Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

Software Quality Models: A Comprehensive Review and Analysis

M. Sadeghzadeh Hemayati[1], H. Rashidi[2, *]

[1]Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.

[2]Department of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran.

Article Info

Article History:
Received 17 March 2017
Revised 31 July 2017
Accepted 01 November 2017

Keywords:
Software
Quality
Software Engineering
Models

[*]Corresponding Author's Email Address: hrashi@atu.ac.ir

Extended Abstract

Background and Objectives: One of the major challenges in software engineering is how to respond to the desolate state of high-quality software development in a timely and cost-effective manner. Many studies have been conducted in an attempt to formalize the quality of software. However, according to the recent researches, the lack of comprehensive quality model is rooted in neglecting all quality aspects.

Methods: In this study, we review nineteen quality models and classify them from three different perspectives, including structural, behavioral, and basic and derived aspects. The main aim is to specify and extract the more comprehensive set of quality factors to evaluate software quality.

Results: This paper compares the different quality models and analyzes the factors to draw the necessary aspects in comprehensive quality models. Since the software quality involves several engineering tasks and several players who deal with quality concepts during software life cycle according to their various roles, in various phases and different artifacts, comprehensive quality models must consider many factors.

Conclusion: These factors are in different aspects such as the measurement time in different development phases, product as well as process-related quality factors, a set of quality metrics measureable on the different type of artifacts such as document, model and source code, and finally a specific mechanism to apply dynamic weights to quality factors to determine their impacts on final quality of a product based on its application domain.

Introduction

There are two kinds of requirements in software engineering projects, namely, functional and non-functional requirements. Functional requirements are concerned with the technical functionality of a software, whereas non-functional requirements define criteria that software can be used to judge the functions of the software in particular conditions, rather than specific behaviors. Non-functional requirements generally are specified by quality models that annotate (qualify) functional requirements. Qualification might be how fast the function must be performed, how resilient it must be to erroneous input, how easy the function is to learn, etc. With the increasing use of software system in today's world, the quality of software and its evaluation has garnered widespread attention. Software depends upon its quality [1], effectiveness, and completeness [2]. Quality is the degree to which a system, component, or process meets certain requirements. Although the subjectivity of quality concept is an obstacle to providing a precise definition for quality of software, there are two acceptable explanations for it [3]:

- Conforming with specification: the quality of a service or product determines compliance of final product or service specifications to the original specifications.
- Meeting consumer requirements: the quality is the capability of a product/service to satisfy implicit and explicit user needs.

Modeling can help with better understanding and controlling of complex concepts. Therefore, the software

quality model is a tool for software quality description and management [3].

Since 1978, multiple quality models have been developed for software quality assessment and measurement.

Figure 1 shows possible building blocks of quality models. Quality models are created based on some building blocks including quality objectives, factors, criteria, sub-criteria, and metrics. A quality model can include all or a part of these building blocks. Quality objectives should be identified according to the non-functional requirements of a software product. After specifying quality objectives, some quality factors (attributes) should be identified based on management point of view. Since the factors are very general, they should be decomposed to a set of criteria and sub-criteria, which are based on software point of view, in order to decrease the abstraction of the related factor. On the other hand, criteria cannot be measured for a software product directly [5]; therefore, each one should be decomposed to some measurable quantitative metrics [6]. Criteria and sub-criteria are assessed by measuring metrics. The quality factor satisfaction is specified by composing and aggregating its related criteria and consequently, the quality of the final product or the achievement of quality objectives is estimated by aggregating their associated factors.

According to the recent researches, the lack of comprehensive quality model is rooted in neglecting all quality aspects [3]. Even standard quality models are not comprehensive enough to be used for different engineering tasks [7]. McCall does not consider final product functionalities. The portability does not affect the quality of the final product in the FURPS. In some models, such as Boehm and Dromey, there are no measurement approaches for quality factors. Bertoa and Vallecillo [8] do not consider reusability. Alvaro et al. [31] did not provide any solution to cope with the subjectivity of quality factors and their dependence upon expert's knowledge and experience [7] and different entities with different levels of knowledge [9]. Software quality is a complex concept that is affected by different aspects of the software development process. Software quality involves some different engineering tasks such as quality of the product, process, design, code, and test and application domain [3]. Moreover, different players are involved in quality concept based on their roles (e.g., analyzer, designer, architecture) at the different times (e.g., requirements, analysis, design phases) with various artifacts such as models, documents, and code. The representation, recognition, assessment, and estimation of software quality require a comprehensive model that covers all involved aspects. In this paper, we analyze and compare multiple quality models, both basic and derived models, to identify the important aspects, which should be included in a comprehensive model.

Quality Models Review

Many types of researches have been done on the software quality area and several models have been developed. Since it is impractical to assess all quality models, we use three common software quality model classifications and evaluate the strengths and weakness for each category by focusing on their potential to be a comprehensive model. At the end of this section, we show the state of each model from the classification's point of view. As mentioned above, each model consists of a set of building blocks including quality objective, factors, criteria, sub-criteria, and metrics. Assessment of how to organize building blocks and their interactions lead to structural classification[32]. From this point of view, the models are divided into three categories: hierarchical, meta-model-based and statistical quality models. The models of the software development process differ in behavior and purpose. Some quality models are used to define quality requirements; some are used to assess product or process quality and some are used to estimate final product quality. So, from a behavioral perspective, the models can be divided into three categories: definition, assessment and estimation models [32]. According to Miguel et al. [25], some quality models are independent of each other and contain several building blocks. These models are known as basic models. Examples are McCall, Boehm and ISO. Other models are derived from the basic models. These models extend the basic model to cover a specific domain. In other words, the derived models cope with the weaknesses of a basic model in the particular area by extending and modifying it. For example, the quality model of Bertoa and Vallecillo [8] was developed on the basis of ISO9126 for effective evaluation of COTS, Alvaro et al. (2005) proposed a framework based on ISO9126 for certification of software components in CBD, and Alrawashdeh et al. [12] adapted the ISO9126 for ERP quality assessment.

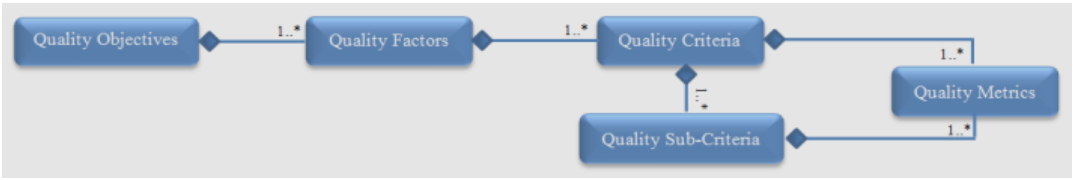

Fig. 1: A building block of quality models.

A. Structural Classification

One of the traditional ways of classifying software quality models is to categorize them in terms of their method for organizing model of building blocks (Figure 1). From this perspective, the models can be classified into three different categories, namely, hierarchical, meta-model-based and statistical quality models [32].

A.1. Hierarchical Quality Models

The first quality models are the hierarchical models (McCall and Boehm) that model the quality of software as a hierarchy of quality factors and criteria according to Figure 2. The underlying concept to this category is the decomposition of the quality concept to some quality factor so that each factor covers a certain aspect of product or process quality.

However, it decomposes quality factors to limited metrics, so it cannot cover all quality aspects.

Furthermore, it is difficult to measure the metrics because of their abstraction. The interpretation of the result is too unclear because of ambiguity in decomposition rules. In another word, if the result of software quality assessment shows that the quality of software is not high enough, it is not precisely specified what can be done to improve quality.

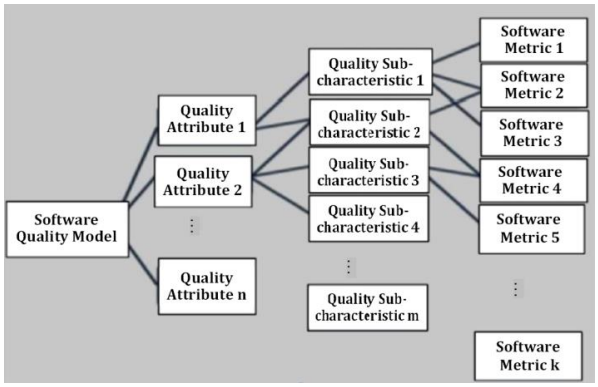

Fig. 2: The scheme of hierarchical software quality model.

For instance, they use functionality, maintainability, and efficiency factors to cover consumer requirements, ease of change after product delivery, and resource consumption, respectively. When the quality factors are more abstract, they are decomposed by the models to be less abstract. For example, ISO25000 decomposed the maintainability factor to the modularity, reusability, analyzability, modifiability and testability. So, given the inability to directly measure quality factors and sub-factors, it is essential to firstly identify the implementation metrics affecting each factor. As an example, ISO25000 uses coupling of components and cyclomatic complexity adequacy for modularity measure assessment. Finally, aggregation of implementation metrics measurements is used to identify the value of quality factors, and final product quality is specified by aggregating the measured values of quality factors.

