision on the choice of the architecture will condition the whole software development and may conduce to the failure of the final system. Some examples are in real-time and distributed computing domains. Successful software projects tend to be associated with sound processes [Dro 96] and robust architecture. Moreover, the achievement of qualities such as performance and modifiability depends more on the software architecture than on code level practices, such as language choice, algorithms, and so forth [Kaz et al 98]. Hence it is extremely important to choose architectures that meet specific issues or required properties, with a certain degree of confidence. This confidence can only be gained on the basis of formal analysis or precise measures. However, the definition of these measures is still a research issue for the design stage. In this paper we will deal with the problem of establishing the properties that should be defined to obtain a quality architectural design. We discuss several approaches that define quality models that can be used in architectural design methods. Actually some of these approaches are used mostly to evaluate the final software product and not the intermediate products of the development process, such as the system's architecture. A comparison of these approaches is then established, on the basis of several comparison criteria.

Software quality is still a vague and multifaceted concept, which means different things to different people [ISO 91]. The way the quality is measured depends on the viewpoint taken [Kru 95]. This makes the direct assessment of quality very difficult. In order to define and apply better the quality notion, the research community has developed indirect models that attempt to measure software product quality by using a set of quality attributes, characteristics and metrics. A *quality model* establishes a framework to perform some kind of measurement of the specific desirable characteristics that are needed in the final system and perceived by the end-user. An important assumption of these models is that internal product characteristics, related to the product development, affect external product attributes or *quality in use*. By evaluating a product's internal characteristics, some conclusions may be drawn about the product's external quality attributes. This approach is followed by many software-metrics advocates because it offers an objective and context independent view of quality. Several of these approaches that are directly related with the definition of a quality model will be discussed in this paper, in order to establish supporting mechanisms and tools for a quality architectural design.

Besides this introduction, this paper is structured as follows: three sections describing and discussing the selected approaches, with a conclusion on each approach: the ISO 9126 quality model, the Dromey model and the ABAS design component. The fourth section will be devoted to the comparison analysis of the three approaches and the criteria used for comparing. The results of the study will be shown in a table. Finally, the conclusion is formulated, discussing the benefits of incorporating a well-defined quality model as an important activity within architectural design methods.

ISO 9126 QUALITY MODEL

ISO 9126 [ISO 91] proposes a general model, inspired in McCall's model [McCRW 77], to specify and evaluate the quality of a software product from different perspectives or views, acquisition, development, maintenance. It considers *internal characteristics*, which are related to the software development process and environment or context and *external characteristics*, which are observed by the end-user on the final software product. The view of quality, on these bases, can be internal or external, and it also affected by the

stakeholder view in the particular stage of development. An external characteristic can be measured internally, but its name and measure may be different, according to the stage of development. For example, *portability* is an external characteristic according to ISO 9126: we can speak of a portable system, from the point of view of the system's end-users. On the other hand, the design can be extensible from the point of view of the system engineer in the design phase, we will then speak in terms of *extensibility*. An important issue on software product quality is that the product internal characteristics determine or influence its external characteristics. In order to establish this influence, internal characteristics must be *linked* or related in some way to external characteristics.

ISO 9126 defines characteristics that can be subdivided into sub-characteristic, introducing the refinement notion, but it does not provide guidelines for a low-level refinement. This model is organized in six external characteristics (see Table 1), which can be refined into sub-characteristics, expressing the combined effort of the quality characteristics on the end-user. The last level of this hierarchic model is conformed by the *attributes*. These are measured using *external* and *internal metrics*. Internal characteristics influence external characteristics in order to obtain the external behavior of the system required by the end-user. For example *adaptability* can be an internal characteristic that may directly affect the final system's *efficiency*, which is an external characteristic. Recently [ISO 98a], the definition of the six characteristics presented in Table 1 have been slightly modified in ISO 9126 part 1.2, with the *quality in use* view. Quality in use considers all the observable characteristics of the final system, once this has been delivered and used by its end-users. External quality is then limited to the characteristics observed on the completed system, that is being used or tested by stakeholders that are not the system's final users. Since the ISO 9126 model considers internal quality characteristics, the approach can be used to evaluate and monitor the intermediate products obtained during the different stages of the software development process. In general, no guidelines are given for constructing the quality model, not even for the final software product. This is the purpose of ISO 14598-3 [ISO 98b], which proposes general guidelines in this sense. In particular, the SQUID approach [Bøe et al 99], based on ISO 9126 and ISO 14598-3, can be used to evaluate software design as a stage of the development process. SQUID offers also a tool for supporting its main activities. Some interesting results have been obtained evaluating the design of interactive systems using SQUID [LC 99], [Chi 99].

