Reaching an Agreement on COTS Quality through the Use of Quality Models

Juan P. Carvallo, Xavier Franch, Gemma Grau, Carme Quer
Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
{carvallo, franch, ggrau, cquer}@lsi.upc.es

Abstract

It is difficult to reach to an agreement on how to measure software quality. It has been argued that one of the reasons for this is that quality is a complex concept, for which a universal definition does not exist. Quality means different things to different people, thus it is highly subjective and context-dependant. In [12] Kitchenham stated that quality is "hard to define, impossible to measure, easy to recognize". Gilles [9] stated quality is "generally transparent when present, but easily recognized in its absence". Both of these statements imply that quality is somehow perceptible. Thus, the problem is not quality being subjective, but how to correlate the different views on quality into a measurable quality framework, which can be commonly agreeable at least in some context (e.g., a domain of knowledge, an organization, a project, etc.).

Software quality models help in this purpose: they can be used as a base to define a commonly agreeable quality framework, which consolidates the different views on quality; they can be tailored to specific contexts; they provide a measurable base to the evaluation of software quality. However, several problems remain which let quality models to be seldom used or misused in the practice. In this paper we review these problems and propose a set of characteristics which provide the basis for the construction of well founded and useful quality models. We focus our analysis in the context of COTS components.

1. Introduction

Software quality has been one of the main goals of software engineering in the last decades. However, despite of the efforts to develop new and more powerful software process improvement techniques, such as CMM (Capability Maturity Model) [6], Bootstrap [4] or SPICE [8], and the development of better metrics and product validation techniques, it remains an elusive target. More recently this problem has been aggravated by some factors: the tendency of both private companies and public administration offices for acquiring software rather than construct it; the increasing availability of new software products in the market; and the continuous creation of new communication and marketing channels (e.g. web markets, search engines, etc.), which bridge the gap between providers and consumers of those products.

These factors have lead to an alternative approach to the traditional one of building systems from the scratch which is becoming increasingly important. This alternative consists on building systems by assembling and integrating existing software components in the form of Commercial-Off-The-Shelf software components (COTS for short) [20,19], into COTS-based software systems (COTS systems, for short). COTS are software components that are available in the market to be purchased, interfaced with and integrated into other software systems. More precisely: *"A COTS component is a [software] product that is: (1) sold, leased, or licensed to the general public; (2) offered by a vendor trying to profit from it; (3) supported and evolved by the vendor, which retains the intellectual property rights; (4) available in multiple, identical copies; (5) used without internal modification by a consumer".* [19]

Intensive use of COTS during software development requires adapting some software engineering practices to this framework. This includes traditional activities such as requirements elicitation, architectural design and testing, but also some specific of the field, among which assessment of product quality plays a prominent role.

According to Voas [25] the difficulties to assess software quality have lead to three distinct and competing approaches: the level of preparation of the people involved in the software development process; certification of the software development process itself; and the inspection of the final product. Those three competing aspects have been considered in what he calls the software quality certification triangle.

Although yet to be evaluated, the positive impact that a well qualified personnel has in the software development process is assumed as a fact.

Several software process improvement techniques have emerged in the last decades. One of the reasons for their advent is that even when adopting a well defined testing program and formal validation techniques, final products are still not 100% reliable. This is true even in those cases in which software survivability is a primary concern [22]. Thus, the notion of "directly assessing software quality" is dismissed as implausible, and replaced by the idea of improving software quality by introducing a set of well defined steps, in the software development processes. According to Voas, these models rely on one premise, "*good processes deliver good software*". The problem is that good processes do not guarantee good software. As stated in [13] "*there is little evidence that conformance to process standards guarantees good products*" and [24] "*if performed properly, good processes merely increase the likelihood of producing quality products*".

