Journal of Computer Science 2 (4): 373-381, 2006
ISSN 1549-3636
© 2006 Science Publications

A New Software Quality Model for Evaluating COTS Components

Adnan Rawashdeh and Bassem Matalkah
IT Department, Faculty of Science, Mutah University, P.O. Box 7, Mutah 61710, Karak, Jordan

Abstract: Studies show that COTS-based (Commercial off the shelf) systems that are being built recently are exceeding 40% of the total developed software systems. Therefore, a model that ensures quality characteristics of such systems becomes a necessity. Among the most critical processes in COTS-based systems are the evaluation and selection of the COTS components. There are several existing quality models used to evaluate software systems in general; however, none of them is dedicated to COTS-based systems. In this contribution, an analysis study has been carried out on several existing software quality models, namely: McCall's, Boehm, ISO 9126, FURPS, Dromey, ISO/IEC TR 15504-2 1998(E), Triangle and Quality Cube, for the purpose of evaluating them and defining a ground to build a new model specializing in evaluating and selecting COTS components. The study also outlines limitations found in the existing models, such as the tendency to ignore a certain quality feature like Functionality or the failure to describe how the quality measurement in these models has been carried out. As a result of this analysis, a new model has been built that supports a standard set of quality characteristics suitable for evaluating COTS components, along with newly defined sets of sub-characteristics associated with them. The new model avoids some of the limitations found in the existing models. The new model ignores quality characteristics that are not applicable to COTS components and is empowered with new ones that are. In addition, it matches the appropriate type of stakeholders with corresponding quality characteristics; such a feature is missing in all existing models. The objective of the new model is to guide organizations that are in the process of building COTS-based systems to evaluate and choose the appropriate products, and that is essential to the success of the entire system.

Key words: COTS, Stakeholders, ISO 9126 Model, Dromey Model, McCall's Model, Boehm's Model, FURPS Model, and Quality Model

INTRODUCTION

Over the past decade, the use of commercial off-the-shelf (COTS) products to implement significant portions of a software system has grown in both government and industry. The use of COTS products emphasizes buying commercial capabilities rather than building unique ones from scratch. Organizations that adopt a COTS-based system approach generally expect either more rapid or less costly system construction. These organizations also hope to stay in step with the technological advancements happening in the competitive marketplace. Government organizations are particularly encouraged to use COTS products by acquisition reform regulations. The use of COTS products can indeed have a beneficial effect. For example, NASA (National Aeronautics and Space Administration) successfully employed COTS products in reengineering the Hubble Space Telescope Command and Control system[1].

Quality is a functional and artistic measurement used, for instance, to specify user satisfaction with a product, or how well the product performs compared to similar products. A model is an abstract form of reality, enabling details to be eliminated and an entity or concept to be viewed from a particular perspective. There are all kinds of models: cost estimation models, quality models, maturity models, etc. Models can be presented in different ways, such as in the form of equations, functions or diagrams. This makes it possible to show how components are related, so they can be examined, their relationships understood and opinions formed. This is one reason why a quality model has become essential for ensuring that a firm product and process meets customers' needs.

In the literature of quality models, the reader can observe that different authors have proposed different models, for example: McCall's quality factors proposed in 1976, Barry Boehm's quality model presented in 1978, FURPS in 1987, ISO (International Standard Organization) proposed the quality attribute ISO 9126 in 1991, and Dromey model in 1996. Such models are intended to evaluate the quality of software in general; none of the models is specialized in or dedicated for COTS-based systems. Thus it is likely they include characteristics that are not necessarily applicable to COTS components.

Corresponding Author: Adnan Rawashdeh, IT Department, Faculty of Science, Mutah University, P.O. Box 7, Mutah 61710, Karak, Jordan

Using COTS software products in large systems provides many benefits, including the potential of rapid delivery to end users, shared development costs with other customers, reusability of the final application due to the reuse of software components already tested and validated, and the opportunity to expand capacity and performance as improvements are made in the products. For systems that depend on COTS products, the evaluation and selection of appropriate products is essential to the success of the entire system. Yet many firms struggle during the evaluation and selection process. In[2] Dean, John et al define a COTS product as one that is (i) sold, leased, or licensed to the general public, (ii) offered by a vendor trying to profit from it, (iii) supported and evolved by the vendor, (iv) available in multiple identical copies, or (v) used without modification of the internals.