Characteristic	Description
Functionality	A set of attributes that bear on the existence of a set of functions and their special properties
Reliability	A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time
Usability	A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users
Efficiency	A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions
Maintainability	A set of attributes that bear on the effort needed to make specific modifications. Modifications may include corrections, improvements or adaptations of software to changes in the environment and in the requirements and functional specifications
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another. The environment may include organizational, hardware or software environment

Table 1. Quality characteristics, according to ISO 9126 [ISO 91]

The ISO 9126 customization. An example in the interactive systems domain.

The quality characteristics presented in Table 1, have to be refined and customized to construct the quality model, according to the domain or application. Figure 1 shows this customization for interactive systems using the PAC (presentation, Abstraction, Control) architectural style [BC 91], [Bus et al 98], [LM 96]. Notice that the PAC style is based on the Mediator design pattern. The components of this style are GUI (Graphical User Interface) agents.

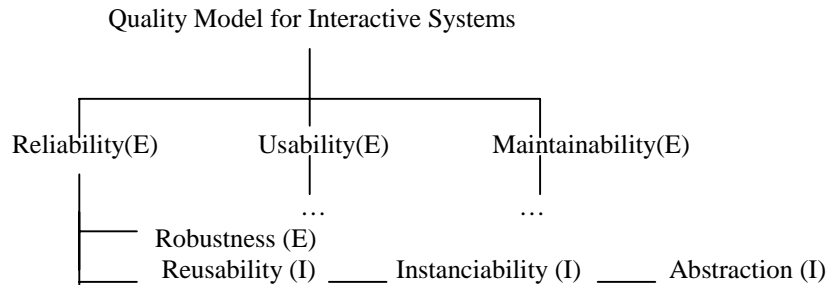


Figure 1. A Quality Model for Interactive Systems

The non-functional requirements for interactive systems in this case are *Reliability*, *Usability* and *Maintainability*, which are expressed as external characteristics, observable by the end-user on the final system. An interactive system is meant to be *reliable*, in the sense of *robustness*, *usable* in the sense of being easy to learn and friendly for the user, *maintainable* for being exposed to frequent changes in the GUI (Graphical User Interface). *Reusability*, *Instanciability* and *Abstraction* are internal sub-characteristics of *Reliability*. The system is constructed according to a component-based approach, using reusable design components, GUI frameworks and reusable toolkit code [LM 97]. The reusable frameworks are instantiated for each GUI agent. Abstraction can be measured in terms of its internal attributes, such as size, with usual object-oriented metrics. Usability and Maintainability are refined further into other sub-characteristics. For more details on this application, the reader is referred to [Chi 99], [LC 99]. The SQUID guidelines, based on ISO 14598-3 are shown in Table 2. These were used to construct and apply the quality model shown in Figure 1. The overall development process has been monitored with respect to the established quality goals, emphasizing in our case, the analysis and design stages.

It is clear that the commonly used kind of reasoning for establishing the relations among internal and external quality characteristics is mostly empirical and based on heuristics and designer experience. Nevertheless, it is crucial for determining to which extent an internal characteristic may affect external characteristics of the whole system, often compromising its functionality or performance. Usual guidelines provided for constructing the quality model do not go any further in this sense, as it will be seen later for the Dromey's model. It has to be pointed out that the SQUID approach provides the "model collation" notion for linking internal to external characteristics. However, the data collation must be provided by the specialist or user of the SQUID tool and it is subjective to the specific domain or application that is being considered, where the designer's experience plays a great role.