Because of the problems identified in process certification and the switch towards the use of COTS in systems development, final product assessment has regained importance. Two approaches to product quality assessment exist: white-box and black-box. White-box techniques require the source code of the products to be made available. This is not the usual case with COTS. COTS are black boxes (or at best very dark ones), customers seldom own their source code, therefore product measurements can be applied and/or obtained only for those externally observable quality characteristics. Traditional black-box techniques include reliability testing and the use of software quality models to evaluate externally visible quality features.

A structured quality model for a given software domain provides a taxonomy of software quality entities and also metrics for computing their value. Once available, requirements over the domain, as well as component features, may be stated with respect to the quality model. Product quality can be evaluated and their characteristics can be compared with requirements based on a common description.

The objective of this paper is review existing software quality models, to identify possible problems and propose a set of guidelines to improve their construction, applicability and reuse. Section 2 reviews the existing approaches and list the problems identified. Section 3 outlines the objectives to be pursued in model construction. Section 4 describes a set of guidelines for quality model construction that we have identified. Section 5 provides the conclusions and future lines of research.

2. Reviewing Existing Quality Models

Many software quality models have been proposed in the last decades [18, 2, 16, 1, 27], all of them share a hierarchical tree-like structure composed of a set of high-level quality attributes to which identifiable and measurable low-level quality attributes are linked. Although the elements of the models have been named unequally along the different proposals, correspondences among them can easily be identified. The number of layers of the hierarchy, as well as the number of fixed elements of the models, largely differs among the different proposals. Some of them e.g. McCall's and the FURPS model, fix the hierarchy in such way that only the metrics can be redefined by end-users, whilst others e.g. IEEE 1061-1998, are very open and let users refine the entire hierarchy and the number of elements included. Unfortunately, as pointed in [7], while those proposals paid attention to high-level attributes, very little has been devoted to the study of lower-level quality attributes and their influence on high-level ones. In our opinion, even less attention has been paid to the methodological aspects required to support their construction.

However, some approaches to the construction of quality models indeed exist: the fixed model approach in which models are constructed by using a subset of attributes of a pre-existing attribute catalogue (i.e. [18], [2]); the custom model approach in which models have to be build from the scratch for each project (i.e. [1], [27]); and the mixed model approach in which a pre-existing model is used as a base for model construction but their attributes are tailored by consensus of the participant stakeholders (i.e. [16, 14]). Although most of the proposals do not provide complete and well-founded guidance of how the models have to be constructed/tailored by final users, they unveil some important principles to be applied in this process:

- The importance of justifying the linkage of lower and upper levels of the hierarchical structure by some underlying logic.
- The importance of building communicable models to be used both by technical and non-technical users (usually by means of a dual notation).
- The existence of different kinds of relationships among quality elements which need to be identified.
- The need to identify internal and external elements of the models to increase the applicability of the models.
- The need for overlapping catalogues because quality attributes affect different factors in different ways.
- The importance of defining metrics not only to the lowest hierarchical levels, but also to the upper levels, in such way that elements in the upper layers of the hierarchy can be measured by the aggregation of metrics of elements in lower layers, or by directly computable metrics.

A great deal of applications for software quality models have been suggested, and in some cases explored in the last decades [7, 15, 21, 10, 17]. This endorses the notion that quality models are useful artifacts which can support several stages of software lifecycle. Despite this fact, several aspects attaining their construction and practical use seem to require further analysis and exploration by the software engineering research community. Some of the problems identified in our research are:

- *Most of them are ad-hoc proposals.* They have been build to support particular applications inside particular organisations or projects, thus they are not the result of the consensus of a widespread community, which makes their further adoption difficult to achieve.
- *Their semantic is not clearly stated.* As a consequence particular definitions are either unclear or ambiguous. Even the concept of quality model is defined in different ways by different organizations and authors.
- *There is no method to support the rationale of their construction.* Model construction is driven just by intuition, thus the existence of the elements included in the model as well as their connections are not well justified.
- *Their level of detail compromises their applicability.* Existing models are either too general, including just some high-level attributes, or too domain and/or application-specific. In the first case they need to be completed by the end-user in order to make them useful, whilst in the second case, their level of detail makes them impractical for those application or domains other than the ones they have been build for. In addition some proposals only include a taxonomy of thousands of domain independent quality attributes, from which users are supposed to select the appropriated sets, without further support.
- *They consider COTS as independent and isolated.* In the practice however components need to communicate with others, either to enable their installation or to extend or enhance their functionalities. This is particularly true in the case of COTS systems where the aim is not only to evaluate particular components but the resulting system as a whole. Some models deal with this problem by providing interoperability characteristics and attributes, but this increase substantially the size of the model, damaging the understandability and the reuse of the deliverable.
- *They lack the use formal representations.* Quality models are technical assets in nature; they are created, manipulated and consulted by software engineers. However existing proposal disregard the use of formal notations to represent the important concepts therein.
- *They do not uphold reusability.* Quality model construction has been recognised in most of the existing approaches as a difficult task, however, mechanisms to improve reusability of the knowledge gained in each experience are not provided.
- *They only include technical factors.* Most of the existing proposals only recognise the importance of technical quality factors whilst in some cases e.g. COTS procurement, non technical factors can be just as important.

3. Objectives of Quality Model Construction

According to Dromey, depending on the orientation of the construction, the resulting quality models can serve for different objectives: to describe a problem; as the basis for design; as an instrument of contract and common understanding among client, users and developers. Therefore the first activity to perform when building software quality models is to state well defined objectives for their construction. In our case the main objective can be stated as:

"Providing support to the efficient construction of reliable and understandable coarse-grained COTS quality models, to be used in the evaluation of COTS systems"

The rest of the section further analyzes this general objective with some descriptions intended to refine each of its aspects.

What do we mean by efficient construction?

Building software quality models is an activity endangered by various threats, among others: lack of experience in quality model construction; ignorance about the quality scope; lack of a standard terminology, and the difficulty to identify, decompose and categorize attributes. Additionally to those problems, the difficulties to identify the appropriate level of detail to which the model has to be build, and when model construction is finished, make this process difficult to rely only in common sense. Thus by efficient construction of quality models we mean:

- To construct quality modes following a rigorous and systematic method, with well-defined steps and objectives, leading to the identification of the appropriated elements of the model. None of the important element should be missed nor should irrelevant ones be included.
- To construct quality models by reusing the knowledge gained in each experience. Thus the method should rely in some artifacts to support reusability of the deliverables.
- To identify useful levels of construction of the quality models. In this way models can be reused by completing and/or tailoring just those areas useful to the specific project and the application for which they are intended.

What do we mean by reliable and understandable COTS quality models?

By *reliable* we mean quality models which are founded on a deep understanding of the domain and its environment. All the relevant actors, technologies, regulations and standards of this domain should be identified as well as their relationships and dependencies among them. The result of this analysis should be formally represented, to create a reliable base for the identification of the different elements composing the quality model.

By Understandable we mean quality models which are based on a clearly stated, complete and unambiguous quality framework, which includes all the required concepts and their relationships. We also mean quality models which, by providing a formal representation of their elements, can be clearly understandable by their final users (COTS vendors, system engineers, clients, etc.).

What do we mean by coarse-grained COTS?
COTS available in the market differ widely among each other. They may range from simple libraries which provide a limited and clear set of functions, to components whose functionality is so broad and complex that their understanding requires extensive exploration. We call this kind of COTS coarse-grained COTS. They are the focus of our attention, because we believe that the expected benefits of building quality models for them are greater than the ones expected for other COTS, in which functionality is so thin that becomes easily identifiable and comparable.

What do we mean by evaluation of COTS systems?
COTS systems are constructed by the integration of several COTS. As a consequence its quality is somehow influenced by the quality of its individual components. However, overall system quality can only be evaluated once they are combined and work together as a whole. As a consequence a COTS system quality model has to be constructed if willing to assess system quality. Such a model should resume and/or adapt the relevant quality characteristics of its components, and also incorporate others which are only applicable to the combined system.

