Towards a Software Component Quality Model

Nitin Upadhyay[1], Bharat M. Despande[1], and Vishnu P. Agrawal[2]

[1] Computer Science & Information Systems Group, BITS-Pilani Goa Campus,
NH-17 B Zuari Nagar,
Goa, Goa-403726, India
{nitinu,bmd}@bits-goa.ac.in
[2] Mechanical Engineering Department, Thapar University, Research & Development Campus,
Patiala, Punjab-403726, India
vpagrawal@thapar.edu

Abstract. The academic and commercial sectors have noticed the growing impact of the component based software development (***CBSD***), in particular, to develop customizable, cost effective, just-in-time and reusable large scale and complex software systems. To build complete software solution the main focus is to create high quality parts and later join them together. One of the most critical processes in ***CBSD*** is the selection of software component as per end user criteria. Quality model plays an important role in the component selection process. This paper presents a software component quality model (***SCQM***) by overcoming shortcomings of existing quality models. Based upon this end user can take decision upon selection, evaluation and ranking of potential component candidates and wherever possible attain improvements in the component design and development.

Keywords: component, quality, component quality model, component quality characteristics.

1 Introduction

During the last years a significant change is noticed in the paradigm of software development. Modern software's are large scale and complex. Many organizations consider implementing such software using commercial off-the-shelf (***COTS***) components with the expectations that ***COTS*** components can significantly lower development costs, shorten development life cycle, produce high reliable and high stable product [1] . Component based software engineering has emerged as a separate discipline which ultimately leads to software system that require less time to specify, design, test and maintain and yet establishes high reliability requirements [2-5]. The quality of a component based software system depends upon the quality of individual components and interactions among them considering any rationale or constraint for selecting them [6-7]. A software quality model acts as a framework for the evaluation

N. Meghanathan et al. (Eds.): CCSIT 2011, Part I, CCIS 131, pp. 398–412, 2011.

of characteristics of software. It is to be noted that with the development of technology and research, software being used in an increasingly wide variety of application areas and its correct operation is often critical for business success [8-9]. To ensure adequate quality of a product it is necessary to perform comprehensive specification and evaluation of software component quality. This can be achieved by defining appropriate quality model/standard for the component market that will guarantee component acquirer, high quality components [10-11]. Existing quality models are inappropriate to evaluate quality of software components as they are very general in nature. Very few research studies which are available have contributed to the development of component specific quality model. More or less each and every model is the customization of ISO-9126 software quality model.

In this paper, we intend to review existing quality models (general and component specific) in order to propose comprehensive component specific quality mode to evaluate quality of a software component. The rest of the paper is structured as follows: In section 2, the existing quality models are reviewed. Section 3 describes the proposed component specific quality model which overcomes the shortcomings of existing quality models. Finally, conclusion is given which details out contribution made and future directions.

2 Review of Quality Models

The state of art literature does not provide a well established and widely accepted description scheme for assessing the quality of software products [12]. One of the major contributions provided by McCall's model is the consideration of relationships between quality characteristic and metrics [13]. There has been a criticism that not all metrics are objective and the issue of product functionality is not considered. Bohem model [14] is similar to the McCall model in that it represents a hierarchical structure of characteristics, each of which contributes to the total quality. In Bohem model the main emphasis is on the maintainability aspect of a software product. However, Bohem's model contains only a diagram and does not elaborate the methodology to measure these characteristics. Dromey [15] proposed a quality evaluation framework taking into consideration relationship between the attributes (characteristics) and sub-attributes (sub-characteristics) of the quality. This model suffers from lack of criteria for measurement of software quality and it is difficult to see how it could be used at the beginning of the lifecycle to determine the user needs. The FURPS [16-19] quality model covers *Functional requirements and* Non-functional requirements but fails to take into account the software product's maintainability, which may be important criterion for application development, especially for component based software systems (***CBSS***). The Bayesian belief network (***BBN***) is a special category of graphical model, which represents quality as root node and other quality characteristics via directed arrows [20-21]. This model is useful to manipulate and represent complex quality model that cannot be established using conventional methods. However, this model fails to evaluate fully software product due to involvement of lack of criteria. Different perspectives of software quality can be represented by star model. Even though it considers various viewpoints of quality it does not evaluate fully software product due to involvement of lack of criteria. The ISO 9126 [22] is a part of ISO

9000 standard, which is the most important standard for quality assurance. In this model, the totality of software product quality attributes is classified in a hierarchical tree structure of characteristics and sub-characteristics. This model is too generic to be applicable to component based environments.