It can be observed in the example, that each internal characteristic is related to a particular stage in the development process. Reusability is considered while instantiating a GUI framework, in the design stage, as the number of public abstract classes of the framework that have to be instantiated. Hence, the deliverables of each stage of the process can be taken into account as sub-products of the final product, the software system.

Guidelines

1. Model the development process

2. Specify the ISO 9126 quality model

3. Specify the measures

4. Collate the models

Description

Specification of the development model in terms of the project objects types: deliverables, development activities, review points

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

- 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

- 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.

Table 2. SQUID guidelines for constructing the ISO 9126 quality model

Relevance of the ISO 9126 approach to architectural design

Notice that the choice of the architecture influences greatly the overall development. The system architecture, usually considered at the end of the analysis or at the beginning of the design phase in traditional development methods, could be evaluated using the ISO 9126 model according to the SQUID guidelines, as a product of the design step. Moreover, the whole development process can be modeled, taking account the quality issues. Similarly, an architectural design method could be modeled using this general framework.

Conclusions on the ISO 9126 approach

From the above discussion, the following conclusions can be derived:

- It proposes a well-defined quality model
- It is based on a quality definition, on the basis of internal, external and in use view of quality
- It allows the instantiation of the quality model according to the context
- Does not offer guidelines for constructing or applying the quality model. However, ISO 14598-3 provides these guidelines
- Does not consider explicitly the development process. It is more oriented towards the implementation stage, or the delivered system (external and in use quality view). However, SQUID based on ISO 14598-3 considers intermediate stages of development

DROMEY'S QUALITY MODEL

Dromey [Dro 96] states that quality characteristics or *high-level attributes*, cannot be built directly into software, but instead important *product properties*, like modules without side effects, can be identified built and measured as *tangible* properties, influencing or inducing high-level attributes, such as reliability or maintainability. These are *intangible* properties in the sense that they cannot be directly measured. In order to point out this influence, product properties must be *linked* with high-level attributes. For this, a quality model framework is proposed. The important thing is to focus on those high-level attributes that describe the priority requirements for the software. Products are built of components. *Rules-of-form* govern each component type. So, product quality is determined by the choice of the components, tangible properties of individual components, tangible properties of the component composition. *Rules-of-composition* govern the way components are used in the context of other components. Violating some of these rules may affect the

functionality of the system or even non-functional requirements, such as performance. The product properties affecting quality, according to the Dromey's model are shown in Table 3.

Product Property	Description
Correctness	Significant properties whose violation affect the product performance; they can be internal or contextual
Internal	Associated with individual components, such as the termination property of a loop
Contextual	Associated with the way components are used in the context, considers the external influence by and on the use of a component, such as correctness or maintainability
Descriptive	Associated with the usability characteristic; a GUI specification, for example

Table 3. Product Properties that affect quality

Dromey affirms that if it is accepted that a quality process is needed to produce a quality product, then a high level attribute process-mature should be added to each product quality model, in order to relate process to product quality.

As we have already remarked for ISO 9126, the delicate point is still the definition of the link between the tangible and intangible properties, where the approach is rather empirical, especially for the design stage. It will be seen in the next section that the ABAS [KK 99] approach can be a contribution to some extent in this sense. Only common sense reasoning and experience is recommended to establish this link in the Dromey's model. For example, the correctness property implies that if a component is not implemented or does not function as intended or designed, neither the functionality nor the reliability of the system can be guaranteed. Therefore, correctness influences the functionality and reliability quality characteristics.

General guidelines, shown in Table 4, for a method to construct and refine the product quality model are also given by Dromey. QMOOD (Quality Model for Object-Oriented Design) [BD 97] is an approach and tool (QMOOD++) similar to SQUID [Bøe et al 99] , for supporting and implementing an extension of the Dromey's guidelines to an object-oriented context.