McCall, ISO, Boehm, FURPS, and some other models fall into this category. ISO/IEC 25000 [50] is one of the most popular one among these models.

A.2. Meta-Model Based Quality Model

The second quality models are based on meta-model since the quality is a complex concept, and the quality model requires a more coherent and structured infrastructure than the hierarchal relationship between quality factors and metrics [38]. Meta-model, a model for model description is a model that includes building blocks and construction rules, which are required for creating a software quality model. In this type of models, the elements influencing the software quality, such as quality factors, criteria, and metrics, are identified such that they depend on modeling techniques. Then, the relationship and interactions between elements and the model interpretation are described more precisely than hierarchy via, say, UML diagrams or statistical equations. Figure 3 shows the hypothetical meta-model. This scheme indicates that the model is created on the basis of quality objective, factors, criteria, sub-criteria, and metrics. Moreover, each quality objective decomposes into several factors, and each factor can be measured indirectly or directly by affecting quality criteria or quality metrics, respectively.

The criteria are calculated by their mapping to one or more measurable metrics. Furthermore, the criteria can have an inheritance relationship with some sub-criteria. Meta-model helps the decomposition and organization of the quality model into quality involved elements. This kind of models is named meta-model-based quality model. Dromey [5] and SQUID [23] are two quality models which can be categorized in this group. According to Figure 4, SQUID considers the quality characteristics, sub-characteristics, internal property, and measurable property as quality model building blocks.

According to this model, each quality characteristic can be decomposed into a set of sub-characteristics. On the other hand, internal software property can affect characteristics and sub-characteristics. Finally, characteristics, sub-characteristics, and internal software property are mapped to measurable metrics to be used for calculating the quality of the final product by measuring metrics and considering their interactions. Dromey model is an extension of ISO9126 to create a tangible relationship between quality attributes and programming structures. This model adds new building blocks, called quality carrying properties, to the ISO9126. Based on this model (Figure 5), it should be determined

which quality properties are essential for each software component (structural forms including

programming language statements and components). After that, it should be specified which quality carrying properties affect high-level quality attributes. Finally, the quality factors underpinning high-level quality attributes are specified in order to measure the quality of a final product based on ISO9126, the quality factors which are influenced by high-level quality attributes are identified.

For instance, in structural forms of module definition, the critical quality attributes include functionality attribute, abstraction level, independence degree and the ease of reusability.

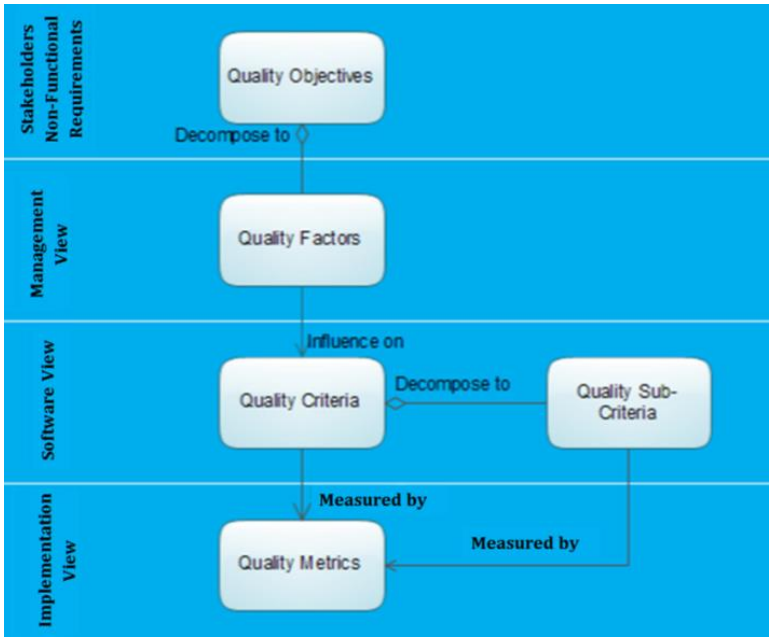

Fig. 3: The scheme of hypothetical meta-model.

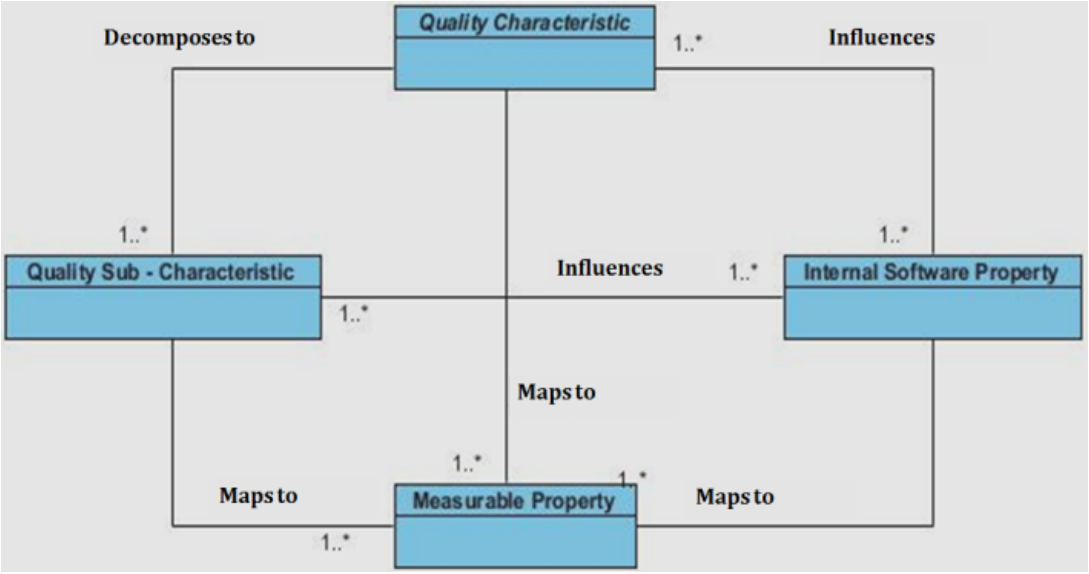

Fig. 4: The schema of hypothetical meta-model.

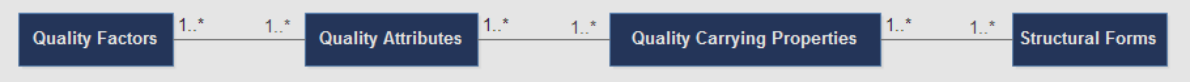

Fig. 5: Dromey meta-model.

The quality carrying properties related to these attributes are completeness, progressive (recursive modules), consistency, homogeneity, utilization, loosely-couple, parameterization, generic, abstractness, specifier, document and self- descriptiveness. In meta-model, each quality factor must be taken into account from several quality attributes. Each certain quality attribute is affected by a set of quality carrying properties. Several quality carrying properties can be applied to each structural form. The quality of meta-models might be compromised by the introduction of smells that can be the result of inappropriate design decisions. Each smell can be linked to the corresponding quality attributes. Recently, a research [14] presented an approach to defining extensible catalogues of meta-model smells. Such links are exploited to automatically select only those smells that have to be necessarily resolved for enhancing the quality factors that are of interest for the modeler. The implementation of the approach is based on the Edelta language, and it has been validated on a corpus of meta-models retrieved from a publicly available repository. To conclude the several meta-model discussed, we can drive the following corollary:

Corollary-1: A meta-model used in the quality model should consider only the quality model building blocks and the other quality related aspects, such as participants who are responsible for quality, artifacts which are used for measuring quality, the quality measurement time and software application domains, are not considered in the meta-model. Therefore, this type of models does not have enough comprehensiveness for creating a multi-purpose software quality model.

A.3. Statistical Models

The third category of quality models is statistical models.

factors underpinning high-level quality attributes are specified in order to measure the quality of a final product based on ISO9126, the quality factors which are influenced by high-level quality attributes are identified.

These models capture properties of product, process, and organization and estimate or predict these quality factors by creating a statistical relationship between different metrics which are measured. The reliability growth models (See [1], [35], [37], [30]) fall into this category. These models transfer the idea of hardware reliability models to software. The main purpose of this model is to monitor the failure behavior of software, for example, during software testing, and to predict how this behavior will change over time. Similar models are the maintainability index (MI) [3], a regression model from code metrics or Vulture [44], and a machine learning model predicting vulnerable components based on vulnerability databases and version archives. Bakota et al. [40] tries to aggregate expert's knowledge and copes with quality attribute's subjectivity by providing a statistical approach for measuring quality attributes.

This model utilizes the continuous function for identifying goodness of software instead of using a single value.