4. Guidelines to Improve Quality Model Construction.

Based on the problems identified in section 2 and the objectives of section 3, in the next sections we propose some guidelines aimed to the construction of well-founded, consensual and useful quality models.

4.1 Use a well–defined, not ad-hoc quality framework

Models shall rely on well known quality standards
Most of the existing quality models are ad-hoc proposals build to be used in particular contexts and applications, thus, their reusability is compromised. Not only quality elements are context-dependent, but also all the quality concepts included. Thus, the use of widespread standards is highly recommended. Elements and concepts of these models are more likely to be reused because they are the result of the consensus of large communities. Some standard quality models exist e.g. ISO/IEC 9126-1 and the IEEE 1061-1998. The IEEE 1061-1998 is a very open standard, it just fixes a general structure of the model but, all the elements have to be defined by the users. On the contrary the ISO standard fixes the top hierarchical levels of the models, which again, favor reusability.

Clarify and complement the standards
Due to its generic nature, some of the concepts presented in the quality model standards need to be refined before being used in real software procurement projects. For example, in the ISO/IEC 9126-1 it is not clear if subcharacteristics can be refined into other subcharacteristics, or if subcharacteristics shall include metrics for their evaluation. A good way to address this problem is to provide conceptual models, which give an unambiguous and complete view of the quality framework and allows a precise understanding of all the enclosed concepts and relationships between them [3].

4.2 Use formal notations to model and understand the system

Develop models to analyze and represent the environment of the COTS system
Systems do not operate in isolation; they interact with external actors (human, software, organizations, etc.) which act together as its environment. External actors constrain the way in which the system behaves and determines the services it has to offer. Thus, in order to clearly understand the boundaries of the system, these actors and their most important dependencies have to be identified. Notations such as the one proposed Yu's i* Strategic Dependency (SD) models [26], serve well for this purpose. One of the advantages of system environment models is that they can be used to identify the services that systems have to offer. This is the base for the decomposition of a system into individual elements (system actors), which correspond to COTS domains, each offering well-defined services.

Develop conceptual models to precisely define and represent the individual COTS domains
Construction of software quality models has to be supported by a deep exploration of the domain of interest. The results of this study cannot entirely rely on common sense and informal documents, even if they are highly structured. Regardless of the level of detail and precision used to describe the domain in natural language, we felt compelled to represent it using some formal notation, e.g. UML class diagrams [23]. In addition to the final product, the diagram construction process itself helps to better understand the domain.

4.3 Define a method to build COTS quality models

Use a mixed approach to the construction of COTS quality models

It was mentioned that there exist three possible approaches for building software quality models the fixed model approach), the custom model approach and the mixed model approach. Software products vary widely from one domain to another which makes the fixed approach hard to be tailored in some domains. On the other hand many software domains share common characteristics. This implies that at least some parts of their quality models can be reused in quality models of related domains. This notion somehow contradicts the custom model approach because models for different domains do not necessarily have to be build (at least not entirely) from the scratch. Therefore the idea of a mixed approach in which models are build departing from a pre-existing catalogue, which has to be completed and/or tailored for the particular domain being treated, seems to be the most appropriated one.

Use well-defined steps to identify COTS quality model elements

Standard quality models, such as the ISO/IEC 9126-1 standard, fix the top level characteristics and their further refinement into subcharacteristics but do not elaborate the quality model below this level, making thus the model flexible and tailor able to domains of different nature. However this standard does not provide any method to identify and justify the elements in which the original categories and subcategories have to be refined. Because of the number of elements and the relationships that can be found, it is utterly important to relay its identification into as set of well defined steps. One approach that we have proposed can be found in [11]

Use an iterative and intertwined approach to construct COTS quality models

Due to the nature of quality models construction this is a basic requirement. On the one hand, model elements are identified as knowledge of the domain increases, thus the model may be extended and refined all the way through the process. On the other hand, the level of detail to which the quality model has to be constructed, depends on the context of use and the final application for which the quality model is intended. An iterative approach, in which the construction steps can be intertwined at any point, helps to refine the quality model to an appropriated level based on particular requirements.