Guidelines

1. Identify a set of high-level quality attribute for the product.
2. Identify the product components
3. Identify and classify the most significant and tangible quality-carrying properties for each component
4. Propose a set of axioms or linking product properties to quality attributes
5. Evaluate the model, identify its weakness and either refine it or scrap it and start again

Table 4. Dromey's method for constructing the quality models

Quality models are defined in this way for each stage of the development: requirements, design, implementation, differing in this from ISO 9126 and SQUID in the sense that these approaches only considers one quality model, and the tangible properties may be measured differently in each stage of development. For the Dromey's *Implementation quality model*, the ISO 9126 model is taken directly, adding the *process-mature* characteristic to the six characteristics shown in Table 1. This fact points out again that the ISO 9126 model is more driven towards the final coded software product, obtained during the implementation stage and does not take into account explicitly the early stages (analysis and design) of the development process. However, as we have pointed out in the previous section, the SQUID tool, following ISO 14598-3, allows the process definition, considering deliverables in each stage of development (see Table 2). For the Dromey's *Requirements quality model*, the high level attributes are: accurate,

understandable, implementable, adaptable, process-mature. Requirements must be *accurate* because they must satisfy precisely the client's needs. On the other hand, the results of the requirements model must be *implemented* by the design stage. *Adaptability* is required because requirements must undergo changes. For the *Design quality model*, the quality high-level attributes are: accurate, effective, understandable, adaptable, process-mature. Dromey states that neither the ISO-9126 nor the proposed requirements quality model are adequate for design, however there are similarities. A design must *accurately* satisfy the requirements, be of course *understandable, adaptable* in the sense of supporting changes and developed using a mature process. However, design quality is distinguished from requirements quality by its *effectiveness* in solving the problem at hand.

Relevance of the Dromey's approach to architectural design

The Dromey's quality models demonstrate to a certain extent that it is possible to create a framework that can be used practically to build better products or assess and ensure their quality. In our opinion it is very important to have a clear differentiation between the stages of development, in particular the design phase, where architectural considerations can be derived. The ISO 9126 approach does not offer this differentiation. However, SQUID, based on ISO 9126 and ISO 14598-3, offers the possibility of modeling the development process. On the other hand, the Dromey's and ISO 9126 approaches are quite similar with respect to the link between properties and high-level attributes.

Conclusions on the Dromey's approach

From the above discussion, the following comments on the Dromey's approach can be formulated:

- It is an extension of the ISO 9126 quality Model
- It proposes a different quality model for each stage of the development process
- It provides guidelines for the construction of the quality model
- It associates the product and process notions
- It introduces the importance of the context or problem domain

ATTRIBUTE-BASED ARCHITECTURAL STYLES (ABAS)

An architectural style is defined by Shaw and Garlan [SG 96]. A style, framework [Pre 95] or architectural pattern [Bus et al 96] provide a skeleton for components or for the global architecture of the system. It includes a description of the component types and their topology, the pattern of data and control interaction among the components. An informal description of benefits and drawbacks and possible uses of the pattern is also provided. In some design patterns [Gam et al 95], which according to Buschmann are components of architectural patterns, a description of a possible implementation in a target language is also given. Some formal descriptions of the style provide also an invariant condition for sufficient conditions for a formal design description to be an instance of the architectural pattern. These conditions may be mechanically tested [LL 99]. Architectural styles or patterns are important artifacts because they define classes of designs along with their associated known properties. They offer experience-based evidence on how each class has been used historically, along with a qualitative reasoning to explain and justify their specific properties. An example of this kind of reasoning is: "use pipe and filter style when reuse is desired and performance is not a top priority". Architectural styles are powerful because they provide the user with previous and tested knowledge of experienced designers for reuse. Nevertheless, the application of the style to a particular problem or domain is not straightforward because the criteria for the classification and comparison of styles are not clear enough and the descriptions of the styles are imprecise and generally informal.

An ABAS [KK 99] is a reusable design component like the architectural patterns [Bus et al 96] or design patterns [Gam et al 95]. It is conceived to make architectural styles the foundation for a more precise reasoning about architectural design. This is accomplished associating a *reasoning framework* (quantitative and/or qualitative) with a *description* of the architectural style. The reasoning framework is based on a *quality model* specific to a characteristic, called *attribute* in this approach. Only one attribute at a time is considered when ABASs are used in design or analysis, because ABAS is associated with only one attribute

reasoning framework, called an *attribute model*. For example, if an architectural style is interesting from both a performance and a reliability point of view would be motivation for creating the respective performance and reliability ABASs. Design and analysis of software architecture is based on the reuse of known patterns of software components with predictable properties. The information for characterizing the quality attributes is divided into three categories and shown in Table 5.