Bertoa's model [23] is a well known initiative to define the attributes that can be described by ***COTS*** vendors. In the model *portability* and *fault tolerance* characteristics disappear together with the *stability* and *analyzability* sub-characteristics. Two new sub-characteristics: *compatibility* and *complexity* are added in this model. Although this model presents a good description on quality characteristics but *portability* and *fault tolerance* which have been eliminated are very significant to ***COTS*** components. The Alvaro's model [24] is similar to Bertoa's model but provides better footprints as the model has introduced a number of components specific quality characteristics or sub-characteristics like *self-contained, configurability* and *scalability*. The purpose of the model is to determine which quality characteristic should be considered for the evaluation of software component [25]. *Reusability* is important for the reason that software factories have adopted component based approaches on the premise of reuse. The *maintainability* and *analyzability* sub-characteristics have been removed from ISO 9126. A high level characteristics '*Business'* have also been added with following sub-characteristics: *development time, cost, time to market, targeted market, affordability.* Alvaro's model also has some drawbacks. Firstly, *reusability* has been treated as quality attribute rather than quality factor. Secondly, the ambiguous definition of *scalability* which is related to only data volume and not the maximum number of components, the component can interact with other components without reducing performance. In Rawedah's model [26] standard set of quality characteristics suitable for evaluating ***COTS*** components along with newly defined sets of sub-characteristics associated with them were identified. The sub-characteristics *fault tolerance*, *configurability, scalability* and *reusability* have been removed. New characteristic manageability with sub-characteristics *quality management* has been added. The model also attempts to match the appropriate type of stakeholders with the corresponding quality characteristics. It can b easily notice that the sub-characteristics that have been remover are significant to components. It can be clearly seen that still a comprehensive component specific quality model is not available.

3 Proposed Software Component Quality Model (SCQM)

Based on critical literature survey [13-31] we propose component specific quality model (***SCQM***) which overcomes the shortcomings of existing quality models. At the highest level the ***SCQM*** consists of eight characteristics – ***functionality, reliability, usability, efficiency, maintainability, portability, reusability*** and ***traceability***. A unified measure for attributes is taken on a level of satisfaction scale (LOS) from 1 – 5, see table 1. The detail description of the ***SCQM*** characteristics is as follows:

3.1 Functionality

This characteristic indicates the ability of a component to provide the required services and functions, when used under the specified conditions. It directly matches

with the functionality characteristic of ISO 9126 but with a set of different sub-characteristics. More specifically, ***security*** and ***suitability*** sub-characteristic retain their meaning with the only exception that the term ***suitability*** is replaced with the term ***completeness*** as it reflects better scope. Since the components are meant to be reuse, ***interoperability*** is moved under the category of ***reusability*** characteristics (not in ISO 9126). ***Reusability*** is treated as a separate. Also, ***accuracy*** is moved to the ***reliability*** characteristic (***Result set*** sub-characteristic) since we consider Accuracy to be a feature of ***reliability*** instead of ***functionality***. We have included ***self-containment*** which is the intrinsic property of a component. Finally, for the evaluation reason, ***compliance*** has been removed temporarily because currently there are still no official standards to which component must adhere to. Following sub-characteristics contribute to ***functionality***:

- *Self-containment*: ***Self-contained*** is an intrinsic property of a component and it means component is encapsulated with well-defined interfaces and can be executed independently with minimal outside support [27]. Following attributes contribute to Self-containment of a component

  - *Preconditions and postconditions:* Well defined interfaces have contacts that are described by the presence of pre-conditions and postcontitions [27].