Build new quality models by inheriting elements of models of related domains

Many COTS share or embrace the characteristics of other COTS in related domains. A repository of COTS domains can be progressively constructed as domains are explored and more quality models are constructed. Models in this repository can be used as departing point in the construction of new quality models. This repository shall not only include quality models bound to each domain, but also the statement of the dependencies among them. Classifying COTS products as belonging to one or more of these domains is the first step in defining to which quality models are bound to. New quality models can be build by inheriting all or some of the characteristics of existing models in stead of starting from scratch.

Consider non-technical factors an integral part of the models

Most of the existing quality models have been conceived as the basis to the evaluation of the technical factors (functional and non-functional) of a COTS domain. Non-technical factors (e.g. managerial, economical or political) have not been considered by the standard. Despite this fact, in many situations these factors are also significant, often even more important than technical ones; therefore, not considering them could compromise the success of the undertaken activity.

These factors can be structured in the same way as the technical ones. That is, defining a set of high level characteristics and subcharacteristics, which can be refined in the usual way, spanning hierarchies of non-technical subcharacteristics and attributes which, can be measured with some metrics. The result is an enlarged COTS quality model including both types of factors, which makes it more applicable.

4.4 Construct COTS systems quality models by composition

Individual quality models give an exhaustive but individual, isolated and therefore incomplete view of parts of the COTS system. They are useful for several applications such as procurement of individual components, support to architectural decisions and COTS system requirements elicitation, among others. However, we aim at the construction of system level quality models capable to support the evaluation of the final COTS system as a whole. One of the ways to construct such models is by providing mechanisms to combine the elements of the quality models of its individual components. These mechanisms include the identification of combination patterns of the elements of the individual models. The result is quality model, which includes elements of the quality models of its components, and other relevant to the evaluation of the system.

4.5 Identify artifacts to improve knowledge reuse

In some cases quality model construction can be cumbersome, but it really pays off if the knowledge gained can be reused in future experiences. This can be obtained if the resulting quality models (or parts of them) as well as other deliverables of the process are stored in proper repositories. We suggest some artifacts that we have found useful in our experience:

- *Environmental patterns repository.* Situations that appear over and over in different system environment models can be identified, refactored and defined as patterns. For each pattern, we state which requirements lead to it and which ISO/IEC 9126-1 subcharacteristics are addressed.
- *Software components domains taxonomy.* Domains of the components available in the market can be arranged as a taxonomy [5]. This facilitates the identification of the kind of components required in the system. The taxonomy includes quality models bound to its nodes' domains as well as relationships among them. The quality models include not only the usual quality features and their metrics, but also parameterized requirements that can be used during requirements analysis and elicitation.
- *Platform strategies repository.* We maintain a catalogue of the most usual platform strategies used when deploying a system (such as clustering, load balancing, and so on). For each of them, the ISO/IEC subcharacteristics affected by their use are identified, as well as the platform parameters likely to appear.

5. Conclusions

Software quality been subjective and difficult to define is not in discussion. It means different things to different people, and is highly context dependant. But we argue that what is needed in order to reach to a consensus, is a framework in which the different views on quality can be consolidated. We also argue that such framework can be described with the use of software quality models. Through this paper we have explored the problems of existing quality models, and presented a set of guidelines to improve their construction. Departing from well-defined objectives, and guided by a well defined-rational, we expect to construct well-founded quality models which can be used as the consensual base to measure software quality.