Software development is increasingly becoming an "acquire and glue" process. How do you know when you can trust a COTS component to do what you expect it to, in your system? Obviously, software certification is viable. In 1997 an estimated 25.5% of a typical corporation's software portfolio was COTS software. Forecasts had that figure rising in 1998 to around 28.5% and exceeding 40% in the subsequent years[3].

A quality model has become an important requirement to avoid purchasing COTS of questionable quality. A starting point for a quality model might be to consider some of the existing quality approaches. In[3], Jeffrey also suggested the software certification triangle, software certification might be one or more of three approaches:

* accrediting developers for demonstrating specific skill sets
* assessing the codes behavior, and
* certifying that processes are properly followed.

These are shown in Fig. 1 below.

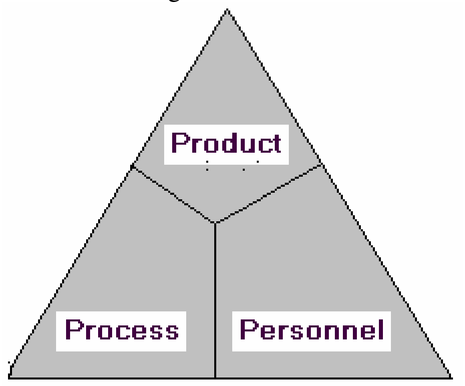

Fig. 1: The software quality certification triangle[3]

The McCall's, Boehm's, FURPS, ISO/IEC 9126, ISO/IEC TR 15504-2 1998(E) and Dromey's are called hierarchy models, whereas Triangle and Quality Cube are called non-hierarchy models. All of the above models have been studied, and analyzed with the purpose of developing a new model that: (i) overcomes some of the existing limitations, (ii) features standard quality characteristic and associated sub-characteristics integrated from hierarchies and non-hierarchical models in an attempt to combine the advantages of both types, and (iii) associates all categories of stakeholders involved in the process with the appropriate set of quality characteristics. As a result, the new model will be utilized as a tool for evaluating and selecting the appropriate COTS product.

Analysis study of selected hierarchy and non-hierarchy quality models: The choice of existing models to be evaluated here was based on the most standard and well-known quality models. Among the hierarchy the following models have been described:

McCall's model for software quality combines eleven criteria around product operations, product revision, and product transitions. These include: Correctness, Reliability, Efficiency, Integrity, Usability, Maintainability, Testability, Flexibility, Portability, Reusability and Interoperability.

McCall's Model is used in the United States for very large projects in the military, space, and public domain. It was developed in 1976, by the USAir-force, ESD (Electronic System Decision), RADC (Rome Air Development Center) and GE (General Electric), with the aim of improving the quality of software products[4].

The main idea behind McCall's model is to assess the relationships among external quality factors and product quality criteria. External quality is quality as measured by the customers; internal quality is quality as measured by the programmers[5,6]. McCall's model is based on three aspects of software: product operation refers to the product's ability to be quickly understood, product revision is related to error correction, and system adaptation, product transition is related to distributed processing, together with rapidly changing hardware, is likely to increase its importance.

One of the major contributions of the McCall model is the relationship created between quality characteristics and metrics, although there has been criticism that not all metrics are objective. One aspect not considered directly by this model was the Functionality of the software product[7].

Boehm added some characteristics to McCall's model with emphasis on the Maintainability of a software product. Also this model includes considerations involved in the evaluation a of software product with respect to the utility of the program. However it is similar to McCall's in that it presents a hierarchy of characteristics, each of which contributes to overall quality. His model is based on a wider range of characteristics and incorporates 19 criteria. It has been noted that Boehm's notation of successful software includes characteristics of hardware performance that are missing in McCall model[8].

Boehm's model looks at utility from various dimensions, considering the types of user expected to work with the system once it is delivered. General utility is broken down into Portability, Utility and Maintainability. Utility is further broken down into Reliability, Efficiency and Human Engineering. Maintainability is in turn broken down into Testability, Understandability and Modifiability.