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

  - *Modularity*: It indicates the degree of independence of the component, which is the degree of functional cohesion. It can be measured by the ratio of *total number of functions provided by the component itself without external support* to the *total number of functions provided by the component.*

  ***Measure* = (Ratio normalized to LOS)**

- *Security*: It indicates the ability of a component to control the unauthorized access to the services provided to it. It also deals with whether a security failure of a component will result to a system wide security failure. Following attributes contribute to security of a component:

  - *Control access*: It indicates the ability of a component to provide control access mechanisms like ***authentication*** and ***authorization***.

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

  - *Data encryption*: It indicates the ability of a component to provide data encryption mechanisms to secure data it handles.

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

  - *Auditing*: It indicates the ability of a component to keep track of user actions and any unauthorized access.

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

  - *Privilege intensification*: It indicates the ability of a component to identify any flaw in the component which may leads to privilege intensification and hence to system security breach.

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- *Completeness*: It indicates the ability of a component to fit into user (re-user) requirements. This provides the overall idea of the level to which component

covers the needs of the users in terms of services offered. Following attributes contribute to completeness of a component:

- *User Satisfaction*: It indicates the ability of a component to meet re-user requirements by offering services as per re-user needs. Here, re-users are not the component providers but are system developers and integrators.

***Measure* = (LOS)**

- *Service Excitability:* It indicates the ability of a component to provide more than required/expected related service. It can be measured by the ratio of *total number of functions required by the user* to the *total number of functions provided by the component itself.*

***Measure* = (Ratio normalized to LOS)**

Table 1. Level of satisfaction (LOS) scale

S.No.	Description	Scale (LOS)
1	Very Low Satisfaction (VLS)	1
2	Low Satisfaction (LS)	2
3	Moderate Satisfaction (MS)	3
4	High Satisfaction (HS)	4
5	Very High Satisfaction (VHS)	5

3.2 Reliability

Reliability is the capability of a component to maintain a specified level of performance when used in stated conditions in a stated period of time. It also indicates the ability of a component to return correct results in a quality manner. This characteristic is directly related to ISO 9126 but with minor modifications in the sub-characteristics notions. ***Maturity*** is renamed as ***service maturity*** in order to encompass the meaning of contributed attributes such as – ***error prone, error handling, recoverability***, and ***availability***. It is to be noted that the sub-characteristics ***recoverability*** and ***fault tolerance*** mentioned in ISO 9126 is retained but included in ***service maturity*** as measurable attributes. Also, a new sub-characteristic *Outcome Set* is introduced which asserts the correctness and quality of the results returned by the component. Following sub-characteristics contribute to Reliability of a component:

- *Service Maturity:* It expresses the level of confidence that the component is free from errors. It also indicates the time when the component is available with the services and the ability to recover from failure. Following attributes contribute to service maturity of a component

  - *Error prone*: It indicates the ability of a component to identify how much it is prone to system errors (complete system failure) and the frequency and the relative importance of those errors. It can be measured by the number of errors occurring per unit of time. Less number of the errors means less errors prone to system failure.

***Measure* = (Ratio normalized to LOS)**

- *Error Handling*: It indicates the ability of a component to provide handling mechanisms to handle if any error encountered.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Recoverability*: It indicates the ability of a component to recover when error occurs (fault tolerance). It also indicates that after recovery whether any data or system loss happens.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

Availability: It indicates the ability of a component to be available for offering services. It is measure on the basis of duration of its availability i.e. the average uptime of the component without serious errors or crashes.

