Conceptualizing Quality in Software Industry

Nermine M. Khalifa, Mona M. Abd Elghany

Abstract— **This paper investigates the different software quality perceptions from the different stakeholders' perspectives and presents a critique to previously developed quality models and measurement theory frameworks associated. It emphasizes the rationale beyond the selection of the Goal Question Metric (GQM) as an evaluation method for the development of the software project with the desired quality needs satisfying the software system. Then it ends up with several concluding remarks that pinpoint the main discussion points and offers guidance for further research.**

Index Terms— **Software; Quality; Perception; Evaluation**

I. Introduction

Over the last twenty years, there has been an increasing emphasis on quality in developing software [1], [2], [3], and [4]. Software quality aspirations are more likely to be achieved with a greater emphasis on customer satisfaction through studying customer wants and needs, gathering customer requirements and measuring customer satisfaction [5], and [6]. The interest in quality is heightened as more system failures are attributed to issues in software quality that often lead to higher maintenance costs, longer cycle times, customer dissatisfaction, lower profits and loss of market share [7], [8], and [5].

Although the importance of quality is acknowledged, managing quality efforts remains a major challenge in software development. Despite the fact that most software developing firms collect quality performance measures such as customer satisfaction, but no operational measures are available for the quality attributes of software projects.

The discipline of software metrics entails identifying various attributes that need to be measured and determining how to measure them in developing quality software [8]. Quality metrics must be utilized and tightly coupled with the software development process. Identifying the applicable body of knowledge required is the first step in equipping software engineers with the essential skill set. The transition from defining the "why" business objectives to defining the "what" business or functional requirements is the most challenging phase. A business analyst should elicit requirements from different stakeholders to discover their concerns and needs.

Nermine M. Khalifa. is an assistant professor in Business Information Systems (BIS) Department in the College of Management & Technology, Arab Academy for Science & Technology, Alexandria, Egypt, e-mail: nerminek@gmail.com

Mona M. Abd Elghany is an assistant professor in Financial & Accounting Department (FAD) in the College of Management & Technology, Arab Academy for Science & Technology, Alexandria, Egypt, Telephone: +2/035565429, e-mail: mabdelghany2000@gmail.com

DOI: 10.5176/2251-3043_3.3.276

Software measurement theory addresses the issue of whether the proposed metrics are valid with respect to the attributes of the software entitled to measure. These evaluations are based on the properties of the measurement scales. Various development artifacts, such as requirements, design and code documents have disclosed the majority of software faults prior to testing and enhanced the ability to make meaningful assessments and predictions of software product quality. As a result, the principle of software measurement is the use of substitute measures that could be available early during evaluation process and hypothesized to be representative of the measures of real interest.

II. Software Quality Perception

Most of researchers argue that quality attributes can be measured absolutely and neglect its contrast with business objectives. On the whole literature deals with quality attributes as standard attributes applicable for all types of software regardless its fitness in various organizational setting and the perception of stakeholder for the quality term. Therefore, the fitness of software to organizational needs had been discussed by many researchers to reflect the effect of IT expertise and cultural issues on software appropriateness. The concept of ''IT-Business alignment" has been recently engaged to define the meaning of quality term and resolve the conflict of interest and preferences between the stakeholders. The emergence of aligning the software to organizational setting is mandatory in order to justify its investments. Reference [9] focused on the variance of quality perceptions between IT-expert and non-expert while reference [10] investigated same issue and included the managerial levels to indicate their perception. Further, reference [10]'s study differentiates between thirteen quality attributes to indicate their priorities.

Accordingly, some researches prefer to classify the non-functional into consumer-oriented and technical-oriented requirements [11]. Many researchers pointed that consumer-oriented attributes are relative one and cannot be assessed by absolute value. Customer acceptance coupled with the time consumed and cost burden are key issue to be considered while weighting software quality requirement [12]. In order to reach an acceptable level of software requirement, the desired values of attributes have to be defined in-advance. Fitness of software in certain organizational setting may differ and indicate a different scale of applicability for the same software used by different users. Different users are dealing with the same software from different perspectives so the term of software quality differs. Therefore, decision makers and system developers are required to set an acceptable level of software quality in accordance to the cost maintained.

Many researchers such as [13] indicate that it is hard to obtain software that can totally satisfy the quality requirements. Sometimes, direct more fund or human resource to improve quality of software might lead to negatives consequences. As a result, different quality measures may fit different business environment and needs so managers need to define the potential of quality measures for their business. Different priorities can be defined to software quality attributes according to its fitness in the organizational setting and its value from stakeholders' perspectives [10]. Software quality attributes are considered to be subjective and have a variable scale in conformity to stakeholders' perception [14]. ISO defines several quality attributes that are considered as controllable attributes, these attributes are relative and variable from different perspectives. Reaching an agreed upon definition of quality attributes and classifications of their specification would make it easier for stakeholder to prioritize quality attributes and address business objectives and requirements. As a result, optimized resource dedication and human factor utilization would be tangible benefits for such IT-Business alignment.