Category	Description
External stimuli or stimuli	Events, such as messages or keystrokes, that causes the architecture respond or change. They are responsible for the presence of an attribute or property, which is seen as an end-user requirement for the architecture. Example: with respect to the modifiability attribute a stimulus is the required change.
Responses	Measurable or observable quantities that are measured or observed in the quality requirements (attributes) desirable in the architecture. They can be internal (the measures observed in the components of the architecture) or predicted external (they can possibly occur in the final system). Example: the number of modules affected by the change.
Architectural decisions	Aspects (components, connectors) and their properties that have a direct impact on achieving attribute responses. Example: the encapsulation mechanisms

Table 5. The ABAS quality attribute information

The main purpose of every ABAS is to organize consistently the existing body of knowledge in each of the quality attributes communities. This knowledge can be reused in every ABAS related to a particular quality attribute. Table 6 [KK 99] shows the ABAS information structure.

Structure	Description
1. Problem description	<i>Informal description of the design and analysis problem</i> that the ABAS is intended to solve, including the quality attribute of interest or whose presence is desirable in the architectural style, the context of use, constraints and relevant attribute-specific requirements.
2. Stimulus/Response attribute measures	A characterization of the stimuli to which the ABAS is to respond and the quality attribute measures of the response. Construction of the <i>quality model</i> for the attribute
3. Architectural style	Description of the architectural style in terms of its components, connectors, properties of those components and

connectors, and pattern of data and control interactions (their topology) and any constraints on the style. Description the of *architectural decisions*.

4. Analysis

Description of how the quality attribute models are formally related to the architectural style and the conclusions about “architectural behavior”. *Establishment of the links* or “tradeoff” points to detect dependencies among the attributes.

Table 6. The ABAS Structure

Notice that the ABAS structure is similar to those proposed in the catalogues of architectural styles or patterns [SG 96], [Bus 96 et al] [Gam et al 95], with respect to Part 1 and 4 of Table 6. The main difference consists in adding the information on the characteristics of the quality attribute relevant to the particular style, expressed in Part 2 of Table 6. These are the measures of the responses and constitute the quality model for the attribute. Moreover, Part 4 of the structure, called “Analysis”, is used to establish the link between the quality model of the attribute, and the measures of the attribute. The aspects discussed in Parts 2 and 4 constitute the reasoning framework for establishing the quality characteristics of the architectural style.

From the above discussion , ABAS can be seen as a reusable design component, providing a quality model for a specific characteristic which is predictable in the context of the application where the particular ABAS will be used. For example, if the modifiability attribute is required, all the ABAS using different forms of the data indirection pattern could be analyzed according to the structure of Table 6.

Relevance of the ABAS approach to architectural design

The ABAS approach considers the analysis of one attribute at a time, for each style. In order to handle the tradeoff points or dependencies among different attributes (performance impacts modifiability, availability impacts security), the ATAM (Architecture Tradeoff Analysis) method has been developed [Kaz et al 98], where ABAS is included as a step of the method, related to the so called “attribute-specific analysis” step. However, we have to point out that neither the ABAS approach, nor ATAM, include guidelines for constructing the quality model, like SQUID or QMOOD. ATAM is a spiral-based analysis and design method that helps to establish the requirements for the system architecture under a multi-views approach, using ABAS to reason about the quality characteristics of the style, to establish the tradeoff points to detect dependencies among the attributes. Another method, due to Jan Bosch [Bos 00], is similar to ATAM. However, it is more general, not being limited to tradeoff points analysis, but like ATAM it does not include explicit guidelines to construct the quality model. Both methods could be easily incorporated into generic software development processes like MOSES, SOMA [Gra 95] or RUP [Kru 99], which are mostly risk and architectural-based methods. Since this interesting subject is outside the scope of this paper, it will not be considered here, and it will be the object of future research.