Comprehensiveness of the quality model, rather than being tied to its structure and building blocks organization, depends on covering different quality aspects and engineering tasks in the model. Therefore, regardless of the model structure, for having a comprehensive quality model, it is essential to have a variety of building blocks covering all engineering tasks such as product, process, design, code, and test and application domain. On the other hand, the role of different participants (analyzer, designer, architecture, tester etc.) which influence quality and different phases (requirement, analysis, design, implementation, test, and maintenance) that they participate in, and different artifacts (documents, code, models) that they are responsible to obtain should be considered in model building blocks selection. Independent of the techniques used for creating a quality model, it is essential to use meta-model for the quality model [3], because the meta-model is used for precisely describing elements composing quality model and their interactions. In other words, using the meta-model, with respect to Corollary-1, along the standard guide for decomposition of quality factors into their related criteria and sub-criteria can help in having a more structured quality model with clearer boundaries to factors, criteria, sub-criteria, and metrics.

B. Behavioral Classification

Since different quality models have different approaches to software quality, this section classifies different models in terms of their approach and assesses the capability of each category according to its ability to propose a comprehensive quality model for covering all quality-related aspects.

Although all quality models assess the quality of software, the comparison of models with each other is impossible because of the diversity of their approach. Some quality models are used for definition, some for assessment, and some for prediction of software quality. Therefore, there are three kinds of software quality models based on their approach including definition, assessment, and prediction of quality models [38]. Although the definition, assessment, and prediction are three different methods, the assessment without having a precise definition of software quality is difficult. Furthermore, software quality prediction is hard without knowing how to assess quality. This classification is called DAP according to Figure 6. Table 1 summarizes the

strengths, weaknesses, and highlights of different types of quality models.

B.1. Definition Models

The definition models are used along with different software-development process.

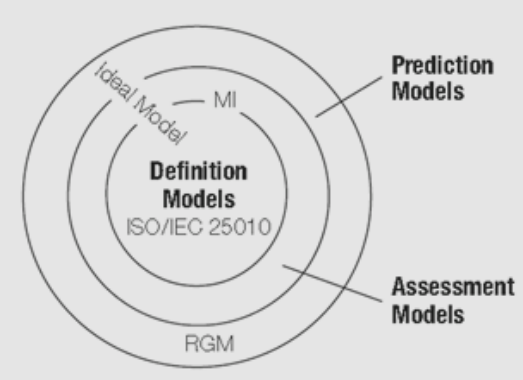

Fig. 6: DAP classification for software quality models.

In the requirement phase, all software requirements as well as quality requirements and method agreed with a customer about the concept of quality are identified [22]. In the design and implementation phases, the quality model is used as a basis for the identification of the designing and programming standards [5] to provide a product with high quality.

For example, FURPS model falls into this category. Four factors (functionality, usability, reliability, and performance) of this model are based on consumer's perspective, and the last factor (supportability) relies on developer's perspective.

B.2. Assessment Models

Assessment models often extend quality definition models to evaluate the qualities characterized in the definition model. During requirement engineering, assessment model can be used for objectively specifying quality requirements [22]. During implementation, quality assessment model can be utilized as a basis for all quality measurements, i.e., for measuring product, activity, and environment (See [5], [29], [32]). This measurement can be done by manual reviews [16] and systematic development and use of static analysis tools (See [5], [28]). Thereby, these models monitor and control internal measures that might influence external properties [22]. EMISQ model is an assessment model based on ISO 14598 for product assessment (See [28], [27]). EMISQ defines an approach to internal quality attribute evaluation such as maintainability. Due to the difference between EMISQ and ISO9126, it can be used as a reference model [38]. In this model, each quality attribute is decomposed into some sub-criteria and these sub-criteria are mapped on the quality metrics. Each sub-criterion is mapped on some metrics and vice versa. The metrics which are used for this model are measured by sharp tools such as PC-Lint and PMD.

One of the most important properties of this model is that its reference model includes about 1,500 different metrics that are mapped on the respective quality criteria. The approach also provides tools to create a customized quality model.

B.3. Prediction Models

The prediction models, in the software quality domain, are the models which are used as source code metrics or past defect detection data for predicting the number of defects of a system or specific modules, mean times between failures, repair times, and maintenance efforts. The reliability prediction is possible via Reliability Growth Models (RGMs).

These models can predict prospective maintainability of the system by the data of detected failure in test or operation phases (See [35], [36]).

The idea behind these models is that if we measure the failure times during system tests with an execution similar to the future operation, we will be able to interpolate them to the failure behavior on the field. Figure 7 illustrates this example data. It shows the calendar time on the x-axis and the cumulated number of failures on the y-axis. Each occurred failure is shown as a cross in the diagram. The curved line is, then, a fitted statistical model to the failure data. This model goes beyond the already occurred failures and is, hence, able to predict the probable future occurrence of a failure. This can, then, be expressed as reliability. There are various difficulties in applying RGMs. First of all, we need to decide on a suitable statistical model that adequately represents the actual failure distribution.

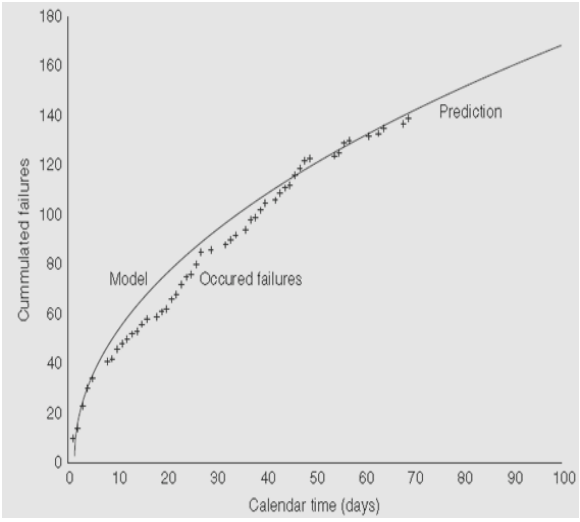

Fig. 7: An example prediction of a reliability growth model.

Table 1: The strengths, weaknesses, and highlights of different types of quality models

	Highlights Features	Weaknesses	Strengths
Definition Model	It can extend to assess and predict quality during development phases [22]	● The available models do not have a clear rule for quality factor decomposition to sub-factor and metrics (See [43], [16], [21], [22], [23]) ● Vague decomposition often leads to overlapping between the different quality factors [3]. ● The interaction of quality models with individuals participating in the project is unclear and non-transparent [3].	By providing direct suggestions for implementation, help in creating a high-quality system building blocks, which are required to have a high-quality system [38].
Assessment Model	The quality factors are defined by a level of sub quality and then are mapped to metrics. So, it is possible to map a sub-factor on a set of metrics and vice versa.	● Its relation with quality definition is unknown. Quality attributes are often too abstract, and it is hard to be automatically evaluated by software (See [43], [3]). ● It is hard to use quality factors in measurements (See [39], [22]). ● Despite providing the definition for metrics, quality models cannot describe the effect of metrics on software quality clearly [22]. ● Due to lack of clear meaning for factors and metrics, it is hard to aggregate those values in a hierarchical structure. ● Most existing methods do not pay any attention to fundamental rules of measurement theory. So, they can provide suspicious results [17].	During implementation they could be used as a basis for all qualitative measures such as a product, activity and environment (See [5], [29], [32])
Prediction Model	To predict the number of errors in a system or a particular component, the mean time between failures, the mean time to repair, or the amount of effort required for maintenance is measured.	● It is difficult to choose a proper statistical model that represents the actual distribution of failure. ● These models often lack a definition for basic concepts on which basis they are established. ● Most of them are based on regression analysis on a set of software metrics. The result of regression is a result of relationships that are difficult to interpret [18]. ● They rely heavily on context, and this complicates the selection of the software domain for their application. ● Another problem is the time measurement. Because the software does not fail over time, it is essential to use the software. ● Another problem is the lack of direct relationship with the definition model. For example, if system reliability is estimated to be low according to the prediction, it is unclear what action is appropriate to improve system reliability.	Using the data collecting during testing and other phases, future system reliability is predicted (See [35], [36]).

Furthermore, time measurement is a problem because the software does not fail just because clock time passes, but it has to be used. Furthermore, a direct relationship as a definition model is missing.

Hence, if the reliability of our system is predicted as too low, it is not clear what we have to do to improve it.

B.4. Multi-Purpose Models

Multi-purpose models are the models which use the same model for quality assessment and prediction with the models which are used for defining quality requirements. One of the rare examples of a multi-purpose model is the COQUAMO [23]. The COQUAMO, which is inspired from COCOMO, links the product, process, and organization-related metrics to the quality metrics by statistical models. Figure 8 shows the three primary measures used in COQUAMO: quality factors metrics, measures of quality drivers, and quality indicators. Each of this metrics has its own purpose. The quality factors with their quality factor metrics define quality and, together with target values, can be used to specify quality requirements.