However, Boehm's model contains only a diagram without any suggestion about measuring the quality characteristics.

The FURPS model proposed by Robert Grady and Hewlett-Packard Co. decomposes characteristics into two different categories of requirements:

* Functional requirements: Defined by input and expected output.
* Non-functional requirements: Usability, Reliability, Performance, and Supportability.

FURPS takes into account the five characteristics that make up its name: Functionality, Usability, Reliability, Performance, and Supportability. One disadvantage of the FURPS model is that it fails to take into account the software product's Portability[7].

In reply to the software industry's need to standardize the evaluation of software products using quality models, the ISO (International Standard Organization) proposed a standard, which specifies six areas of importance for software evaluation, and, for each area, specifications that attempt to make the six areas measurable, these include: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. One of the advantages of the ISO 9126 model is that it identifies the internal and external quality characteristics of a software product. On the other hand, it does not show very clearly how these aspects can be measured[7].

Dromey proposed a model consisting of eight high-level quality attributes, namely the same six from ISO 9126 plus Reusability and Process Maturity[9,10]. He suggested a more dynamic idea for modeling the process on three prototypes concerning quality. These are:

* implementation quality model
* requirements quality model, and
* design quality model. Dromey's model seeks to increase understanding of the relationship between the attributes (characteristics) and the sub-attributes (sub-characteristics) of quality. It also attempts to pinpoint the properties of the software product that affect the attributes of quality[7]. The disadvantage of the Dromey model is associated with Reliability and Maintainability. It is not feasible to judge both attributes Reliability and Maintainability of a system before it is actually operational in the production area.

Among the non-hierarchy, Triangle and Quality Cube models have been described. In[3] Jeffrey Voas suggested that software quality certification might take one or more of three approaches: (i) accrediting developers for demonstrating specific skills sets, (ii) assessing the code behavior, and (iii) certifying that processes are properly followed. The three distinct approaches are shown in Fig. 1 above. Although you can approach software certification in any one of these ways, a combination of the three provides a more balanced approach. This is preferable, since any one of the three can be inadvertently misapplied[3].

In[11] Nagib et al discussed the 'totality of quality' concept in information system design. He stated that the conventional design in software engineering is initiated with a set of given requirements and the task of the designer is to find the most efficient way to satisfy these given requirements. This kind of design could be called 'efficiency-design'. However, no matter how efficient the system design is, it won't be useful if the given requirements are not the real requirements of the system users. So the design should also be effective, and this is the essential characteristic of the 'total quality movement' and the new methodologies that are emerging in software engineering (such as prototyping, and joint application design) the requirements are to be defined by the users of the system. The designer should design the right system, in addition to designing the system right. This is the essence of the 'total quality movement'. The meaning of the term 'design' includes both senses: the design as a product and the design as a process. The system designed (the product) is different to the system of human activities (the process) through which the product-system is being designed. Both product and process could be objective of efficiency-design and effectiveness-design. There are four kinds of qualities: product-efficiency, product-effectiveness, process-efficiency and process-effectiveness. Putting these together, Fig. 2 shows our own illustration of the Quality Cube model described in[11].

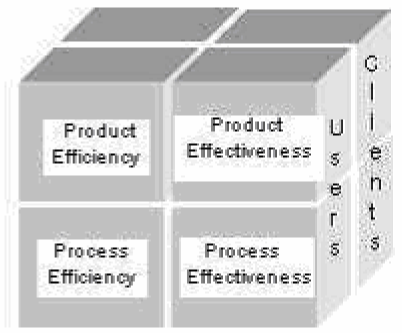

Fig. 2: The quality cube model

The disadvantage of such non-hierarchy models is the lack of identifying sub-attributes for the associated high level attributes. In addition, it is not clear how to measure the quality of those attributes.

The quality characteristics found in the majority of the models are: Efficiency, Reliability, Mainability, Portability, Usability and Functionality, these have been present in more recent models, as described above. In order to examine, compare and come to a conclusive result, we have prepared Table 1 below. The table shows six hierarchy models among the ones that have been studied here, each associated with its software quality characteristics.