Conclusions on the ABAS approach

From the above discussion, the following general considerations on the ABAS approach can be derived:

- It considers the context or problem domain
- It considers only the design stage, for a particular architectural style
- It considers a quality model for one attribute
- It predicts the behavior of an architectural style with respect to one attribute

COMPARISON OF THE QUALITY MODEL-BASED APPROACHES

In the previous sections, we have described and discussed three quality model-based approaches that can be used to improve the design of the system architecture. On the basis of the above discussion, the following criteria for comparing the three approaches have been selected:

1. *Terminology used:*
 - 1.1 *External quality.*
The three approaches studied consider external quality properties, using different terminology. Each term used will be precised
 - 1.2 *Internal quality.*
The three approaches studied consider internal quality properties, using different terminology. Each term used will be precised
 - 1.3 *Link for relating internal with external characteristics*
The three approaches “link” internal or measurable properties with external or high level properties. The “linking process” consists in analyzing the tradeoff points or dependencies among the system’s required quality or external properties against the internal or measurable properties. Each term used for defining this link will be precised
2. *Application of the model*
The stage or stages of the development process where the quality model is applied, and whether the model considers only one or several quality properties, will be precised.
3. *Guidelines for constructing the quality model*
The presence or absence of guidelines will be precised
4. *Construction style of the quality model*
The strategy used in the construction of the quality model will be precised
5. *Tool supporting the approach or the application of the guidelines to construct the quality model*
The presence (name) or absence of the tool will be precised
6. *Relevance to architectural design*
We will be speaking of low, medium or high relevance, meaning a poor, medium or high support to the process for a quality design of the system architecture

These criteria will be presented as entries in Table 7, showing the results of the study for each one of the three approaches to quality studied in the previous sections.

Criteria	ISO 9126	Dromey	ABAS
1. Terminology used:			
1.1 External quality	External characteristic or sub-characteristic refined into attributes (measurable)	High-level attribute (intangible property)	Attribute, responses (measurable)
1.2 Internal quality	Internal sub-characteristic refined into attributes (measurable)	Tangible product property (measurable)	Attribute, responses (measurable)
1.3 Link internal with external characteristics (tradeoffs)	Link or collate	Link	Analysis (qualitative or quantitative semiformal reasoning)
2. Application of the quality model	To the whole process	To each stage	To the design stage

3. Guidelines for constructing the quality model	Not in ISO 9126; present in ISO 14598-3	General method	None
4. Construction style of quality model	Top-down	Bottom-up	Bottom-up
5. Supporting tool	SQUID	QMOOD++	None
6. Relevance to architectural design	Medium	Medium	High

Table 7. Comparison of ISO-9126, Dromey and ABAS approaches

CONCLUSION

The three approaches that have been discussed in the previous sections, according to the criteria defined in Table 7, use the same terminology, only the names of the terms change, showing the lack of a unified language. They share the fact of considering that the quality characteristics wanted or expected (high-level quality characteristics) in a software product must be defined and quantified (measured) in order to be assured. External and internal quality views are considered. The high-level characteristics, that may affect the exit or failure of the final system, cannot in general be directly measured. They must be “refined” in order to get the measurable aspects. Moreover, these measures are used to link or relate the low-level characteristics, which are measurable, with the high-level characteristics. In this way, a tradeoff to detect the dependencies among these characteristics is established. The definition of these links is always performed empirically or on the basis of experience.

On the other hand, the approaches differ mostly on the stage of development where the quality model is applied. However, an important issue is that at design stage, all the approaches could be used. From our point of view, this stage is very important because it concerns the definition of the system architecture, characterized by non-functional properties. Nevertheless the ABAS approach, specific to this stage, does not offer any guideline. We are now studying the use of ISO 9126 or Dromey’s approach to support the extension of the ABAS structure with the construction of the ABAS quality model for the attribute. In this way a possible cooperation of the two approaches is favored.