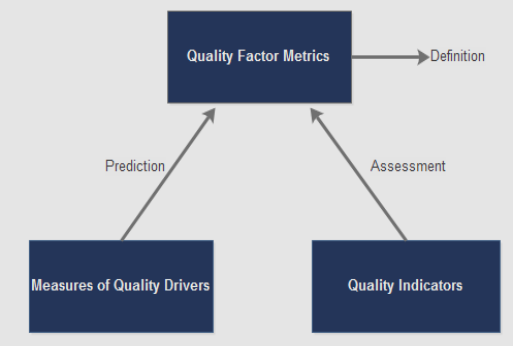

Fig. 8: Definition, assessment, and prediction in COQUAMO [38].

The quality factor metrics measure quality factors such as usability, testability, or maintainability, directly. For example, the quality factor reliability could be measured by the meantime to failure. The quality drivers include product attributes, such as the quality requirements; process attributes, such as the process maturity; or attributes of the personnel, such as experience. Similar to COCOMO, this model can estimate quality drivers using measures of quality drivers for an early prediction of the final quality of the software. The quality indicators are used for assessing the quality of the product during its development to monitor and control quality. These quality indicators measure attributes of the product directly, and through established statistical relationships to quality factors, they should give an indication of the current quality. For example, a quality indicator can be the number of called modules in a module or McCabe's cyclomatic complexity [24].

One of the most important responsibilities of the quality model is the early estimation of product quality for identifying corrective actions to improve product quality before the end of development. The quality requirements are defined based on the particular quality model, and related quality factors and measuring metrics are also identified on the same basis. So, the product quality is estimated and predicted by measuring the factors and metrics which are specified in the quality model.

Finally, if the assessment results show low quality, some actions should be applied to improve it. The identification of corrective actions based upon the evaluation results has two requirements. The first one is that elements influencing the product quality used for assessment and prediction should be clearly defined and their interaction should be identified precisely. The second one is that specified quality requirements related to measured quality factors and metrics should be specified so as to be able to determine the right corrective actions. To conclude the behavioral classification of quality models discussed, we can drive the following corollary:

Corollary-2: The integrations of quality definition, assessment and prediction are required for assuring consistency of quality evaluation results and corrective actions, which are applied to improve it. According to this Corollary, it is required to cover quality definition as well as its assessment and prediction in order to have a comprehensive quality model which includes all quality-related aspects and engineering tasks.

C. Basic and Derived Classification

According to Miguel et al. [25], quality models which have been proposed since 1977 are classified into two categories: basic and specific purpose models (Figure 9). The basic models, which are usually based on a hierarchy structure, can be applied to several kinds of software products.

The McCall, Boehm, FUPRS, Dromey, and ISO fall in this class. The specific-purpose quality models, which are the extension of basic models by factors and sub-factors modification, can be applied to a specific domain of software applications. Based on organization, requirements, and product specifications, the factors considered in specifically derived models may be different with basic models. The Berota, Georgiadou, Alvaro, and Rawashdesh fall in this category. This section evaluates the quality models based on this classification. Table 2 summarizes the quality models from the basic and derived points of view.

Although this model is a basic model, the considered aspects are inspired from McCall and Boehm to cope with their weaknesses [3].

C.1. Basic Model

Basic quality models are stand-alone. In other words, these models identify quality-related aspects based on their own approach and accordingly, include a set of quality factors, criteria, and metrics on the basis of the identified aspects.

McCall, the first proposed quality model, considers the operation, revision, and transmission aspects. So, it utilized 11 factors for covering these three aspects. Boehm, on the other hand, considers ease of use, ease of maintenance and ease of change in the environment and organizes a set of quality factors, criteria and sub-criteria at three levels to evaluate the quality of product based on these aspects. ISO9126 considers six quality related aspects including functionality, reliability, usability, efficiency, maintainability, and portability.

C.2. Derived Model

Derived quality models are created based on the specific basic model with modifications in its approach, aspects, factors or other kinds of building blocks to cover more specific than product or development process.

Also, some of the derived quality models provide a certain method for the assessment of specific quality factor. The main characteristic is that they are specific to a particular domain of application and the importance of features may be variable in relation to a general model [25].

Dromey model extends ISO9126 and tries to make the relationship between software product specifications and software quality specifications by adding a layer called quality carrying properties between software structural forms and ISO quality criteria. This model improves the final product quality by providing a guideline to define the certification in this area is more important than general definition of ISO9126.

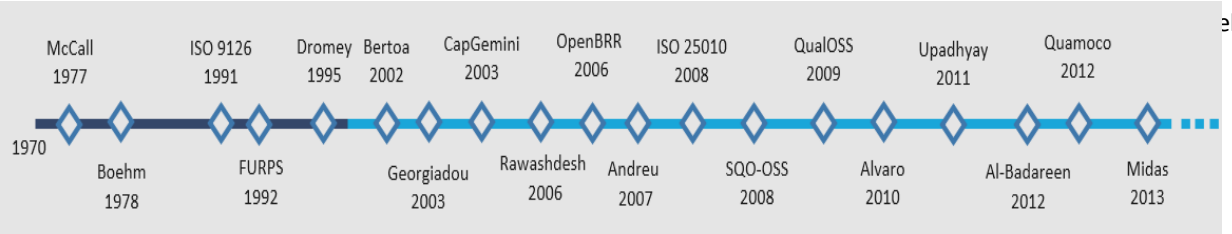

Fig. 9: Basic and specific purpose quality models.

Table 2: The quality models from structural, behavioral, and basic and derived points of view

	Structural Classification			Behavioral Classification			Basic and Derived Classification		
Model/Category [Reference]	Hierarchical	Meta-Model Based	Statistical	Definition	Assessment	Prediction	Basic	Derived	Based on
McCall [6]	✓			✓			✓		
Boehm[33]	✓			✓			✓		
ISO9126 [49]	✓			✓			✓		
ISO25000 [50]	✓			✓	✓		✓		
FURPS [34]	✓			✓			✓		
Dromey [5]		✓			✓			✓	ISO9126
Bertoa [8]	✓				✓			✓	ISO9126
Alvaro **Error! Reference source not found.**	✓			✓				✓	ISO9126
Alrawashdeh [12]	✓			✓				✓	ISO9126 – Dromey
Franch and Carvallo [19]	✓				✓			✓	ISO9126
SQO-OSS [46]	✓				✓			✓	ISO9126
Radulovic and Castro [45]	✓				✓			✓	ISO9126
SQUID [23]		✓			✓		✓		
Reliability Growth Models [37]			✓			✓	✓		
MI [3]			✓			✓	✓		
Vulture [44]			✓			✓	✓		
Bakota [40]			✓		✓			✓	ISO9126
EMISQ [27]	✓				✓			✓	ISO 14598
COQUAMO [23]			✓	✓	✓	✓		✓	COCOMO

Bertoa model is proposed based on ISO9126 for effective evaluation of COTS components by considering their quality. This model refines some of ISO9126's quality criteria based on COTS components requirements. For example, compliance criteria in the COTS domain foc us on standardization and certification. Learnability in ISO9126 is just about the effort required to learn how to use the software application, but in the COTS context, based on Bertoa model, it can be divided to time to use, configure, admin and expertize. Operability of software in the domain of COTS components includes effort for operability, tailorability and administration-ability, but in the general model, these criteria cover just effort for operating. SQO-OSS, based on ISO9126, is a hierarchical model that evaluates the source code and the community process allowing used in CBSD area by adding some criteria such as replaceability, adaptability, reusability, etc. model adapts ISO9126 for being used in CBSD area by adding some criteria such as replaceability, adaptability, reusability, etc.

Software Quality Models Analysis

In this section, we do an analysis on software quality models, in general and individual, to extract corollaries. In the first part, we analyze whole 19 investigated quality models based on their considered quality related aspects with respect to their quality factors which are included in them. Then, different basic quality models are examined individually to extract their weakness and opportunities to being comprehensive quality model. Table 3 shows a summary of the quality factors of investigated software quality models.

A. Software Quality Models: General Analysis

This section analyzes the models based on quality factors considered in different quality models in general, regardless of their classification. The models cover different quality related aspects by quality factors - ease of use and ease of maintenance are, for instance, covered by usability and maintainability factors, respectively. So, it is essential to extract popularity of different quality factors in order to have a clearer view on different quality related aspects that must be considered in a comprehensive quality model.