With this tabular illustration, it becomes easy to observe the models that support a wider range of characteristics in comparison to the ones that support fewer features. For example, it can be seen clearly that ISO 9126 and McCall support more characteristics than Boehm, FURPS and Dromey. Furthermore we can observe the common characteristics that are supported by almost all of the models, namely: Efficiency, Reliability, Functionality, Mainability, Portability, and Usability; because of this general agreement, the six characteristics can be observed as the standard requirements for any new model. This is a useful conclusion to be considered in our new model. Therefore, the general agreement on the six characteristics, in addition to their applicability for COTS components, supports our decision to specify them as the standard requirements for our new model. Table 1 also facilitated the projection of the most appealing model among the existing ones, and that is the ISO 9126. Therefore making the ISO 9126 our starting point for building the new model with respect to the others is a justifiable decision.

Addressing the problem: There is no general consensus on the quality model which can fit into all types of applications, considering the different classifications: McCall's, Barry Boehm's, ISO 9126, FURPS, Dromey, Triangle, and Quality Cube models. McCall's ignored Functionality, Boehm's contains a diagram without any suggestion about measuring the quality characteristics, FURPS ignored Portability, ISO-9126 does not show very clearly how the attributes can be measured.

Thus, there is an absence of any kind of metrics that could help in evaluating quality characteristics objectively, in particular when the underlying software project is a COTS component. In addition, none of the existing models attempts to relate certain characteristic with the type of stakeholders that are most concerned with such characteristic.

Among the available models for evaluating software quality, none of McCall's, Boehm's, FURPS, ISO/ICE, and Dromey explicitly consider process-efficiency and process-effectiveness. An efficient product is obtained when correct physical design and programming practices are applied; product-effectiveness is determined by activities involving requirement identification, interface design and general network design. Process-efficiency is associated with project management activities which include meeting deadlines, increasing productivity and saving resources, process-effectiveness is related to general management activities such as leadership, change management, human and group relations, as these lead to good relations between the members of the team responsible for developing information systems[12,13].

ISO 9126 identifies the external quality characteristics of a software product. Therefore it represents product-effectiveness. Dromey's model identifies internal properties[9,10], which specify four areas: Correctness, Internal, Contextual and Description. Therefore, Dromey represents product-efficiency. Process-effectiveness and process-efficiency are not presented in Dromey's model nor in ISO 9126; however the solution is proposed in a reference model called ISO/IEC TR 15504-2. (IEC stands for International Electro-technical Commission). As the ISO/IEC model described in[14], it groups the processes into four categories, so that each category is associated with a set of processes. These are shown in Table 2.

SUGGESTED METHODOLOGY

Steps that are used to build a quality model prototype based on ISO 9126 and Dromey's model.

Step 1: Identify a small set of agreed-upon, high-level quality attributes, and then, in a top-down fashion decompose each attribute into a set of subordinate attributes.

Step 2: Distinguish between internal and external metrics. For COTS components, it is essential to observe such distinctions, specifically; the internal metrics measure the internal attribute of a product (e.g. specification or source code) during the design and coding phases, known as 'white box' metrics[15]. Whereas external metrics specialize in the system behavior during testing and component operation, from an outsider view. In fact, external metrics, known as 'black-box', are more appropriate for COTS components.

Step 3: Identify Stakeholders (type of users) for each high-level quality attribute.

Step 4: Put the pieces together; constructing the new model that implement ideas from international standards: ISO-9126, Dromey, ISO.IEC TR 15504-2, and accordingly recognize appropriate Stakeholders for each set of attributes.

Executing the methodology to build the new quality model: The objective of creating our new model is to build one suitable to work for a variety of COTS-based systems. The starting point for building our model is the ISO 9126, simply because it includes the common software quality characteristics that are supported by the other six models, as shown in Table 1.