***Measure* = (LOS)**

- *Outcome Set*: It indicates the ability of a component to provide correct and quality results. In addition, it indicates that the component support transaction based computation. Following attributes contribute to Outcome set of a component:

  - *Correctness*: It indicates the ability of a component to return correct results. In addition, it also indicates the quality of results in terms of computational precision and accuracy. It is measure as a ratio of *'as expected' results* to the *total number of results.*

***Measure* = (Ratio normalized to LOS)**

  - *Transactional*: It indicates the ability of a component to provide transactional processing i.e. rollback facility if transaction fails.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

It is to be noted that reliability can also be assessed by measuring the frequency and severity of failures, the accuracy of output results, the mean time between failures and the ability to recover from failure and the predictability of the program [26].

3.3 Usability

The usability characteristic represents the significant difference in meaning between ISO 9126 and component quality model. The difference lies in the fact that in component specific quality model the users are considered primarily system developers, who handle the integration of the components to their systems. Although most of the sub-characteristics remain the same in term of naming (only ***understandability*** is renamed to *help* mechanisms). Help mechanisms need to target different people. Some look for "*how to*" administer component and some are interested in understanding "***interfaces***" support. From the sub-characteristics list of ***usability***, ***attractiveness*** *and* ***compliance*** have been removed as end-users are the system developers and not the one which interact with complete system and due to lack of standards compliance is not possible. Additionally, approachability as sub-characteristic of ***usability*** is introduced in order to represent the need for effective identification and retrieval of the desired component [28]. The description of ***usability*** sub-characteristics is as follows:

- Help Mechanisms

This sub-characteristic denotes availability and effectiveness of help facility for the usage of the component. Following attributes are used to compute the level of *help mechanisms* supported by the component:

- *Help System:* It indicates availability and completeness of the help files for the system.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Manuals:* It provides the presence of *user manual, administration manual* and *installation manual*

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Tutorials and Demos:* It indicates presence of *tutorial*s and *demos* to support the usage of component.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Support Tools and services:* It denotes the presence of other supporting tools and services to enhance the usage of component such as online help using chat services or telephonic help, expert training modules or sessions etc.,

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- Learnability

The sub-characteristic indicates the time needed by a user to learn ***how to use***, ***configure*** and ***administer*** a component. Here, the users are considered to have an average experience in component related projects. The contributing attributes for ***learnabilty*** are as follows:

- *Time to use:* It indicates the time needed by an average user to learn how a component works and how it can be used in a software system.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Time to configure*: It denotes the time required for an average user to learn how to configure the component for its usage.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- *Time to administer:* It indicates the time required for an average user to administer the component.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0).**

- Operability

It indicates the level of effort required to *operate, administer and customize* the component. Following attributes contribute to *operability*:

- *Operation effort:* It indicates the effort required to operate the component. It is heavily dependent upon the suitability of the component on the assigned tasks. For example, whether a component requires some manual tasks to be performed for its operation or not.

Measure = (LOS).

- *Administration effort:* It denotes the effort required to administer a component. The functioning is similar to *operation effort* but mainly focus on administration related tasks.

Measure = (LOS).

- *Customizability effort:* It indicates the effort required to customize a component for its per-defined interfaces.

Measure = (LOS).

- Approachability

It indicates the capability of a component to be searched by its users for its usage through search mechanisms. The contributing attributes for *approachability* are as follows:

- *Directory listings*: It indicates the simplicity of finding a component. Dedicated sites on the World Wide Web or special software magazines are the source of components listings and can be used for such findings. It can be measured as the *Ratio of popular directory listings that the component is marketed to the total number of popular directory listings.*

***Measure* = (Ratio normalized to LOS)**

- *Search & Fetch:* It denotes the simplicity in searching and fetching a component. For example, whether any pre-requisites are required for approving download of component (trial version) such as registration, software or hardware, training etc., or not.

***Measure* = (LOS)**

- *Classification:* It denotes the supportability of the classification scheme by a component. For example, the component can be classified according to packaging or platform specific views etc.

***Measure* = (LOS)**

- *Marketing information:* It indicates the ability to understand the component capabilities without actually installing it. It deals with the information the vendor has provided as part of marketing strategies.