According to Table 3, each model assesses different aspects of product quality based on its own approach and structure. For instance, some special-purpose quality model such as RGM and MI assess reliability and maintainability of software, respectively. On the other hand, some general-purpose quality models such as McCall and ISO try to assess the quality of software with a closer consideration of quality-related aspects. Figure 10 shows the frequency of the consideration of each quality factors in one or more quality models. According to Figure 10, the following corollaries can be made:

- **Corollary-3:** From 31 factors which are taken into account in 19 different models, 58.06%, 12.9%, and 9.68% are considered in one, two, and three models, respectively.
- **Corollary-4:** Just 19.35% of factors are considered in more than 8 models.
- **Corollary-5:** The most widely used factors are the six factors of ISO9126 – reliability, maintainability, efficiency, usability, portability, and functionality.
- **Corollary-6:** The popularity of these factors can be related to the fact that the 55% of investigated models are derived models and 90% of them are based on ISO9126.
- **Corollary**-7: Quality factors are mostly measured based on source code.

According to the Corollary-6, the fact that 90% of the investigated models are based on ISO9126 implies the popularity of this model. This popularity has other causes. For example, a multiplicity of factors is considered in ISO9126 as well as the non-overlapping in the set of sub factors of each factor. Although this popularity can be considered as strength of ISO, it can also show a weakness of ISO because it can show the inability of this model to cover different quality-related aspects and the necessity of applying extension on that to cover the different application domains.

Assuming that ISO25000 improves the ISO9126 and covers its weakness, Figure 11 shows the compliance rates of other investigated quality models with ISO25000.

These rates are calculated according to some overlaps with ISO25000. For example, the value in the first bar on the left of the figure, there are only 5 factors out of 11 factors of McCall model have overlaps with 13 factors of ISO 25000, so McCall's compliance rate is 38% (5 * 100 / 13 = %38).

The other values are calculated in the similar manner. The ISO9126, Dromey, Alvaro, Alrawashdeh and Franch and Carvallo exhibit the highest compliance with ISO25000, where ISO9126 is previous release of ISO25000 and it is basis of ISO25000.

The Alrawashdeh model and Franch and Carvallo model use exactly ISO9126 factors for ERP quality assessment and software package selection, respectively. Dromey added the reusability factor to other ISO factors. Alvaro added a business aspect to ISO9126 for the evaluation of COTS quality.

From the discussion above, we can drive the following corollary:

- **Corollary-8:** Although ISO25000 includes the most important quality factors, it does not take some quality-related aspects such as cost of development, process related quality factors and business into account.

The fundamental of SQUID is that the software quality assessment is impossible without considering the software domain. So, this approach tries to create a relationship between the quality requirement of a specific product and the quality specification which is defined with quality models such as ISO9126.

Also, SQUID customizes ISO9126 based on software operational specification and its operational environment and uses the customized model for the assessment of certain software product.

MI models are called a set of quality models which are used to assess the maintainability of the software product. These models usually measure the ease of maintaining the software product by source code metrics such as Halstead. A vulture is a tool for analyzing the vulnerability of software product.

A comprehensive software quality model should cover all quality-related aspects in all kinds of software such as system software, application software, COTS, open source and so on. On the other hand, the importance of quality factors may vary based on the kind of software. For instance, the importance of reliability is different in system and application software. Therefore, not only should the model cover all quality-related aspects by different quality factors, but it can also support the specific weighting method to specify the importance of factors in specific domain such as Olsina et. al. [26] for specifying websites quality and Bansiya and Davis [13] for object oriented design quality assessment. Due to the lack of extension on some of the basic models such as McCall, Boehm, and Dromey for a variety of applications, these models cannot act as a

comprehensive model. On the other hand, according to the developments imposed by ISO9126 to cover the weaknesses of the model in evaluating the quality of different products, it can be concluded that ISO9126 model alone is not enough to operate in different domains. Also, in accordance of ISO25000 defects in considering of development cost and business aspects of quality in one hand, and the use of inappropriate methods for measuring some aspects such as context coverage, it is not suitable to use as a comprehensive quality model. Recently, a research (Dallal and Abdin [15]) conducted a systematic literature review including 76 primary studies investigating the impact of refactoring on several internal and external quality attributes, which were published before the end of 2015. In many cases, JDeodorant was used to extract code smell datasets from open-source projects.

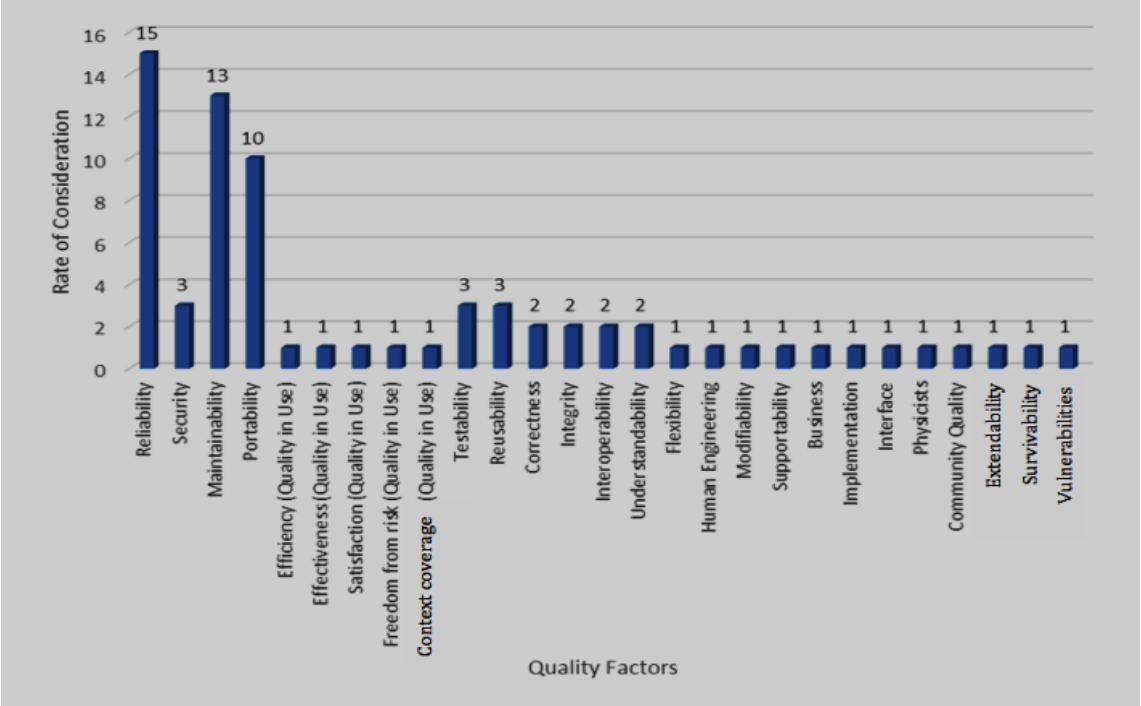

Fig. 10: Comparison of the rate of quality factor consideration in different quality models.

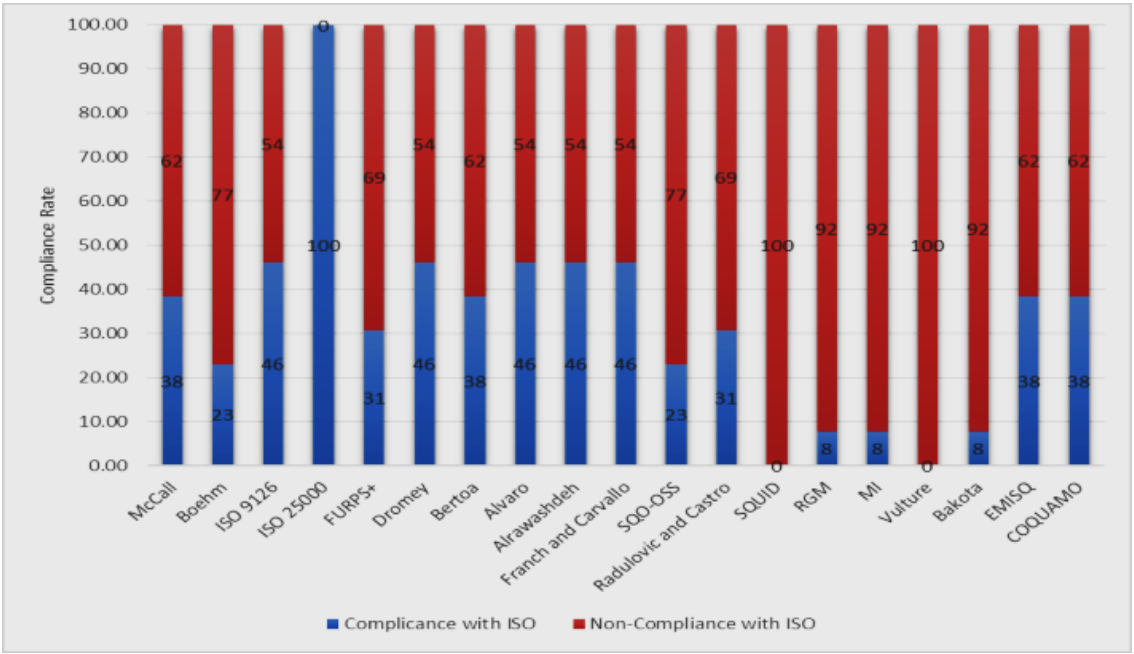

Fig. 11: Comparison of different model compliance with ISO25000.