Table 1: Quality characteristics in Boehm, McCall, FURPS, ISO 9126 and dromey models

Software Quality	Boehm	McCall	FURPS	ISO 9126	Dromey
Testability	X	X		X	
Correctness		X			
Efficiency	X	X	X	X	X
Understandability	X			X	
Reliability	X	X	X	X	X
Flexibility		X	X		
Functionality			X	X	X
Human Engineering	X				
Integrity		X		X	
Interoperability		X		X	
Maturity					X
Mainability	X	X	X	X	X
Changeability	X				
Portability	X	X		X	X
Reusability		X			X
Usability		X	X	X	X

Table 2: Process categories in ISO/IEC TR 15504-2 1998(E) model

Category	Processes
Customer-Supplier	System or Product Acquisition Process, Supply Process, Requirement Bidding Process, Operation
Engineering	Development, Software and System Maintenance
Support	Documentation, Configuration Management, Quality Assurance Process, Verification, Validation, Joint Review, Auditing, Problem Solving Process
Management	Management, Project Management, Quality Management, Risk Management
Organizational	Organizational Alignment, Change Management, Improvement, Infrastructure Process, Measurement Process, Re-use Process

Table 3: ISO 9126 Quality Characteristics

Characteristics	Sub-characteristics
Functionality	Suitability, Accuracy, Interoperability, Compliance, Security
Reliability	Maturity, Recoverability, Fault tolerance
Usability	Learnability, Understandability, Operability
Efficiency	Time behavior, Resource behavior
Maintainability	Stability, Analyzability, Changeability, Testability
Portability	Installability Conformance, Replaceability, Adaptability,

The next step is to apply some tailoring on the ISO 9126 that harness COTS evaluation requirements.

The six areas of importance for software evaluation, as proposed by ISO 9126 as a standard, are shown in Table 3.

The following is the evaluation discussion of the high-level set of characteristics, along with their associated sub-characteristics; the implementation of step 1 of our methodology:

Functionality is the capability of the software product to provide functions, which meet stated and implied needs when the software is used under specified conditions. Functionality is a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs. Functionality is assessed by three things: (i) evaluating the set of features and capabilities of the program, (ii) the generality of functions that are delivered, and (iii) the security of the overall system[15]. The sub-characteristic Compatibility has been added to our model. The purpose of Compatibility is to reflect the degree to which a component can be used and function correctly in different environments, and that is consistence with evaluating COTS components.

Reliability is the capability of the software product to maintain a specified level of performance when used under specified conditions[16]. Reliability is the extent to which a program can be expected to perform its intended function with required precision[17], usually evaluated by measuring the frequency and severity of failure, the accuracy of output result, the mean time between failures (MTBF), the ability to recover from failure and the predictability of the program[18], because unreliable programs fail frequently, or produce incorrect data[19]. Also, Reliability is 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[20]. Reliability is the degree to which a work product operates without failure under given conditions during a given time period.

The Maturity sub-characteristic is measured in terms of the number of commercial versions and the time interval between them. The Recoverability sub-characteristic is a capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of failure. Therefore, it tries to measure whether the component is able to recover from unexpected failures, and how it implements these recovery mechanisms. The Fault

Tolerance tries to measure the capability of the software product to maintain a specified level of performance in cases of software faults. The COTS-based system that supports Recoverability feature is in a subsequent stage of passing a Fault Tolerance stage, thus Recoverability implies Fault Tolerance and not vice versa. For this reason Fault Tolerance is omitted from our model, while the emphasis remains on Recoverability.

Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. Usability is related to the 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. In addition, Usability is the effort required to learn, operate, prepare input, and interpret output of a program[15]. In COTS, most stakeholders of components are the application developers, designers that have to build applications with them, and end-users who interact with COTS. Thus, the Usability of a component should be interpreted as its ability to be used by the application developer and designer when constructing a new software product. Under this characteristic we must add an attribute that measures the component's Usability during the design of application. Therefore, Complexity is a new sub-characteristic that is added to provide a measure of the components complexity when integrating and using it within a software product or system. This characteristic aims to measure the complexity of using and integrating the component into the final system.

Efficiency is the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. Efficiency is the degree to which something effectively uses (i.e., minimizes its consumption of) its resources. These may include all types of resources such as computing (hardware, software, and network), machinery, facilities, and personnel[15]. In fact, Efficiency will used in our new model as it is described in the ISO.

Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to change in an environment, and in requirements and functional specifications[16]. Also, the effort required to locate and to fix an error in an operational program[17,18]. Maintainability is the ease with which an application or component can be maintained between major releases. Also, a set of attributes that bear on the effort needed to make specified modifications[20], the degree of changing or modifying the components to correct errors, to improve performance, or to adapt for changing the environment. The user of a component (i.e. the developer) does not need to do the internal modifications but he does need to adapt it, re-configure it, and perform the testing of the component before it can be included in the final product. Thus, Stability and Analyzability are omitted from our model.