***Measure* = (LOS)**

3.4 Efficiency

It is the capability of a component to provide appropriate performance, relative to the amount of resources used under stated conditions. It corresponds to the efficiency characteristics of ISO 9126. Following sub-characteristics contributes to efficiency of a component:

- *Time Behavior:* It indicates the time difference between component's method invocation and getting its response. Following attributes contribute to time behavior of a component:

- *Throughput*: It indicates the ability of a component how fast it serves requests and provides results over a given period of time. It is measured as the ratio of *number of successful served requests per unit time*.

***Measure* = (Ratio normalized to LOS)**

- *Capacity*: In indicates the ability of a component to serve to number of users simultaneously without degrading performance level. It is measured by the number of users supported by the component. More the users better the capacity.

***Measure* = (LOS)**

- *Concurrency*: It indicates the ability of a component to provide synchronous or asynchronous invocation.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- *Resource Behavior:* It indicates the ability of a component to utilize resource to run itself. attributes contribute to resource behavior of a component:
  - *Memory utilization*: It indicates the amount of memory needed by a component to operate. It is measured by the number of kilobytes of RAM required for its operation. More kilobytes means component utilizes high memory.

***Measure* = (LOS)**

  - *Processor utilization*: It indicates the amount of processing time (CPU cycle) needed by a component to operate. It is measured by the average number of CPU cycles required for its operation. More CPU cycles means component utilizes high processing.

***Measure* = (LOS)**

  - *Disk utilization*: It indicates the amount of disk space needed by a component to operate. It includes both the disk space for installing the component and other supported materials (such as documentation, help files etc.,), as well as the disk space used temporarily during execution time. It is measured by the number of kilobytes of disk required for its operation. More kilobytes means component utilizes high disk space.

***Measure* = (LOS)**

3.5 Maintainability

Maintainability is the effort required to replace a COTS component with the corrected version and to migrate an existing, software component from a current component based software system to a new version of the system [29]. The description of *maintainability* sub-characteristics is as follows:

- Customizability

It evaluates the capability of a component to be customized according to the user needs. Following attributes contribute to customizability of a component:

  - *Parameterization*: It indicates the number of parameters offered for change by the component with the number of provided interface. Black-box parameterization is measured by *the ratio of number of parameters available for change to the total number of interfaces supported (offered) by a component.*

***Measure* = (LOS)**

- *Adaptability*: It indicates the ability of a component to adapt itself to a changing environment at runtime.

***Measure* = (LOS)**

- *Change control capability*: It indicates the ability of a component to make user aware of current version of a component.

***Measure* = (LOS)**

- *Priority*: It indicates the capability of a component to provide prioritize service, i.e. some of its function assume priority at runtime (ISO/IEC, 1991).

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- **Testability**

It examines the features of a component that can be tested by supporting test cases, tools or test suites. Following attributes contribute to testability of a component:

- *Start up self test*: It indicates the capability of a component to test itself and environment for operation.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- *Trial version*: It denotes the ability of a component to support trial version in order to facilitate user to perform test for the functionality support by a component. Associated with trial version is its timeline usage scale – limited or full version, whether a component is a freeware (f), shareware (s), or commercial (c), see table 2.

***Measure* = (LOS).**

Table 2. Component trial version type and scale

S.No.	Trial version Limited/full	Type {f, s, c}	Scale (1-5)
1	Limited	F	{VL, L, M, H, VH}
2	Limited	S	{VL, L, M, H, VH}
3	Limited	C	{VL, L, M, H, VH}
4	Full	F	{VL, L, M, H, VH}
5	Full	S	{VL, L, M, H, VH}
6	Full	C	{VL, L, M, H, VH}

- *Test suite provided*: It indicates the presence or absence of a test suite on a

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- *Test materials*: It denotes the existence of other useful test materials like demos, gray code (some level of visible code), logical and data flow diagrams and test cases.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- **Changeability**

It indicates the effort needed to modify a component at ease as per requirements. Following attributes contribute to changeability of a component:

Upgradeability: It denotes the ability of a component to be upgraded to a new version at ease. If upgradation requires manual support or intervention leading to system downtime then this will result into negative impact.