Table 3: The quality factors of investigated software quality models

Factors/Models [Ref.]	McCall [24]	Boehm([33],[42])	ISO9126[49]	ISO25000[50]	FURPS+ [34]	Dromey [5]	Bertoa[40]	Alvaro**Error! Reference source not found.**	Alrawashdeh[12]	Franch and Carvallo[19]	SQO-OSS [46]	Radulovic and Castro	SQUID [23]	RGM ([35],[36])	MI[3]	Vulture[44]	Bakota[40]	EMISQ ([27], [28])	COQUAMO [23]
Functionality			✓	✓	✓	✓	✓	✓	✓	✓		✓							
Performance Efficiency	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓						✓	✓
Compatibility				✓															
Usability	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓							✓
Reliability	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓				✓	✓
Security				✓							✓							✓	
Maintainability	✓		✓	✓		✓	✓	✓	✓	✓	✓				✓		✓	✓	✓
Portability	✓	✓	✓	✓		✓		✓	✓	✓								✓	✓
Efficiency				✓															
Effectiveness				✓															
Satisfaction				✓															
Freedom from Risk				✓															
Context Coverage				✓															
Testability	✓	✓																	✓
Reusability	✓					✓													✓
Correctness	✓																		✓
Integrity	✓																		✓
Interoperability	✓																		✓
Understandability		✓																	✓
Flexibility	✓																		
Human Engineering		✓																	
Modifiability		✓																	
Supportability					✓														
Business								✓											
Implementation					✓														
Interface					✓														
Physicists					✓														
Community Quality											✓								
Expendability																			✓
Survivability																			✓
Vulnerabilities																✓			

B. Software Quality Models: Individual Analysis

The quality concept is complex and all software development process activities affect the final product quality. Therefore, it is important to identify the quality-affected aspects in order to offer a comprehensive quality model. For this reason, different quality models, regardless of their classification, are compared to identify the quality-related aspects they have considered, as well as the aspects that are not considered. This section evaluates the capability of quality models to cover all quality-related aspects to identify their comprehensiveness degree. The basic models are examined separately. But all derived models are examined in one section because of their diversity.

B.1. ISO9126

ISO9126 includes functionality, reliability, usability, efficiency, maintainability, and portability quality attributes. Conformance with requirements has two different issues – static and dynamic aspects. Static aspect is related to conformance with specified requirements. On the other hand, the dynamic aspect is involved with the future and non-specified requirements. ISO9126 includes attributes related to functionality (specified requirements).

As well, it considers ease of conformance with future and changing requirements (maintainability). So, this model pays attention to static aspects of quality alongside dynamic aspects of quality.

The consumer satisfaction and the cost of development are two important factors of final product quality. Although ISO9126 pays attention to customer satisfaction by considering suitability, compliance, and usability attributes, it does not have any metrics to measure customer satisfaction and development cost, directly. Moreover, this model does not include reusability which has a significant role in reducing construction cost [5].

ISO9126 does not consider different artifacts and development phases. So, it is impossible to evaluate various artifacts based on their conformance with the initial specification, and therefore, all metrics should be applied to all artifacts. Besides, because of lack of development phase-specific quality metrics, all metrics should be used in all phases.

Although improvement in development process leads to produce a high-quality final product, ISO9126 does not consider process quality-related metrics and all its attributes and the metrics are about product and its quality. Based on ISO's attributes and metrics, all participants in development process are responsible for implementing quality-related tasks. On the other hand, its attributes and metrics are not involved in application domain and so it can be applied to all kinds of software products (Behkamal et al. [7]).

B.2. ISO25000

ISO25000 considers two aspects of quality, quality in use and product quality. From quality in use point of view, it takes into account five quality factors called efficiency, effectiveness, satisfaction, freedom from risk and context coverage. On the other hand, from product quality perspective, it includes functionality, performance efficiency, compatibility, usability, reliability, security, maintainability and portability.

Model 25000 has improvements over the 9126 model. The factors considered in the product quality and quality in use aspects are extended to cover more quality related factors. For example, the compatibility and security factors are added to product quality model, and on the other hand, the satisfactions, freedom from risk and context coverage factors also were added to product quality model.

Generally, ISO25000 improves the ISO9126 from three points of view. First, it considers new factors, such as context coverage, freedom from risk and satisfaction and criteria, reusability for example, which is neglected in ISO9126. Second, some of criteria was introduced in the ISO9126, due to their importance, are presented as independent factors in ISO25000. Security was one of criteria subset of functionality factor, but in the ISO25000 it is presented as an independent factor, for instance. Finally, the third improvement has been done on ISO9126 is about its reorganizing. For example, co-existence criterion from subset of portability and interoperability criterion from functionality factor were moved to subset of compatibility factor in ISO25000. Despite all the improvements, there are still following flaws in this model:

- ISO25000 pays attention into static and dynamic aspects of quality related factors with considering functionality for measuring conformance with origin requirements (static) and maintainability, portability and compatibility for measuring easiness adaptation of future and unforeseen changes (dynamic).
- In this model, the consumer satisfaction was considered as an independent factor of quality in use, but the cost of development as an important goal of quality was omitted.
- ISO25000 still does not consider different artifacts and development phases.
- ISO25000 does not consider process quality-related metrics and all its attributes and the metrics are about product quality and quality in use.
- ISO25000 cope with different application domain of product with considering context coverage in

quality in use. But, it does not pay attention to variation of factor's importance based on product application domain. In other word, measuring the number of context with acceptable usability and risk over total number of required distinct context which is prescribed by ISO25022 (ISO/IEC 25022 [51]) is not an effective way to consider different scope of use. It is expected that the quality model is affected by application domain and model restructured based on the importance of factors or at least, the contribution of each factors in determining the final product quality be different based on application domain of product.

B.3. McCall Model

The McCall model includes eleven quality metrics to cover product revision, transmission, and operation aspects (Cavano and McCall [6]). The main idea of this model is about the relationship between external quality factors of product and its internal quality factors.

Because of the special position of maintainability, test, and flexibility in product revision aspect, this model specially considers the dynamic aspects of software quality via measuring ease of changes over the time. Although the completeness and traceability are parts of correctness, there is no any other attribute directly related to software functionality [7]. So, this model does not pay enough attention to static aspects of quality, and conformance with original specifications. Quality is a relative concept, and quantity is strongly related to different user's opportunity. McCall tries to consider user's opportunity via the ease of use (usability). But, customer satisfaction has no effect on determining the final quality of software in the model. Quality criteria considered by the McCall model evaluate the quality of the end product, and the internal measures are obtained based on source code. However, different artifacts are developed in different phases, each of which has an effect on the quality of the final product.

B.4. Boehm Model

Another popular commonly used model is Boehm software quality model (See [33], [42]). Boehm proposed a hierarchy structure for software quality same as McCall. This model consists of high-level, mid-level, and primitive quality attributes. High-level quality attributes indicate high-level quality requirements which are the basis for quality level assessment. Mid-level quality attributes include seven quality attributes for representation of expected quality of software systems. Primitive attributes are the foundation for quality metrics definition.

Rather than focusing on software functionality and compliance with specifications, this model focuses on maintainability. So, this model emphasizes dynamic aspect of software more than static aspect. Consumer satisfaction is not considered as an important metric to determine software quality. The only criterion for determining the user satisfaction is completeness which is indirectly reflected.

Boehm did not introduce separate quality attributes for different development phases, and measured just the quality of final product. Since most metrics are measured through the source code, other types of generated artifacts are not precisely measured. Accordingly, one can conclude that the role of other participants involved in the generating of software artifacts such as designers and architectures is not considered.

In this model, software application domain which plays a significant role in determining the quality objective is not taken into account.

B.5. Dromey Model

Dromey Model is an extension of ISO9126 with a set of quality carrying properties [5]. Quality carrying properties enable this model to apply to statement structural forms and programming components. These quality carrying properties are associated with ISO9126 high-level attributes. This model supports building quality into software, the definition of programming language specific standard, a systematic classification of qualitative errors, and development of code inspection tools.

This model considers the static aspect of software as well as dynamic aspect same as ISO. Besides, given that the focus has been on the source code and implementation and attempts have been made to implement through standard manner with the fewest qualitative shortcomings, coding standards enhance the flexibility to change code according to user requirements.