References [15] and [18] proposed hieratical models for quality measures, accordingly it was agreed that there is no need for estimating a measure subjectively. Preliminary level of quality model will be assessed so low-level attributes will be assigned to numerical value. Since, there is no formal relationship model that can indicate how to relate quality attributes with low level measures. Estimating time and effort needed for each task would provide an objective measure for each attribute [17]. Some researchers proposed comparison techniques in order to validate subjective measurements where this validation technique is to be used as an alternative of referring back to a conceptual model or piece of theory. The following sections presents a critique of quality models.

III. SOFTWARE MODELS' CRITIQUE

Quality models are still being criticized, due to the vague definition of quality models and the services they provide. There is scarcity of clearly specified requirements for quality models in reference to their application mode. If the requirements for quality models are obtained, then they can be benefited to guide further quality model development. It still continues to be unclear how quality models can be applied to define, assess and estimate the software quality.

Reference [18] discussed number of software quality models in their publication. The McCall 77 quality model specifies a number of aspects that can each be quantified according to several factors. McCall uses a grading scheme ranging from 0 (low) to 10 (high) and defines Usability as an aspect that proposes operability, training and communication. Reference [20] extended a quality model to describe "general utility" that can be decomposed into as-is utility (how easily the system can be used reliably and efficiently), maintainability and portability. Reference [19] developed a quality evaluation framework that investigates the quality of software components using the measurement of quality properties. These properties are used to evaluate the quality of the components. According to ref. [19], components can be considered as variables, functions, statements or requirements of the model.

Reference [21] proposed that quality models can be classified as taxonomic models like the ISO 9126 found in [22], metric-based models such as the maintainability index (MI) detailed in [23] and stochastic reliability growth models (RGMs) discussed in [24]. ISO 9126, for instance, provides a definition for software quality but does not give clues for assessment; the ML outlines an assessment nevertheless not so clear to quality definition. RGMs as well proposed estimations reliant on data that is not obviously related to quality definitions. The variety in software systems is extremely large, ranging from huge business information systems to tiny embedded controllers. These differences must be accounted for in quality models by defining a means of customization. In current quality models, this is not considered [25], [26], and [27].

During requirements' engineering, quality models should express quality attributes and requirements for desired software systems [28], and [29]. *"The earlier key quality attribute requirements are identified and prioritized, the more likely it is that the essential quality attributes will be built into the system. It is more cost-effective to reason about quality attribute trade-offs early in the lifecycle than later in the lifecycle when modifications are often difficult, impractical, or even impossible".* During implementation, quality models serve as basis of modeling and coding standards or guidelines to provide direct recommendations on system implementation and thus constitute constructive approaches to achieve high software quality [19]. During quality audits, they serve as a basis of the performed audit procedure. Thereby, internal measures that might influence external properties should be monitored and controlled [28].

Most taxonomic models rely on a hierarchical decomposition of quality attributes. This decomposition does not adhere to well-defined guidelines and can be haphazard [30], [21], [31], and [28]. Therefore, it is not easy to enhance quality attributes. Further the communication between the project developers and the software quality models is still imprecise. A generic method of such information communication is a set of guidelines and these guidelines are often not sufficiently concrete and detailed or the document structure of the guideline is not adapted to the application area. Additionally, the defined quality attributes are so intangible to be checked in a real software system [30]. For the reason that current quality models do not describe checkable attributes and even do not provide refining methods, they are difficult to be measured [32], and [28].

Metric-based models extend quality definition model usage to control compliance. During requirement engineering, they can only be used to objectively specify and control stated quality requirements [28]. Despite defining metrics, they miss to weigh the influence that specified metrics could have on software quality [28]. Another problem is that the provided metrics have no clear motivation and validation. Moreover, many existing approaches do not respect the most fundamental rules of measurement theory and, hence, are prone to generate dubious results [33]. Measurement is too important for controlling processes and simultaneously the quality attributes measurement is vital for an effective requirements engineering.

Most of stochastic models depend on regression through a set of software metrics [21]. This regression outcomes in equations that are not easy to be interpreted [34]. Furthermore, these models tend to be strongly context-dependent, also complicating their broad application in practice. Stochastic models are used during project management. More specifically, such models are used for release planning and in order to provide answers to the classical "when to stop testing" problem [35].

Based on the above critique of existing quality models, general requirements are derived as a quality criterion for a quality model. Any software quality model should state the procedures for its integration with the development tasks where further details are required [21]. The quality model is adequate only if it encounters the business goals and the standards set for the software product. The criteria for software quality could have more than one rationalization thus; there can be overlaps between them [21]. For example, the addition of an authentication mechanism may secure the data in a system but may also render its normal usage more difficult.

A satisfactory model should involve both the internal characteristics as well as the external detectable characteristics of the software product [30], [21], and [19]. The information accessibility to software engineers for implementation added to the detailed guidelines for assessable rules and structured guidelines should be included in an appropriate model. Quality criteria evaluation can be qualitative or quantitative nevertheless, it should be described with measures.