Portability is the capability of the software product to be transferred from one environment to another[16]. Also, the effort required to transfer a program from one hardware configuration and/or software system environment to another[17,18]. Portability is the ease with which an application or component can be moved from one environment to another[20,21]. In COTS, Portability is an important property in the nature of components, which are in principle designed and developed to be re-used in different environments (it is important to note that in COTS, re-use means not only to use more than once, but also to use in different environments. Thus, Portability is omitted from our model.

Manageability; in order to empower our model with new a feature, the characteristic Manageability has been added. Manageability is concerned with developing and refining estimates of effort and deadlines for the project as a whole, and with gathering any data that might be needed for such estimates. We have added to our model the sub-characteristics Quality Management, which indicates the people within the organization, who are constantly monitoring what they do to find ways to improve quality of operation, product, budgets, schedule, services, and everything else about the firm. Table 4 shows the quality model we propose for selecting COTS components.

The second step in the proposed methodology, distinction between internal and external metrics, is already described and reasoning led us to consider the external metrics 'black-box' as more appropriate for COTS components.

Stakeholders: The term stakeholder is used to refer to any person or group who will be affected by the system, directly or indirectly. Stakeholders include the end-user who interacts with the system and everyone else in an organization that may be affected by its installation. Other system stakeholders may be engineers who are developing or maintaining a related system, and business managers[21]. From our experience, end-users are concerned with observable attributes (such as Functionality, Reliability, Availability, and Efficiency). BO (business owner) is concerned with Maintainability, while system administrators are concerned with Scalability, Portability, and Manageability.

In this work, a typical set of stakeholders as explained in[22] has been adopted in order to name the appropriate category of evaluators for each quality characteristic. We start with the analyst who produces the business model, the end-user who interacts with the system, QA officer (quality assurance) who tests the product, and the PM (project manager) who constructs and manages the process.

* the solution verifiability satisfies the requirement, both functional and non-functional, this should be verifiable by the analysts and the QA professionals.

Table 4: Quality Model for COTS Components

Characteristics	Sub-characteristics (Product)	Sub-characteristics (Process)
Functionality	Accuracy, Security	Suitability, Interoperability, Compliance, Compatibility
Reliability	Recoverability	Maturity
Usability		Learnability, Understandability, Operability, Complexity
Efficiency	Time behavior, Resource behavior	
Maintainability		Changeability, Testability
Manageability	Quality management	

Table 5: Tabular illustration of the new model components

Stakeholders (Professional Parties)	Characteristics	Product Sub-characteristics	Process Sub-characteristics
End user, analysts, quality assurance	Functionality	Accuracy, Security	Suitability, Interoperability, Compliance, Compatibility
End user, analysts, quality assurance	Reliability	Recoverability	Maturity
End user, analysts, quality assurance	Usability	{Non Applicable}	Learnability, understandability, operability, complexity
End user, analysts, quality assurance	Efficiency	Time behavior, Resource behavior	{Non Applicable}
Project manager or business owner	Maintainability	{Non Applicable}	Changeability, Testability
Project manager	Manageability	Quality management	Quality management

* the solution is verifiable by other architects, who can evaluate trade-offs and determine its fitness as a solution to the problem. This implies clearly stating the system goals.
* the developers can build the solution. This implies partitioning the solution into comprehensible pieces, with clear interface and definitions, and explicit mapping of dependencies among pieces.
* the product can be tested by QA. This relies on the mentioned partitioning (to plan unit testing) and traceability (to verify deployed functionality and properties).
* the process can be managed by PM. This relies on partitioning (to determine work units for teams and individuals) and on dependencies (to schedule work); thus, the project manager must be able to determine "intermediate deliverables" that are usable, testable and allow to show working progress.

The domain of the above classification of stakeholders can be re-organized as follows; which implements the third step of our methodology:

* the solution must offer the Functionality, observable attributes (Reliability, Usability, and Efficiency) specified requirements according to end-users, verified by analysts and QA.
* the solution must be maintainable according to the future PM, verified by the BO.
* the construction process must be manageable according to the project manager.