References

[1] V.R. Basili, *"Software modeling and measurement. The Goal-Question-Metric paradigm"*, Computer Science Technical ReportSeries NR: UMIACS-TR-92-96, 1992

[2] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, and M.J. Merrit. *"Characteristics of Software Quality"*, volume 1 of TRW Series of Software Technology. North-Holland, 1978.

[3] P. Botella, X. Burgués, J.P. Carvallo, X. Franch, G. Grau, J. Marco, C. Quer. *"ISO/IEC 9126 in practice: what do we need to know?"*. First Software Measurement European Forum, Roma, January 2004.

[4] BOOTSTRAP team: BOOTSTRAP: Europe's assessment method. IEEE Software, May 1993.

[5] J.P.Carvallo, X. Franch, C. Quer, M. Torchiano, *"Characterization of a Taxonomy for Business Applications and the Relationships between them"*. Proceedings of the 3rd ICCBSS, Redondo Beach (USA) , February 2004

[6] Capability Maturity Model (Software Engineering Institute): http://www.sei.cmu.edu/cmm/cmm.html

[7] R.G. Dromey. *"Cornering the Chimera"*. IEEE Software, vol. 20, January 1996.

[8] El Emam, K.; Drouin, J. N.; Melo, W. (Eds): *"Spice: The Theory and Practice of Soft-ware Process Improvement and Capability Determination"*. IEEE Computer Society, 1998.

[9] A. C. Gillies, *"Software Quality, Theory and Management"*, International Thomson Computer Press, 1997

[10] D. G. Firesmith. *"Using Quality Models to Engineer Quality Requirements."* Journal of Object Technology 2(5) 2003.

[11] X. Franch, J.P. Carvallo. "Using Quality Models in Software Component Selection". IEEE Software, 20(1), 2003.

[12] B. Kitchenham. *"Software Metrics"*. In Software Reliability Handbook, Elsevier, 1989.

[13] B. Kitchenham, S.L. Pfleeger, *"Software Quality: The Elusive Target"*, IEEE Software, January 1996.

[14] G. Horgan, S. Khaddaj, P. Forte, *"An essential views model for software quality assurance"*, from Project Control for Software Quality, Editors, R. Kusters, A. Cowderoy, F. Heemstra, E. van Veenendaal. Shaker Publishing, 1999.

[15] L.E. Hyatt, L.H.Rosenberg, *"A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality"*, ESA Software Assurance Symposium, 8th Annual Software Technology Conference, 1996,

[16] ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model, June 2001.

[17] F. Losavio, L. Chirinos, N. Lévy, A. Ramdane-Cherif. "*Quality Characteristics for Software Architecture"*. Journal of Object Technology 2(2), 2003.

[18] J.A. McCall, P.K. Richards and G.F. Walters. *"Factors in software quality"*, Vols I-III, Rome Air Development Centre, Italy , 1977.

[19] B.C. Meyers, P. Oberndorf. *"Managing Software Acquisition"*. Addison-Wesley, 2001.

[20] Proceedings of the 1st ICCBSS, LNCS 2255, Orlando (FL, USA), February 2002.

[21] L. Olsina. *"Web-site Quality Evaluation Method: a Case Study on Museums"*. ICSE 99 - 2nd Workshop on Software Engineering over the Internet.

[22] P. Regan, S. Hamilton. *"NASA's Mission Reliable"*. IEEE Computer 37(1): 59-68 (2004)

[23] Unified Modelling Language (UML) 1.4 specification. OMG document formal/ (formal/2001-09-67). September, 2001.

[24] J. Voas, *"Can Clean Pipes Produce Dirty Water?"* IEEE Software, July 1997, pp. 93-95

[25] J. Voas, *"The Software Quality Certification Triangle,"* Crosstalk, Nov. 1998, pp. 12-14.

[26] E. Yu. *"Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering"*. Proceedings of 3rd ISRE, 1997.

[27] IEEE Standard 1061-1998. Standard for a software quality metrics methodology. IEEE, 1998.