Quality requirements are different across software systems. Then quality models should specify the various required quality profiles for the software development [31]. The production of checklists for the criteria of software supports in configuring clear description. Conversely, the whole research area of software quality is diverse and fuzzy without a clear defined measure. Further there exist still open problems that have not yet been solved particularly in the adopted practice. That explains the reason for the subjectivity of existing quality models to several points of criticism.

Too often system requirements are not well understood. Understanding and bounding the requirements in a specification is an essential step to solving the problem of unmanageable projects. In particular, requirements specification drives effort required to build software systems and the time it takes to build them. Thus, software engineers can understand the sensitivity of requirements specification to the likelihood of producing a workable system. Combining functional requirements with effort estimation leads to a holistic understanding of feature, cost, schedule and trustworthiness.

When customers present ideas that need system solutions, engineers should have an ethical and professional obligation to help them define and simplify their problem. They must build the best solution to the customer's problem, even if the customer does not yet understand how to ask for it. For that reason, the customer should be encouraged to write a short outline that states the purpose of the system, its value and any constraints essential to making it useful leading to a complete set of requirements, which will emerge only through analysis, prototyping and validation with an iterative process. The requirements and design phases are important steps in a software project. If these steps are not carried out correctly, the quality of the final product will almost certainly be low. Investing in prototyping is a very helpful to find the way directly into the product quality.

Without an iterative plan for approaching the development of requirements, the design organization can find itself, months along on the project, developing the wrong software functions. For example, the designer of an ordering system could not guess that suppliers' invoices would not be directly related to orders because suppliers grouped orders for their own delivery convenience. Taking into consideration, the continuing stream of requirements changes can prevent coding and testing from moving along. So changes can be made in an orderly way in future releases after evaluation, but not by altering the requirements document. In brief, many design organizations do not have the necessary human factors specialists to analyze the users' tasks. Without specific attention to the people who will use the product, the organization can develop the wrong user interface.

Quality attributes impose specific constraints on software projects. Features that raise the software projects have to be considered from the earliest development stages. Whereas investigating software quality features is likely to be beyond the knowledge of most requirements engineers and developers. The proposed approach is based on developing a questionnaire that capitalizes on the key elements frequently used in the software quality features elicitation and specification processes. The use of those questionnaires provides requirements' analysts with a knowledge repository to ask the right questions and capture precise software quality requirements information. Because building quality attributes into a software system has a cost and calls for negotiation with users and other stakeholders about which quality attributes features should be included according to the software project type, the consequences of their inclusion, how to provide them, etc., it is more cost effective to reason about quality attributes trade-offs early in the lifecycle.

Dealing with quality attributes in the shape of non-functional requirements does not provide developers with enough information about what kind of artifacts to use to satisfy such requirements. The features represent particular functionalities that can be built into a software system. Since functional requirements describe the functions that the software is to execute, these functional requirements need to be explicitly specified, just like any other functionality. Consequently, the proper description of these functionalities in the requirements specification leads to the expected built-in into the system.

IV. COMPARSION BETWEEN SOFTWARE QUALITY MODELS

Reference [15] proposed the earliest software quality model. The model defines software-product qualities as a hierarchy of eleven quality factors, criteria and metrics. A quality factor represents a behavioral characteristic of the system. A quality criterion is an attribute of a quality factor that is related to software production and design. A quality metric is a measure that captures some aspect of a quality criterion. More than one quality metric are associated with each criterion such as portability is measured through aggregating modularity, self-descriptiveness, software independence and machine independence.

In 1987, reference [36] conceptualized the software quality s into decomposed parts of component till each can be expressed in terms of measurable attributes. Though he divided software quality into various factors; nevertheless, he did not suggest a universal set of concepts and measurements. Same as other researchers who portrayed quality in a hierarchical way as highlighted above.

After that efforts resulted in the development of a standard for software-quality measurement named ISO 9126. The standard consisted of six characteristics to establish an elementary set of independent quality characteristics. The ISO 9126 refines the standard's features into sub-characteristics and data elements to construct indicators to measure quality sub-characteristics. Indicators are represented in ratios brought from data elements. For instance, the ratio of number of faults can be used to define the fault rate. The ISO 9126 standard advised the direct measurement of characteristics, but did not specify how to do so.

The ISO 9126 model is different from the McCall model in the following. It uses other quality framework and terminology, the term "quality characteristic" is exercised instead of quality factor, the term "quality sub-characteristic" is used instead of criterion and data element indicator is used instead of quality metric in McCall's. Additionally the ISO framework is totally hierarchical regarding detectable quality aspects for the user, instead of internal software properties.

Nevertheless, the two models reveal the same problems: inadequate justification for the selection of the quality factors that should be encompassed in the quality definition as well as the quality criteria relative to a certain factor. The choice of quality characteristics and sub-characteristics also seems haphazard for instance; it is vague why portability serves as a top-level characteristic of ISO 9126, whereas interoperability serves as a sub-characteristic of functionality. Furthermore, there exists no verification confirming that the selected metrics impact the detected performance of the above quality factor. Moreover, the used terminologies consist of multiple names and represent highly theoretical terms in addition to their lack of clear concise definitions.