***Measure* = (LOS).**

Debugging: It denotes the ability of a component to support debugging (functional). It evaluates the efficiency of an error messages returned by the component in order to understand the erroneous functionality of component and then corrects it. *The ratio of descriptive and understandable errors to the total number of errors provided by a component can be used as a measure for debugging capability of a component.*

***Measure* = (Ratio normalized to LOS)**

Backward compatibility: It indicates whether a new component is compatible (backward) with previous versions or not. If a component is not backward compatible then re-user has to re-write the software system to achieve full functionality in order to accommodate changes.

***Measure* = (LOS)**

3.6 Portability

This characteristic indicates the ability of a component to be transferred from one environment to another with little or no modification. It corresponds to ISO9126 model in the meaning but two of its sub-characteristics have been transferred to maintainability characteristics. ***Replaceability*** is renamed as ***backward compatibility***, has been transferred to sub-characteristic ***changeability*** of ***maintainability***. Similarly, ***adaptability*** has been transferred to sub-characteristic ***customizability*** of ***maintainability***. ***installability*** characteristic is retained. In addition, d***eployability*** sub-characteristic is added as a component instance may be deployed only once but installed any number of times [28]. Following sub-characteristics contribute to portability of a component:

- *Installability*: It is the capability of a component to be installed on different platforms. Following attributes contribute to installability of a component:
  - Documentation: It indicates the ability of a component to provide supportive installable documentation or not.

  ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**
  - Complexity: It is the ability of a component which shows how complex it is to carry out the installation of a component.

  ***Measure* = (LOS).**
- *Deployability*: It is the capability of a component to be deployed on different platforms. Following attributes contribute to deployability of a component:
  - Documentation: It indicates the ability of a component to provide supportive delpoyable documentation or not.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- Complexity: It is the ability of a component which shows how complex it is to carry out the deployment of a component.

***Measure* = (LOS).**

3.7 Reusability

It is a critical characteristic in the development of component based software system [30], as component can be reused to create more complex applications. Since Reusability is a driving force for component markets we have included this as a separate characteristic. Reusability improves productivity, maintainability and overall quality of a system. Following sub-characteristics contribute to Reusability of a component:

- *Interoperability*: It is the ability of a component to be reused. The evaluation of interoperability is of prime importance since reusability aspect is the cornerstone of CBSS success. Following attributes contribute to Interoperability of a component:
  - *Platform Independence*: It indicates the ability of a component to be used in different component models such as CORBA, COM/DCOM and other similar models. It is measured as a ratio of *number of platforms/models supported* to the *number of most used platforms/models.*

    ***Measure* = (Ratio normalized to LOS)**
  - *Operating System Independence*: It indicates the ability of a component to be used in different operating systems such as Windows, LINUX, UNIX, AIX etc., and other similar operating systems. It is measured as a ratio of *number of operating systems supported* to the *number of most used* operating systems.

    ***Measure* = (Ratio normalized to LOS)**
  - *Hardware Compatibility*: It indicates the ability of a component to be used in different types of hardware such as personal computers, laptops, PDAs and other similar hardware. It is measured as a ratio of *number of* hardware *supported* to the *number of most used* hardware.

    ***Measure* = (Ratio normalized to LOS)**
  - *Data Open-format Compatibility*: It is the ability of a component to output data that is compatible with the well-known formats.

    ***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**
- *Generality*: It is the capability of a component to be generic so that it can be reused in number of applications. Following sub-characteristics contribute to generality of a component
  - *Domain abstraction:* It is the ability of a component to be reused across several domains related to specific functionality that the component offers.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

- *Reuse maturity:* It is the ability of a component to show/support the reusability record and documentation for future reference.