Table 5 shows the components that constitute our new model. Consequently, we have adapted to our model the common characteristics that are found and agreed upon by the majority of the existing models, and these are consistent with COTS component evaluation criteria. However, we did omit some of the characteristics that are inconsistent with the new model requirements. New characteristics are added, and these are necessary to empower our new model. Accordingly, any modification step, including removal or additions has been justified above.

Next, a new set of sub-characteristics has been defined and associated with each high-level characteristic that is supported by the new model, this was done by breaking down the characteristics into two categories; one set supports the development process (the process) and the second one supports the operational state on the production area (the product).

Finally, stakeholders, the members of the team responsible for developing, maintaining, interacting with and/or using the information system have been categorized then matched accordingly with the appropriate characteristics throughout the entire system development life-cycle, including operational and maintenance phases.

Figure 3 shows the final structure of the new model, containing all the associated components, which implements the fourth step of our methodology:

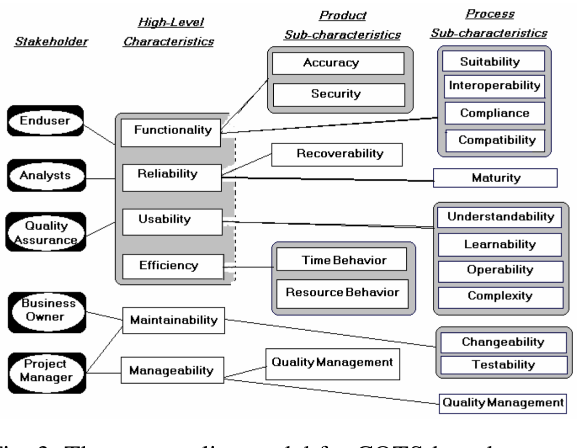

Fig. 3: The new quality model for COTS-based systems

CONCLUSION AND RECOMMENDATIONS

The number of COTS-based systems being built continues to increase. Consequently, the need for a model that ensures quality characteristics of such systems becomes a necessity.

Several models, including hierarchy and non-hierarchy, specializing in measuring the quality of software products have been described. The features of such models have been studied, analyzed and their limitations outlined. Specifically, Functionality of a software product was not considered directly by McCall's model. No suggestion about measuring the quality characteristics has been found in Boehm's model. FURPS model fails to take account of the software product's Portability. ISO 9126 has the limitation of not showing very clearly how certain quality aspects can be measured. The disadvantage of Dromey's model is associated with Reliability and Maintainability. It is not feasible to judge these two attributes of a system before it is actually operational in the production area. The disadvantage of non-hierarchy models, Triangle and Quality Cube, is the failure to identify sub-attributes for the associated high level attributes. In addition, it is not clear how to measure the quality of those attributes.

Among all the existing models that have been studied, we found the ISO 9126 is the most appealing model, irrespective of some limitations. For this reason, we based our new model on the ISO 9126. We defined a four-step methodology to guide the process of building the new model that is specialized in evaluating COTS components. The analysis step assisted us to benefit from existing general quality models and simultaneously avoiding repetition of such limitations.

Subsequently, justified high-level characteristics have been projected and a new set of sub-characteristics has been defined for each one. This is accomplished by breaking down the characteristics into two categories; 'the process' and 'the product'.

The distinction between internal and external metrics led us to realize that external metrics is more appropriate for COTS components.

A major advantage of the new model is the addition of stakeholders, the members of the team responsible for developing, maintaining, interacting and/or using the COTS-based system. End-users, analysts, QA, PM and BO categories are matched with appropriate characteristics that each one is concerned about.

Finally, the pieces are put together to construct the new model. Although our proposed model features specialization and improvement over existing models, it lacks the ability to measure the internal quality characteristics. This can be accomplished in future research work by applying one of the evaluation techniques such as AHP (Analysis Hierarchy Process).