Reference [19] established a model built of components displaying a complete and consistent set of product properties resulting in the materialization of software quality attributes. He believes that hierarchical models using top-down decomposition are so unclear in their definitions of lower levels hence; providing little assistance to software developers for constructing quality products. Reference [37] describes his approach in that software engineers should construct components exhibiting a complete set of product properties results in the materialization of quality attributes. Dromey's model delineates the basis for the relationships between the internal quality properties of the software products and the external high-level quality attributes through stating the methodological procedures to be tracked.

V. SOFTWARE EVALUATION METHOD

The software quality assessment became a vital aspect in software development as discussed formerly. Continuous computer application could not be broadly employed without controlling efficient software. The software quality assurance for both developers and users is essential. Nevertheless, it is not easy to assess software quality in software engineering field. Software quality evaluation is a significant approach to further direct software quality forward independently of testing besides granting quantitative assessment of software quality.

The authors see that software quality is mainly characterized into external attributes in terms of practised application using the predicted implicit characteristics of software products that satisfies operating requirements and customers' demands. Software quality fundamentally impacts the application and its maintenance, that's why software quality has come to be the hottest issue in software engineering field. Software quality evaluation standard involves the following for software evaluation [38]: measuring software quality during development process, revealing current software condition, predicting the following-up as well as the provision of the powerful means to do so for the buyer, the developer and the evaluator. The activities involving evaluation could be generally acknowledged in the identification of the project type with the related software quality specifications as well as the definition the project plan [38]. Accordingly, the evaluation plan serves as a verification of the initial selected methodology. Nevertheless, the software developer views the precondition for software quality evaluation in that the development process should conform to software engineering standards.

There are a lot of factors influencing software quality for example: human factor, software demand, shortage in quality management, testing control, traditional custom adopted through the software developers, the development specifications, inadequacy in development tools and other. These factors are regarded with respect to administrators and developers [39].

Referring to the administrator, the factors influencing software quality involve: the inexistence of complete plan or efficient measures verifying quality and the lack of adequate concern to the quality from the beginning. Additionally, developer's personal does proceed irrelatively to his working performance due to the absence of good personal performance evaluation mechanism.

Whereas relating to the developer, the factors influencing software quality involve the product quality assurance is believed to be the responsibility of the quality inspector, which illustrates the missing idea of the total quality management. Also, the deficiency of ownership sense to realize the importance of the enlarged quality is to the organization development [40]. For instance, non-uniformity between products' versions comes from the inapplicability of the administrator's direction. Further, the lack of the idea of rendering customers satisfaction could be added due to unfamiliarity with customer's quality requirement [41].

To users, when the developer, develops an application and delivers it on time, it is far from enough with satisfaction. Users have not got appropriate indicators for software quality evaluation in addition to the developer who misses an indicator for productivity in software development. Thus, the users are incapable to evaluate correctly the working quality of the developer. Such evaluation method for software will lead to a life cycle shortage other than further development [42], [43], and [44].

Software demand analysis and preliminary design for software development customization should be established. Yet several software types such as control software, management software, educational software, internet software and others will require different weights on evaluation standards and quality requirements, as exemplified in [45] and summarized in table 1. During the requirements' analysis, preliminary design and development, specified metric units for rating quality elements and additional evaluation would be practical to organizations in collaboration with software developers. However, the ultimate purpose for software quality metrics remains in controlling cost and enhancing software development quality.

TABLE I. DIFFERENT FACTORS CONSIDERED BY DIFFERENT SOFTWARE

Software application features	**Factors to be considered**
Software requiring long lifetime period	Portability, Maintainability
Real time system	Reliability, Efficiency
Software needed to be applied in several environments	Portability
Banking system	Reliability, Functionality

Source: [45]

The users need to check if the software supplier company or the software developers have determined their quality metrics and evaluation data, if the constructed database has saved software associated requirements to proper industry and if it possess the adequate development expertise.

The Goal-Question-Metric (GQM) method results in defining the metrics needed for the software system, consisting of a set of rules to better understanding the perceived data [46]. The GQM method depends on such theory of a software organization requiring interpretation of each project and then describes operable data and finally carries a framework for data interpretation. The GQM method is a levelled structure, emerging from a high level goal. Following the software levelled evaluation method, the project manager, assigned to the current software, will rate the fundamental characteristic factors.

According to reference [47], GQM can confirm the appropriateness, uniformity and fullness of data collection and measure plan. Further GQM helps in the discussion of measure and the enhancement of goals, reliant on common interpretation, and at the end accomplishes an agreement that enable defining the widely accepted measure in the organization, which is principle of effective measure. Though the GQM technique provided a great help to defining realistic measure, yet it has some limitations. GQM method produces a measure definition through decomposing the goal. But the process of this decomposition is not clearly described and its quality is dependent on executor's experience. Reference [48] claimed the following limitations in the GQM method:

(1) Repetition is not granted: two different teams within the same organization can produce different measures even the same team can differ in the problem definition and measure again after several months to produce other measure.

(2) The termination time is not determined: so the final measure can be too large.