Also, according to the standard coding, attempts to reduce qualitative shortcomings lead to reduce duplications and development cost. In this model, different activities are done in various phases and their qualities are not taken into account. Moreover, the high-level quality attributes and quality carrying properties are related to the product, so the quality of development process is ignored in this model. The focus on source code in this model leads to ignoring the role of other people participating in the development of the final product. On the other hand, the quality carrying properties which are considered in this model are general and do not affect the application domain of software.

Table 4 summarizes the aspects considered by different basic quality models. In this table, each aspect in each model can be in fully considered (FC), partially considered (PC) or not considered (NC) status.

Table 4: A comparison of the considerations of different quality-related aspects in different models (fully considered (fc), partially considered (pc) or not considered (NC))

Quality related aspects	ISO 9126 [49]	ISO 25000 [50]	McCall [24]	Boehm [33], [42])	Dromey [5]
Static aspect of requirements	FC	FC	PC	PC	FC
Dynamic aspect of requirements	FC	FC	FC	FC	FC
Cost Reduction	NC	NC	NC	NC	PC
Increase Customer Satisfaction	PC	FC	PC	PC	PC
Development phases	NC	NC	NC	NC	NC
Process-related aspects	NC	NC	NC	NC	NC
Product-related aspects	FC	FC	FC	FC	FC
Different artifacts	NC	NC	NC	NC	NC
Role of different Participants	NC	NC	NC	NC	NC
Software Application Domain	NC	PC	NC	NC	NC

B.6. Derived Models

As regards, the derived models are proposed on the basis of some specific basic models and add some quality factors to them to cover different kinds of software applications such as COTS, ERP and so on. Therefore, it is essential to consider the quality-related aspects which are taken into account in derived quality models in order to have a comprehensive quality mode for the assessment of the quality of different types of software applications. In the COTS domain, the component adaptation to standards and certifications is important in component quality assessment. On the other hand, the time of using, configuration, administration and expertise of components are important.

Moreover, the effort of operability, tailorability and administrability of components impact the component's quality. Finally, the business perspective is crucially important in component quality evaluation. Examples include development time and cost, the time it takes to make the component available on the market, the targeted market volume, and how affordable the component is.

In the open source applications, it is important to consider the community quality via available documentations, update frequency, the rate of developer intake, the rate of developer turnover, and growth in active developers.

For the quality assessment of semantic web, it is imperative to consider some other aspects such as ontology, reasoning, discovery and so on. For example, the degree to which the software product can interchange ontologies, the accuracy of the reasoning process, semantic search and discovery process, the capability of the software product to provide appropriate response and processing times when working with ontologies, performing reasoning tasks, and performing search tasks are the aspects which are considered in quality evaluation of semantic web applications.

From the discussion above, we can drive the following corollary:

- **Corollary-9:** The investigated models do not consider the development phases. However, each phase has its own activities and the quality of product should be early measured during different phases. So, it is essential for a quality model to have phase-based quality attributes.

Discussion

This section summarizes all aspects that should be considered in a comprehensive model. The quality of software products is affected by product quality, development process quality, different development phases, application domain, and different artifacts.

According to the Corollary-4 and Corollary-5, the most common quality aspects are related to the quality of product.

However, the quality of software is affected by product as well as development process quality. Activity carried out during development process and the qualities of those activities are important in determining the final product quality. It is possible that a development process lacks appropriate quality monitoring activities. So, the consideration of this aspect can help in precisely and early assessment of the quality of product.

For example, if a development process does not have bug analyzing and tracking activities, the root of the identified bug (in the design phase, for instance) cannot be determined in prior phases. However, the identification of source of bug helps in the discovery of other potential bugs. Therefore, the quality model should consider the process-related aspects.

Based on the Corollary-9, software development process consists of different phases including requirement, analysis, design, implementation, test, delivery, and maintenance. Based on activities that should be performed in each phase, it is essential to measure certain quality attributes. For instance, in requirement elicitation phase, the quality attributes should measure the comprehensiveness and consistency of requirements; in the design phase, the architectural specifications such as reusability of modules or the amount of avoiding fault propagation in communication of designed modules should be considered in determining final product quality; in the implementation phase, source code-related quality attributes such as

source code readability, interface related quality attributes (usability), and the compliance of source code with design should be taken into account. This helps to have early estimation of software quality and to take corrective actions as soon as possible.

According to the Corollary-4, 19.35% of factors are considered in more than 8 models. This result shows that, although there are some general quality aspects which are applicable to all kind of products such as functionality, usability and so on, but according to the Corollary-3, 58.06%, 12.9%, and 9.68% of the investigated factors are considered in one, two, and three models, respectively. This result shows that, there are some domain-specific quality aspects such as business which are applicable on certain domain of products such as COTS. A comprehensive quality model should consider a wide range of quality factors, general and specific purpose, to be applicable on all products. Also, the quality model should have a mechanism for dynamic weighting for quality attributes to decide the impact degree of a specific attribute on final quality of the product.

According to the Corollary-7, various activities in different phases are carried out on different artifacts. For instance, the activities usually generate textual documents such as requirement specifications, scenarios and so on in the requirement and analysis phases, different models are usually generated in the analysis and design phases such as architectural design and so on, and the source code is developed in the implementation phases. So, the quality model should be able to cover the variety of quality attributes and their related metrics in a way that the measurement of each attribute in each phase is done by different types of artifacts such as documents, models, and code.

Finally, when the quality of a final product is measured, some properties should be assessed and others should be estimated depending on the time of measurement. For instance, after determining the software architecture, some attributes such as fault tolerance are measurable, but some others such as maintainability are predictable.

Conclusions

Nowadays, improvements in the quality of software are getting increasing importance. Software quality involves in several different engineering tasks and several participants who deal with quality concepts into the software life cycle according to their various roles, in the various phases and via different artifacts. According to the Corollary-2, a comprehensive quality model has to cover definition, assessment and estimation of software quality.

Our studies reveal that the existing quality models are not comprehensive enough because they do not consider all quality-related aspects. To identify all quality-related aspects in the software development process, we surveyed nineteen quality models and analyzed their quality attributes.

Finally, we analyzed their potential to be a comprehensive model.

We came to the conclusion that a comprehensive quality model should consider different aspects. These aspects include:

- Static and dynamic aspects of requirements,
- Development costs including both budget and time,
- Customer satisfaction,
- The role of different participants,
- The measurement time (different development phases),
- Product as well as process-related quality factors,
- A set of quality metrics measureable on the different type of artifacts such as document, model and source code,
- A specific mechanism in order to apply dynamic weights to quality factors to determine their impacts on final quality of a product based on its application domain.

In the future work, we are going to propose a comprehensive quality model which can cover all identified quality related aspects. This model will have some factors to measure static aspects of quality (conformance with origin requirements), as well as dynamic aspects (flexibility for future changes). Moreover, it will take some quality factors into account for considering development cost in determining final product quality. This model will pay attention to different participants and their roles in different development phases.

The proposed comprehensive model will measure the different artifacts. The model should support a dynamic weighting mechanism.

Author Contributions

M. Sadeghzadeh Hemayati did the study and prepared the first draft of this manuscript. H. Rashidi extended the draft and provided many helpful remarks. Moreover, he did proofreading the final version of this paper.

Acknowledgment

The authors gratefully acknowledge Z. Rashidi for their work on the original version of this document and prepared it in the requested format.

Conflict of Interest

The authors declare that there is no conflict of interests regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or

falsification, double publication and/or submission, and redundancy have been completely observed by the authors.

Abbreviations

COQUAMO	Constructive Quality Model
COTS	Commercial/Consumer Off-The-Shelf
ISO	International Organization For Standardization
IEC	International Electro-Technical Commission
UML	Unified Modelling Language
SQUID	Superconducting Quantum Interference Device
MI	Maintainability Index
EMISQ	Method For Internal Software Quality
FURPS	Functional And Non-Functional Requirements
RGMs	Reliability Growth Models
PMD	Programming Mistake Detector
FC	Fully Considered
PC	Partially Considered
NC	Not Considered
ERP	Enterprise Resource Planning

References

[1] N. Gorla, S. C. Lin, "Determinants of software quality: A survey of information systems project managers," Information and Software Technology, 52(6): 602-610, 2010.

[2] R. Nienaber, E. Cloete, "A software agent framework for the support of software project management," in Proc. Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology: 16-23, 2003.

[3] R. W. Hoyer, B. Y. Hoyer, "What is quality?" Quality Progress, 34(7): 52-62, 2001.

[4] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner., "Software quality models: Purposes, usage scenarios and requirements," in Proc. International Conference on Software Engineering Software Quality: 9-14, 2009.

[5] R. G. Dromey, "A model for software product quality," IEEE Transactions on Software Engineering, 21(2): 146-162, 1995.

[6] J. P. Cavano, J. A. McCall, "A framework for the measurement of software quality," in Proc. Software Quality Assurance Workshop on Functional and Performance Issues: 133-139, 1978.