***Measure* = (Boolean scale: If value is 1 then LOS otherwise value is set to 0)**

3.8 Traceability

This characteristic expresses the extent of a component's built in capacity of tracking the status of component attributes and component behavior. According to [31] traceability refers to the ability to show, at any time, where an item is, its status, and where it has been. It can be noticed that while reconfiguring a component for changed/improved functionality, maintainers must perform a full cycle of product evaluation, integration and testing. Replacing an older version of a component with newer version can also create problems such as security violation, resource usage and performance usage degradation etc. This characteristic is not present in ISO 9126 but we felt that *traceability* of the component is important so as to validate any violation during modification/replacement of a component. So we treated *traceability* as a separate characteristic. Following sub-characteristics contribute to traceability of a component:

- *Behavior traceability* - It is the ability of a component (black box) to track its external behaviors. The major purpose is to track component public visible data or object states, visible events, external accessible functions and the interactions with other components. Following attributes contribute to behavior traceability

- *Performance trace* - It is the ability of a component to records the performance data and benchmarks for each function of a component in a given platform and environment.

***Measure* = (LOS)**

- *State trace* - It tracks the object states or data states in a component.

***Measure* = (LOS)**

- *Controllability Trace* - It is the ability of a component that refers to the extent of the control capability to facilitate the customization of its tracking functions. Following attributes contribute to controllability trace

  - *Customization trace*: It is the ability of a component to control and set up various tracking functions such as turn-on and turn-off of any tracking functions and selections of trace formats and trace repositories.

***Measure* = (LOS)**

  - *Error trace* - It records the error messages generated by a component. The error trace supports all error messages, exceptions, and related processing information generated by a component.

***Measure* = (LOS)**

The proposed component quality model does not discuss *business* characteristics (specific to market) as it does not certify to be quality characteristic specific to component (in quality context).

4 Conclusion

The proposed Quality model (***SCQM***) is a first step towards research effort in the direction of building a software component quality assurance framework. The model was designed in such a way as to overcome the drawbacks of existing models. Research is undergoing to develop capability maturity model for software components that can be mapped to the proposed component model. Our future work will try to incorporate emerging needs of (re)users as the software technology advances such as mobile computing and pervasive computing. We are also focusing on developing dedicated web application providing XML schema, evaluation tool and structured feedback mechanisms in order to achieve world-wide evaluation. This may results for evaluators to discuss and share their experiences regarding usage and difficulty level in using ***SCQM***.

References

[1] Tran, V., Liu, B.D., Hummel, B.: Component-based Systems Development: Challenges and Lessons Learned, Software Technology and Engineering Practice. In: Proceedings of the Eighth IEEE International Workshop on ICASE, pp. 452–462 (1997)

[2] Raje, R., Bryant, B., Auguston, M., Olson, A., Burt, C.: A Unified Approach for the Integration of Distributed Heterogeneous Software Components. In: Proceedings of the 2001 Monterey Workshop Engineering Automation for Software Intensive System Integration, Monterey, California, pp. 109–119 (2001)

[3] Kallio, P., Niemela, E.: Documented quality of COTS and OCM components. In: Proceedings Fourth ICSE Workshop on Component-Based Software Engineering, Toronto, Canada, pp. 111–114 (2001)

[4] Preiss, O., Wegmann, A., Wong, J.: On Quality Attribute Based Software Engineering. In: Proceedings of the 27th Euromicro Conference, pp. 114–120 (2001)

[5] Szyperski, C.: Component Object-Oriented Programming. Addison-Wesley, Reading (1998)

[6] Andreou, S.A., Tziakouris, M.: A Quality Famework for Developing and Evaluating Original Software Components. Information and Technology 49, 122–141

[7] Upadhyay, N., Deshpande, B.M., Agrawal, V.P.: MACBSS: Modeling and Analysis of Component Based Software System. In: Proceedings of IEEE World Congress on Computer Science and Information Engineering, Los Angeles, USA, pp. 595–603 (2009)

[8] Rakic, M., Medvidovic, N.: Increasing the Confidence in Off-The-Shelf Components: a Software Connector-Based Approach. In: Proceedings of the Symposium on Software Reusability, pp. 11–18 (2001)