(3) The GQM application generates quite a great deal of problems with prioritized measures.

(4) In addition, there exists no description of the way of measure goal choice in the GQM. Then GQM can be viewed as a guiding principle towards the measure definition direction instead of a firm difficult engineering method for designing a measurable system.

The analysis of measure results extends the perception of modeling ability. The GQM method has been applied broadly in software industry; many companies have accomplished enhancement according to their practical experience to the GQM method, as demonstrated in [49]. However, GQM has not yet overcome the above mentioned limitations and measure maker should have a thoughtful understanding of the software desired in the organization to have meaningful segmentation of the goal measure.

VI. CONTRIBUTION TO KNOWLEDGE

The software quality attribute definitions according to the International Standard Organization (ISO) are relative to participant perspectives. For instance, maintainability, usability and portability could barely be designated objectively; they are associated with the adoption, maintenance and applicability of its use in different operational environment. Additionally, effort needed and time spent or resources consumed for maintenance operations varies in accordance to programmer expertise, program complexity, used tools and simplicity of documentation. Reference [50] considers important the relationship between the program modules and source code difficulty and the ability of applying further amendments.

Then software quality attributes are abstractly subjective. For instance; the definition of maintainability represents the simplicity to which maintenance tasks can be made.

Consequently, subjectivity will make their measurement and even the validation of their estimation systems challenging. Few statistically valid attempts to estimate and measure these attributes have been available. One of the followed approaches is to request experts to designate if the estimates of the estimation system confidentially comply with their insight without taking into consideration if the experts are able to generate direct consistent measurement thus, a statistical valid predictive system cannot be obtained. This paper argues that direct measurement of quality attributes has to be stimulated.

Other software quality attributes like readability, testability, understandability have served as alternative strategy for enhancing software engineering standards [34]. Moreover, modern developed approaches in software management and control entail ordinal scale measurement like Bayesian Belief Networks in which the measurement of the internal process attributes as code complexity and external attributes as reliability could be consistent on ordinal scales through their users [51] and [52]. However, reference [53] claimed that Bayesian statistical tests require a skilled statistical capability to practice than that most software managers and engineers do have. Reference [54] attempts to distinguish between software abstractions and their relative attributes that should be associated with these abstractions and earlier in the same year Circa explicitly tried to discuss the relationship between abstracted features and the external attributes but could not reach a comprehensive fundamental theory.

The theme of this paper agrees with the measurement frameworks proposed in [54], [53] and [55] and consistent with measurement theory provided in [56] and [57]. The authors demonstrate the correlation between an abstracted software artifact and the external quality attribute. Despite the fact that such demonstrations needs the experts' opinion for the assessment of the quality attributes yet it is different than the expert assessments employed to acquire quantified criteria for validation in [58] and [54].

There appears to be a consensus that the quality measurement is not fundamental, since this problem has not reached a solution through hierarchical models for almost the past 30 years as that of for instance references [16], [59] and [60]. Then there exists no measurement problem and all that is required is to describe a quality model measuring the lower level attributes and developing a mathematical theory to provide a quality attribute value. Nevertheless, the lower level attributes are subjective, e.g. flexibility, understandability etc. Further, there exists no theory providing a mechanism to grant that current hierarchical models measure quality attributes. Or in other words, no valid systematic relationships between the lower level measures and the quality attributes were acknowledged [55]. Due to the lack of such theoretical models other models such as COQUAMO have been developed [17]. These models rather focusing on objective tasks measurement relating effort and time to carry out tasks considered to be associated with the quality attribute of interest like the effort needed to learn how to use the software system as well as time spent for learning which represents the usability attribute [17], and [61].

For example, published papers produce predictive systems for maintenance effort (i.e. maintainability) such as that of reference [62]. Any maintenance effort measurement is affected by the time needed to perform a maintenance activity using a specific set of factors like the documentation available, the maintenance engineers, the testing extent made after maintenance and the expertise of the maintenance team. Yet, maintainability is a non-specified measure of the maintenance ease and maintenance effort should be measured through a specific set of activities that should be performed on any given system. Software developers need to obtain a measure for the software quality attributes that is simple to achieve through modest procedures for ensuring consistency.

It is challenging to establish consistency in case of subjective measurements. The problem occurs if the attribute needed to be measured is not well-defined. If the measure follows a representation condition then it serves as consistent measurement [56]. Consistent measurement is being capable of repeatedly allocate the same value to an attribute of an entity; in case of software development it is concerned on attributes of software product entities (artifacts) or process entities. Reference [57] declared that repeatability in measurement exactly does not at all times employ. Even for objective consistent measurement repeatability might be practiced but within specific limits of error.

Through the conduction of an experiment that supports the experts or the project managers to rate the true class of a set of entities without being biased. The introduction of bias could have happened if they were to convey their agreement with values generated by a predictive system. Such predictive systems for software quality attributes are appreciated at architectural and detailed design [58], and [62]. The estimates resulting from the true classes of the entities can be employed to develop an independent and unbiased assessment of a predictive system for a quality attribute. Predictive systems do not often tend to give definitive values for quality attributes of software entities.