[7] B. Behkamal, M. Kahani, M. K. Akbari, "Customizing ISO9126 quality model for evaluation of B2B applications," Information and Software Technology, 51(3): 599-609, 2009.

[8] M. F. Berto, A. Vallecillo, "Quality attributes for COTS components," I+D Computacion, 1(2): 128-144, 2002.

[9] Y. Fernández, C. Cruz, J. L. Verdegay, "A new model based on soft computing for evaluation and selection of software products," IEEE Latin America Transactions, 16(4): 1186-1192, 2018.

[10] D. Coleman, B. Lowther, P. Oman, "The application of software maintainability models in industrial software systems," Journal of Systems and Software, 29(1): 3-16, 1995.

[11] M. A. Akbar, J. Sang, A. A. Khan, F. E. Amin, M. Shafiq, S. Hussain, H. Hu, M. Elahi, H. Xiang, "Improving the quality of software development process by introducing a new methodology—AZ-model," IEEE Access, 6: 4811-4823, 2017.

[12] T. A. Alrawashdeh, M. Muhairat, A. Althunibat, "Evaluating the quality of software in ERP systems using the ISO9126 model," International Journal of Ambient Systems and Applications (IJASA), 1(1): 1-9, 2013.

[13] J. Bansiya, C. G. Davis, "A hierarchical model for object-oriented design quality assessment," IEEE Transactions on Software Engineering, 28(1): 4-17, 2002.

[14] L. Bettini, D. D. Ruscio, L. Iovino, , A. Pierantonio, "Quality-Driven detection and resolution of metamodel smells," IEEE Access, 7: 16364-16376, 2019.

[15] J. A. Dallal, A. Abdin, "Empirical evaluation of the impact of object oriented code refactoring on quality attributes: A systematic literature review," IEEE Transactions on Software Engineering, 44(1): 44-69, 2018.

[16] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J. F. Girard, "An activity-based quality model for maintainability," in Proc. IEEE International Conference on Software Maintenance (ICSM)): 184-193, 2007.

[17] N. Fenton, "Software measurement: A necessary scientific basis," IEEE Transactions on Software Engineering, 20(3): 199-206, 1994.

[18] N. E. Fenton, M. Neil, "A critique of software defect prediction models," IEEE Transactions on Software Engineering, 25(5): 675-689, 1999.

[19] X. Franch, J. P. Carvallo, "Using quality models in software package selection," IEEE software, 20(1): 34-41, 2003.

[20] E. Georgiadou, "GEQUAMO - A generic, multilayered, customizable, software quality model," Software Quality Journal, 11(4): 313-323, 2003.

[21] B. Kitchenham, "Towards a constructive quality model - Part 1: Software quality modelling, measurement and prediction," Software Engineering Journal, 2(4): 105-126, 1987.

[22] B. Kitchenham, S. L. Pfleeger, "Software quality: The elusive target," IEEE software, 13(1): 12-21, 1996.

[23] B. Kitchenham, S. Linkman, A. Pasquini, , V. Nanni, "The SQUID approach to defining a quality model," Software Quality Journal, 6(3): 211-233, 1997.

[24] T. J. McCabe, "A complexity measure," IEEE Transactions on Software Engineering, SE-2(4): 308-320, 1976.

[25] J. P. Miguel, D. Mauricio, G. Rodríguez "A review of software quality models for the evaluation of software products," International Journal of Software Engineering and Applications (IJSEA), 5(6): 31-54, 2014.

[26] L. Olsina, G. Lafuente, G. Rossi, "Specifying quality characteristics and attributes for websites," Web Engineering): 266-278, 2001.

[27] R. Ploesch, H. Gruber, G. Pomberger, M. Saft, S. Schiffer, "Tool support for expert-centred code assessments," in Proc. The 1st International Conference on Software Testing, Verification and Validation (ICST): 258-267, 2008.

[28] R. Plösch, H. Gruber, A. Hentschel, C. Körner et al., "The EMISQ method and its tool support-expert-based evaluation of internal software quality," Innovations in Systems and Software Engineering, 4(1): 3-15, 2008.

[29] J. Tian, "Quality-evaluation models and measurements," IEEE software, 21(3): 84-91, 2004.

[30] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, A. Blenk, W. Kellerer, C. M. Machuca, "Assessing the maturity of sdn controllers with software reliability growth models," IEEE Transactions on Network and Service Management, 15(3): 1090-1104, 2018.

[31] A. Alvaro, E.S. de Almeida, S.R. de Lemos Meira 2010. A Software Component Quality Framework. ACM SIGSOFT SEN 35, 1(Nov. 2010), 1-18, 2010.

[32] S. Wagner, Cost-Optimization of analytical software quality assurance: models, data, case studies, VDM Verlag, 2008.

[33] B. W. Boehm, Characteristics of Software Quality, 1, North-Holland, Amsterdam,:169, 1978.

[34] R. Grady, D. Caswell, Software metrics: establishing a company-wide program, Prentice Hall, 1987.

[35] M. R. Lyu, Handbook of Software Reliability Engineering, 222, CA: IEEE computer society press, 1996.

[36] J. D. Musa, Software Reliability Engineering: More Reliable Software, Faster and Cheaper, Tata McGraw-Hill Education: 632, 2004.

[37] J. D. Musa, A. Iannino, K. Okumoto, Software Reliability: Measurement, Prediction, Application. McGraw-Hill, Inc., 1987.

[38] S. Wagner, Software Product Quality Control, Berlin: Springer: 1, 2013.

[39] C. Frye, "CMM founder: Focus on the product to improve quality," 2008.

[40] A. Alvaro, E. S. Almeida, S. R. L. Meira, "Towards a software component quality model," presented at the 5th International Conference on Quality Software (QSIC), Bangalore, India, 2005.

[41] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, T. Gyimóthy, "A probabilistic software quality model," presented at the 27th IEEE International Conference on Software Maintenance (ICSM): 243-252, 2011.

[42] B. W. Boehm, J. R. Brown, M. Lipow, "Quantitative evaluation of software quality," in Proc. The 2nd International Conference on Software Engineering: 592-605, 1976.

[43] M. Broy, F. Deissenboeck, M. Pizka, "Demystifying maintainability," in Proc. The 4th Workshop on Software Quality: 21-26. ACM Press, 2006.

[44] S. Neuhaus, T. Zimmermann, C. Holler, A. Zeller, "Predicting vulnerable software components," in Proc. The 14th ACM Conference on Computer and Communications Security: 529-540, 2007.

[45] F. Radulovic, R. García-Castro, "Extending software quality models - A sample in the semantic technologies domain," in Proc. The 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE): 25-30, 2011.

[46] L. Samoladas, G. Gousios, D. Spinellis, I. Stamelos, "The SQO-OSS quality model: Measurement based open source software evaluation," in Proc. IFIP International Federation for Information Processing, 275: 237-248, 2008.

[47] ISO/IEC., Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (Square) - System and Software Quality Models, 2011.

[48] ISO/IEC 14598: Information Technology – Software Product Evaluation, 1999.

[49] ISO/IEC 9126-1., Software Engineering - Product Quality -Part 1: Quality Model, International Organization for Standardization, Geneva, Switzerland, 2001.

[50] ISO/IEC 25000, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE). International Organization for Standardization, Geneva, Switzerland, 2014.

[51] ISO/IEC 25022, Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) - Measurement of quality in use, International Organization for Standardization, Geneva, Switzerland, 2016.

Biographies

Mohammad Sadeghzadeh Hemayati was born in Ardabil, Iran, in 1981. He received the B.Sc. degree in Software Engineering from the Islamic Azad University, Tabriz, Iran, in 2008, the M.Sc. degree in Software Engineering from the Islamic Azad University, Tehran, Iran, in 2011, and he is attending the Ph.D. degree in Software Engineering from the Islamic Azad University, Qazvin, Iran, from 2011 up to know. In 2010, he joined the department of Electrical, Computer and Information Technology, the Islamic Azad University as a Lecturer. His current research interests include software quality, software testing, non-functional requirements in service-oriented architecture and dynamic software product line.

Hassan Rashidi is an Associate Professor in Department of Mathematics and Computer Science of Allameh Tabataba'i University. He received the B.Sc. degree in Computer Engineering and M.Sc. degree in Systems Engineering and Planning, both from the Isfahan University of Technology, Iran. He obtained Ph.D. from Computer Science and Electronic System Engineering department of University of Essex, UK. His research interests include software engineering, software testing, and scheduling algorithms. He has published many research papers in International conferences and Journals.

How to cite this paper:
M. Sadeghzadeh Hemayati, H. Rashidi, "Software quality models: A comprehensive review and analysis," Journal of Electrical and Computer Engineering Innovations, 6(1): 59-76, 2018.
DOI: 10.22061/JECEI.2019.1076
URL: http://jecei.sru.ac.ir/article_1076.html