REFERENCES

1. Pfarr, T. and J.E. Reis, 2002. The integration of COTS/GOTS within NASA's HST command and control system. Proc. First Intl. Conf. COTS-Based Software System, Systems, Orlando, FL, Feb. 4-6,. In Lecture Notes in Computer Science 2255, Berlin: Springer-Verlag, pp: 209-221.
2. John, D., G. Lewis, E. Morris, P. Oberndorf and E. Harper, 2004. A Process for COTS Software Product Evaluation. Technical Report CMU/SEI-2003-TR-017, ESC-TR-2003-017.
3. Jeffrey, V., 1999. Certification: Reducing the hidden costs of poor quality. Reliable Software Technologies, IEEE Software, 0740-7459/99.
4. Ronan, F., 1996. Software quality definitions and strategic issues. Staffordshire University, School of Computing Report, http://www.comp.dit.ie/rfitzpatrick.
5. Kent, B., 1999. Extreme Programming Explained: Embrace Change. Addison Wesley Professional.
6. Lisa, C., 2001. Is quality negotiable? STARWest 2001 Conference. citeseer.ist.psu.edu/crispin01is.html
7. Maryoly, O., M.A. Perez and T. Rojas, 2002. A systemic quality model for evaluating software products. Laboratorio de Investigcin en Sistemas de Informacin, 2002, http:/www.lisi.usb.ve/publicaciones.
8. Nihal, K. and A. Abran, 2001. Analyzing Measuring & Assessing Software Quality Within A Logic-Based Graphical Framework, Dept. of Computer Science, Software Engineering Management Research Laboratory (SEMRL), Universite du Quebec a Montreal, P.O. Box 8888, Centre-Ville Postal Station, Montreal (Quebec), Canada H3C 3P8, nkececi@lrgl.uqam.ca.
9. Geoff, D., 1995. A model for Software Product Quality, IEEE Transactions on Software Engineering, 21(2nd): 146-162.
10. Geoff, D., 1996. Cornering the Chimera, Australian Software Quality Research Institute, EE Software 0740-7459/96.
11. Nagib, C.B. De Callaos, 1994. Designing With A Systemic Total Quality, Educational Technology 34:29-36.
12. Rojas, T. and M. Perez, 1995. A Comparison of Three Information System Development Methodologies Related to Effectiveness/Efficiency Criteria, International Symposium on Applied Corporate Computing, (ISACC'95) Monterrey, Mexico.
13. Rojas, T. and M. Perez, 1997. Determination of Factors That Affect The Process Effectiveness In The Development of Information System, Americas Conference on Information Systems, Simon Bolivar University, Processes and Systems Dept., Caracas, Venezuela, P.O.Box 89000, trojas@usb.ve, (AIS'97) Indianápolis, USA.
14. ISO/IEC TR 15504-2 1998(E) Technical Report, (1998), Information Technology – Software Process Assessment – Part 2: A Reference Model for Processes and Process Capability, ISOSTD ISO Template Version 3.3, 1998, Canada.

15. Khashayar, K. and Y.-G. Gueheneuc, 2004. A Quality Model for Design Patterns, Master's Thesis, Laboratory of Open Distributed Systems & Software Engineering, Dept. of Informatics and Operations Research, University of Montreal, C.P. 6128 succ, Centre-Ville, Montreal, Quebec, H3C 3J7, Canada, guehene@iro.umontreal.ca.
16. International Standard. ISO/IEC 9126-1 (2001), Institute of Electrical and Electronics Engineering, Part 1,2,3: Quality Model, 2001, http://www.iso.ch.
17. Gaffney, J.E., 1981. Metrics in Software Quality Assurance. Proc. ACM '81 Conf., pp: 126-130. http://portal.acm.org.
18. Roger, S.P., 1992. Software Engineering A Practitioner's Approach, McGraw-Hill, Inc.
19. Lowell, J.A., 1951. Software Evolution, The Software Maintenance Challenge. John Wiley and Sons.
20. Joc, S. and E. Curran, 1995. Software Quality, A Framework for Success in Software Development and Support. Addison-Wesley Publishing Company.
21. Ian, S., 2004. Software Engineering, Seventh Edition, Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England.
22. Hernan, A. and S. Hammer, 1998. Understanding The Architect's Job: An Opinionated View of What We Do and Why Do it, Hernan; University of Sao Paulo, Brazil, hernan@acm.org. Stuart; Object Practice, MCI Systemhouse, Shammer@shl.com.