Researchers and software developers tend to measure attributes or processes that are well-thought-out to be associated with quality attributes rather than direct measurement of software quality. This is due to their belief that direct measurement cannot be quantified and it would be just a waste of time and money. However, there occur few attempts for the quantification of quality attribute measurement consistency through a measurement theoretic approach. Current approaches for quality attribute assessment depend on the measurement of time and effort to accomplish tasks associated with the quality attribute as mentioned above. These approaches generate objective measures but do not provide the representation condition of measurement for the quality attribute.

This paper addresses the correlation between an abstracted objective measure of software artifacts and a subjective quality attribute. Experiments should be further encouraged as they are meaningful, they do provide a feedback loop to validate, modify and improve on the discussed theory. Also, further predictive systems can be developed for quality attributes through the measurement of the software artifacts.

VII. Concluding Remarks

To summarize in brief, the following act as concluding remarks of the above discussion. Quality attributes evaluation serves as a tool to cope with the increasing information demands of users. This paper directs the attention of the designers to the factors that affect the quality attributes of software projects and the needs of users and demonstrates how features evaluation is useful in discovering the just enough quality attribute of a software project. Valuable insights were gained and the paper suggested on the applicability and significance of software quality features evaluation in the field of information science. Its acceptance lies in the project managers' perception and attitudes towards the system and its ability to deliver the anticipated service.

Quality is a complex concept because it means different things to different people; it is highly context-dependent. Just as there is no way to satisfy everyone's needs, there is no universal definition of quality either. Quality is an elusive target hence; there exist no single simplified measure of software quality that is acceptable to everyone. In order to assess the software quality in an organization, the aspects of quality in field environment or in other words, the operating environment to which the software project system would be embedded into, must be identified and then be measured. By defining quality in a measurable way, it would be easier for other people to understand the notified quality and related business goals. Hence highly reliable software is determined by a mature process in a good successful business.

The assessment of the adequate quality level is vital in a software product. For example; errors that can be found in word-processing are certainly insupportable in nuclear-power software. An organization’s viability depends on the quality of the software developed. It should be recognizable how quality can influence the usage of the software product after being delivered and that the time and resources spent for high quality assurance would benefited greater market share. Measurements investigate whether the used techniques really improve software. The answer is reliant on the adopted quality improvement approach. Some companies follow a process-based approach and others the product-based one.

Reference [63] considered the perception of quality with respect to several domains, involving economics, marketing, operations management and philosophy. He determined that "quality is a complex and multifaceted concept" that can be defined relative to five perspectives. The transcendental perspective regards quality as "something that can be recognized but not defined". The user perspective regards quality as "fitness for purpose". The manufacturing perspective regards quality as "conformance to specification". The product perspective regards quality as "inherent characteristics of the product". And the value-based perspective regards quality as "reliant on the extent a customer is ready to expend for it".

Software specialists are required to generate products that satisfy users; this satisfaction denotes the struggling acknowledgment in the transcendental definition of quality. The user perspective is more tangible, substantiated in product characteristics that fulfill the user's needs. This quality perspective assesses the product in a task framework and can therefore be highly customized. Researchers perceive the users’ interaction with software products. The manufacturing perspective emphasizes the product quality during production and after being delivered. This perspective investigates if the product was built "right the first time" in an attempt to evade the costs related to rework during development and after being delivered. Dealing with process standards assure output consistency and can therefore institutionalize the production of end products like the manufacturing approach adopted through ISO 9001 [64] and the Capability Maturity Model [65] that support conformance to process rather than to specification. The product perspective of quality adopts that the internal product properties measurement and control will yield enhanced external product behavior.

Different groups engaged in software development can follow different perspectives. Customers normally adopt a user perspective, researchers adopt product perspective and the production department adopts manufacturing perspective. These perspectives supplement each other. If the user's perspective is identified plainly during requirements specification, the technical specification handling the production process can be followed on directly as product functionality and features. However, problems can be caused by changes to the requirements. This is where the value-based perspective of quality comes to be useful that is conveying quality to the extent the customer is prepared to expend accordingly evaluate the trade-offs between cost and quality. Design to cost perception is applied to revise requirements in regard to costs and benefits. Project managers are responsible for trade-offs evaluation between quality and cost. All quality aspects concerning user needs during requirements specification corresponding to the ISO definition of quality [66] that is "the whole characteristics of an entity that allow the capability of fulfilling specified and embedded needs".

Measurement is the key to achieving high quality software. The software engineer would apply the body of knowledge elicited to improve the quality of software throughout the development life cycle. In addition, the body of knowledge may be used as guidelines for practitioners, licensing of software professionals and for training in software quality measurement. Lack of knowledge could result in significant costs to the supplier in terms of unsatisfied customers, loss of market share, rework caused by rejected and returned systems and the costs to customers of faulty systems failed to meet their mission goals. The interpretation of the desirable properties of a software product in quantitative terms is an important part of the engineering activity in the modern world. Because once the product is released, it is no longer in a controlled test situation but instead in-practice with different users.