[9] Mann, S., Borusan, H., Grobe-Rohde, M., Mackenthun, R., Sunbul, A., Weber, H.: Towards a Component Concept for Continuous Software Engineering, Fraunhofer ISST, Technical Report (2000)

[10] Gao, W.: Testing and Quality Assurance of Software Components. Arctech Pubishing House, Boston (2003)

[11] Parminder, K., Hardeep, S.: Certification Process of Software Components. ACM SIGSOFT Software Engineering Notes 33 (2008)
[12] Behkamal, B., Kahani, M., Akbari, K.M.: Customizing ISO 9126 Quality Model for Evaluation of B2B Applications. Information and Software Technology 51, 599–609
[13] Fizpatrick, R.: Software Quality definitions and strategic issues, Technical Paper, Staffordshire University (1996)
[14] Bohem, B.W., Brown, J.R., Kaspar, H., Lipow, M., McLeod, G., Meritt, M.J.: Characteristics of Software Quality. North Holland Publishing, Amsterdam (1978)
[15] Dromey, R.G.: A Model for Software Product Quality. IEEE Transactions on Software Engineering 21, 146–162 (1995)
[16] Khosravi, K., Guehneuc, Y.: A Quality Model for Design Patterns, Technical report 1249, University of Montreal (2004)
[17] Jacobson, I., Booch, G., Rumbagch, J.: The Unified Software Development Process. Addison-Wesley, Reading (2000)
[18] Krutchen, P.: The Rational Unified Process: An Introduction. Addison-Wesley, Reading (2000)
[19] Grady, R.: Practical Software Metrics for Project Management and Process Improvement. Prentice-Hall, Englewood Cliffs (1992)
[20] Stefani, A., Xenos, M., Stavrinoudis, D.: Modeling E-Commerce Systems' Quality with Belief Networks. In: Proceedings of International Symposium on Virtual Environments, Human- Computer Interfaces, and Measurement systems, Switzeland (2003)
[21] Stefani, A., Stavrinoudis, D., Xenos, M.: Experimental Based Tool Calibration used for Assessing the Quality of E-commerce Systems. In: Proceedings of the First IEEE International Conference on E-Business and Telecommunication Networks, Portugal, pp. 26–32 (2004)
[22] ISO International Organization for Standardization.: ISO 9126-1:2001 Software engineering-Product quality, Part 1: Quality Model (2001)
[23] Bertoa, M., Vallecillo, A.: Quality Attributes for COTS Components. In: Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE), Spain (2002)
[24] Alvaro, A., Almeida, A.S.: Towards A Software Component Quality Model. In: Proceedings of the 5th International Conference on Quality Software (2005)
[25] Alvaro, A., Almeida, A.S., Meira, S.R.: Quality attributes for a Component Quality Model. In: Proceedings of the 10th International Workshop on Component Oriented Programming (WCOP) in conjunction with the 19th European Conference on Object Oriented Programming (ECOOP), Glasgow, Scotland (2005)
[26] Raweshdah, A., Matalkah, B.: A New Software Quality Model for Evaluating COTS Components. J. of Comp. Sc. 2, 373–381 (2006)
[27] Szysperski, C., Dominic, G., Stephen, M.: Component Oriented Programming-Beyond Object Oriented Software, 2nd edn. Addison Wesley and ACM Press, New York (2002)
[28] Hansen, W.J.: An Original Process and Terminology for Evaluating COTS Software, http://www.sei.cmu.edu/staff/wjh/Qesta.html (accessed March 25, 2010)
[29] Gao, J., Gupta, K., Gupta, S., Shim, S.: On Building Testable Software Components. In: Palazzi, B., Gravel, A. (eds.) ICCBSS 2002. LNCS, vol. 2255, pp. 108–121. Springer, Heidelberg (2002)
[30] Jon, H.: Component Primer. Communication of ACM 43, 27–30 (2000)
[31] Schmauch, C.H.: ISO 9000 for Software Development: Revised Edition, American Society for Quality (1995)