Finally, an important research issue is the extension of the software development methods that do not consider explicitly a quality model, with one of the three quality model approaches studied. Those offering guidelines should be better candidates, or the use of an extended ABAS with ISO 9126 or Dromey’s design model. Moreover, since these approaches lack a common language, the specification of the quality models studied using notational standards, such as UML (Unified Modelling Language) [RSC 97] should be considered. In [SR 98], UML is used to model architectures of real-time systems, where the selection of an architecture meeting precise quality requirements is crucial.

REFERENCES

- [BC 91] Bass L., Coutaz J, “Developing Software for the User Interface”, Addison Wesley, SEI series, 1991
- [BD 97] Bansija J., Davis C. “An Object-Oriented Design Assessment Model”, PHD Dissertation, University of Alabama, Spetember 1997.
- [Bøe et al 99] Bøeg J., DePanfilis S., Kitchenham B., Pasquini A. “A Method for Software Quality Planning, Control and Evaluation, IEEE Software, 69-77, March/April 1999
- [Bos 00] Bosch J. “Design and Use of Software Architecture”, Addison Wesley, Harlow, England, 2000
- [Bus et al 96] Buschmann F., Meunier R., Rhonert H., Sommerlad P., Stal M. "Pattern-Oriented Software Architecture. A System of Patterns", John Wiley & Sons, 1996.
- [Dro 96] Dromey, R., “Cornering the Chimera”, 33-43, IEEE Software Vol. 13, No.1, January 1996

- [Chi 99]. Chirinos L., "Evaluación de la calidad del software en un proceso orientado a objetos", Magister Thesis, Posgrado en Ciencias de la Computación, Universidad Central de Venezuela, July 1999
- [Gam et al 95] Gamma E., Helm R., Johnson R., Vlissides J. "Design Patterns. Elements of Reusable Object-Oriented Software" Addison Wesley Publishing Co., 1995.
- [Gra 95] Graham I "Migrating to Object Technology", Addison Wesley, 1995
- [ISO 91] ISO/IEC IS 9126: "Information Technology-Software product evaluation: quality characteristics and guidelines for their use", 1991.
- [ISO 98a] ISO/IEC FCD 9126-1.2: Information Technology-Software Product Quality. Part 1: Quality Model, 1998.
- [ISO 98b] ISO/IEC 14598-3. Information Technology - Software Product Evaluation - Part 3: Process for Developers. Software Engineering. June, 1998.
- [Kaz 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
- [KK 99] Klein M., Kazman R., "Attribute-Based Architectural Styles", CMU/SEI-99-TR-022, ESC-TR-99-022, October 1999.
- [Kru 95] Krutchen P. "The 4+1 Model of Architecture, IEEE Software, 12(6):42-50, 1995
- [Kru 99] Krutchen P. "The Rational Unified Process" Addison Wesley, 1999
- [LC 99] 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
- [LL 99] Levy N., Losavio F. "Analyzing and Comparing Architectural Styles" Proceedings of the XIX IC CCSS, (87-95), November 11-13, Talca, Chile, 1999
- [LM 96] Losavio F., Matteo A., "Object-Oriented User-Interface Design Based On Agents Frameworks", Proceedings of TOOLS USA '96, Santa Barbara, California, August 1996.
- [LM 97] Losavio F, Matteo A. "A Method for User-Interface Development", Journal of Object Oriented Programming, Vol.10, 5 (22-27) (1997)
- [McCRW 77] McCall J.A., Richards P.K., Walters G.F. "Factors in Software Quality" Vols 1, 2, 3, AD/A-049-015/055. Springfield, VA: National Technical Information Service, 1977.
- [Pre 95] Pree W. "Design Patterns for Object-Oriented Software Development", Addison Wesley, 1994.
- [SG 96] Show G., Garlan D. "Software Architecture. Perspectives on an Emerging Discipline", Prentice Hall, New Jersey, 1996
- [RSC 97] UML Unified Modeling Language, V1.0, Rational Software Corporation, 1997
- [SR 98] Selic B., Rumbaugh J., Using UML for Modeling Real-Time Systems, Rational Software Corporation, 1998.