Measuring quality can formulate baselines, estimate quality and observe enhancement. When users visualize software quality, they often consider reliability, usability encountering ease-of-installation, learning-to-use. Reference [36] advises the direct measurement of these characteristics. For instance; learning time can be encapsulated as the average passed time (in hours) for a typical user to attain a specified skilled level. Users evaluate product quality in terms of their interaction with the final product. Such interaction is represented through their perceived satisfaction that is obtained by a combination of product's functions whether present or absent and the product's non-functional qualities. Developers and customers are similar in their interest to recognize as early as possible the likely quality of the final product. Developers should figure these internal properties into a product to display in the preferred external quality attributes.

Previous discussed quality models provide more like characteristics of an art rather than an engineering discipline. There occur no common acceptable guidelines for quality assessment. Several quality frameworks have been suggested in the literature, though no one has been broadly practiced in software industry or even appeared as a potential standard. Quality criteria are expressed in terms of abstractions with no detailed specification. It is difficult to operationalize in practice. Further, reference [67] ensured that standardization of concepts and terminology is missing in the software quality research, conveying the disjointed nature of the research area. Also, there exist very few references concerning software quality or quality management literature in addition to inconsistency with relevant international standards in software quality such as (ISO/IEC 9126). The Goal-Question-Metric (GQM) model is considered as one of the most effective models for measurement and evaluation in the software engineering field [68], and [69]. It offers a structured approach to develop metrics in a top-down manner, emerging from the high level goals then coming down to detailed metrics. The focus of GQM model is not particularly dedicated to quality; nevertheless it showed usefulness and effectiveness in different contexts.

The proposed discussion serves as the basis for estimating the quality of the final product system based on characteristics of internal quality properties through the adoption of the GQM approach. The GQM approach investigates the hypothesized relationship between quality attributes and software quality factors and provides understandings into the correlation between internal and external quality.

Most of the existing models adopted for software development process use the result of design, implementation and test phases; whereas the assessment of the software desired characteristics in the early phase of software development process would better support risk management and effort estimation associated with software projects. Since corporate enterprises are highly investing in information system and software development; henceforth the potential of IT-business alignment concept is more stressed. Poor alignment of IT applications with business objectives affects not only the potential of adapting IT solutions but the rank of its organization will consequently be affected as well.

A small amount of research has been done into the interrelation between software quality and business efficiency. The investment of businesses in technology, that has been adequately tested and evaluated, represents high risks. The investigation of the software alone is insufficient; case studies and experiments are needed to reinforcing technology evaluation. The business context of how it is used determines if investment in higher software quality is worthy. Reference [70] declared that "sometimes less-than-perfect is good enough". Business goals and priorities specify the acceptable level of "less than perfect". For the assessment of software quality in an organization, the aspects of quality in interest must be at first defined and then comes the decision of how to measure. The definition of quality in a measurable way makes it easier for other people to relate the adopted notion of quality to the desired business goals.

The appropriate definition of requirements for software projects in specifications of programmatic requirements in contractual documentation can avoid most of the problems arising after development phase during the practical use of the software. For a non-functional requirement, the software product does or does not meet the requirement is the mutual equivalent perception.

Large-scale, operational software products should be investigated, with the data required (ISO/IEC metrics) extracted from related documentation collected over a longer period of time. Larger sample sizes should be used to increase internal validity. Also, participants possessing greater experience with software development should be sought since those involved in the presented experiment had limited exposure to certain software projects. This means that participants with more constrained experience profiles should be required (for example, novice software engineers with experience ranging from 1 to 12 months and experts with experience ranging from 10 to 20 years).

VIII. Closing Suggestions

Among the researchers' knowledge bases, software quality content knowledge is uniquely their interest area of research, their adopted methodologies and developed tools to better estimate the quality of the software product and its associated effort, attain the primary importance in the diffusion of software engineering industry. In this paper, the provided evaluation method is expected to fulfill the user need to expand the software developers' acquisition of development knowledge at the early phase of requirements analysis and product design that can be applied and practiced. Although several methods and models do address various aspects of software quality, they are not usually implemented in a coordinated way because many are not actively supported by project managers. Additionally, the Goal Question Metric top down approach with its rated score prioritizes the needed quality features for the software development at hand and correspondingly the effort associated in person-months and resources devoted.

IX. Further Remarks

It should be taken into account to continue the evaluation of software quality attributes through other assessment methods. In any case, the quality attributes that would be selected and their associated features provide a starting point in cases where argumentation is needed for further improvement. Even small software projects should have the opportunity to be evaluated in order to be developed further.

It has been suggested that software quality can be defined from many points of view, depending on the role the person plays with the software and on the type of system being developed ([71]; [72]; [73]; [19]; [74]). In this manner, the voice of the customer is actively being pursued in order to produce a software product meeting its desired needs. Nevertheless, the business analysis and enterprise administration software types have to differentiate between user and developer stakeholder roles. Enterprise administration software ranks usability lower than reliability or integrity while users would have done the opposite. Similarly, developers for example highly rank maintainability more than any other stakeholder. Simultaneously, software quality priorities differ according to the type of software being considered. Thus, variation in software quality priorities varies by stakeholder role with the software type. Further research might be devised to use other stakeholder roles. If a suitably diverse population could be found then it would be beneficial to segment further categories and even distinguish between different roles within the same category.

References

[1] E. Duggan, "Silver pellets for improving software quality," *Information Resource Management*, Vol. 17(2), pp. 60-95, 2004.

[2] S. Haag, M. Raja, & L. Schkade, "Quality function deployment usage in software development," *Communications of the ACM*, Vol. 39(1), pp. 41-49, 1996.

[3] D. Harter, & S. Slaughter, "Quality improvement and infrastructure activity costs in software development," *Management Science*, 49(6), pp. 784-796, 2003.

[4] D. Prajogo, & A. Sohal, "The integration of TQM and technology/R&D management in determining quality and innovation performance," *Omega*, 34(3), pp. 296-312, 2006.

[5] S. Kan, V. Basili, & L. Shapiro, "Software quality: An overview from the perspective of total quality management", *IBM Systems Journal*, 33(1), pp. 4-19, 1994.

[6] W. Lin, & B. Shao, "The relationship between user participation and system success," *Information & Management*, 37(6), pp. 283-295, 2000.

[7] J. Arthur, *Improving software quality: An insider's guide to TQM*, New York: Wiley, 1993.

[8] A. Gopal, T. Mukhopadhyay, M.S. Krishnan, and D. Goldenson, "The Role of Communication and Processes in Offshore Software Development," *Communications of the ACM*, Vol. 45, pp. 193-200, 2002.

[9] M. Jeffery, & I. Leliveld, "Best practices in IT portfolio management," *MIT Sloan Management Review*, 45(3), 41–49, 2004.

[10] M. Haigh, "Software quality: non-functional software requirements and IT-business alignment," *Software Quality Journal*, (18) 361–385, 2010, DOI 10.1007/s11219-010-9098-3.

[11] A. Avizienis, J. C. Laprie, and B. Randell, "Fundamental concepts of computer system dependability," *IARP/IEEE-RAS workshop on robot dependability: Technological challenge of dependable robots in human environments*, Seoul, Korea, May 21–22, 2001.

[12] W. M. Gentleman, "Software quality world-wide: What are the practices in a changing environment," *Proceeding of the sixth international conference on software quality (6ICSQ)*, Ottawa, Canada, 1996.

[13] A. F. Shumskas, *Software risk mitigation total quality management for software*, New York: G.G.a. J.I.M. Schulmeyer, Van Nostrand Reinhold, (pp.190–220), 1992.

[14] S. W. Coniam, & Diamond, A.W., "Practical Pain Management - a guide for practitioners," Oxford, UK: OUP, (1995).

[15] J. A. McCall, P. K. Richards, and G. F. Walters, "Concepts and definitions of software quality factors in software quality". *NTIS*, (Vol.1), Springfield, VA: NTIS, 1977.

[16] B. Boehm, *Characteristics of software quality*, New York: North Holland, 1978.

[17] B. A. Kitchenham, and J. G. Walker, "A quantitative approach to monitoring software development," *Software Engineering Journal*, 4 (1), 2-13, 1989.

[18] M. A. Côté, W. Suryn, and E. Georgiadou, "In search for a widely applicable and accepted software quality model for software quality engineering," *Software Quality Journal*, 15(4): 401-416, (2007).

[19] R. G. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, pp. 146–162, 1995

[20] B. W. Boehm, *Software Engineering Economics*, Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[21] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, "Software Quality Models: Purposes, Usage Scenarios and Requirements," *Technische Universitat Munchen*, QuaMoCo Project, Germany, 2009.

[22] ISO/IEC, *ISO/IEC 9126-1:2001 Software Engineering: Product Quality - Quality Model*, International Organisation for Standardisation/ International Electro-technical Commission, 2001.

[23] D. Coleman, B. Lowther, and P. Oman, "The application of software maintainability models in industrial software systems," *J. Syst. Softw.*, 29(1):3–16, 1995.

[24] M. R. Lyu, editor, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill, (1996).

[25] E. Georgiadou, "GEQUAMO—a generic, multilayered, customizable, software quality model," *Software Quality Journal*, 11:313–323, 2003.

[26] S. Khaddaj, and G. Horgan, "A proposed adaptable quality model for software quality assurance," *Journal of Computer Sciences*, 1(4):482–487, 2005.

[27] Michael Kläs, and Jürgen Münch, "Balancing upfront definition and customization of quality models," *TUM*: 26, (2008).

[28] B. Kitchenham, and S. L. Pfleeger, "Software quality: The elusive target," *IEEE Software*, 13(1):12–21, 1996.

[29] S. Wagner, and F. Deissenboeck, "An integrated approach to quality modelling," *In Proc. 5th Workshop on Software Quality (5-WoSQ)*, IEEE Computer Society Press, (2007).

[30] M. Broy, F. Deissenboeck, and M. Pizka, "Demystifying maintainability," In Proc. 4th Workshop on Software Quality (4-WoSQ), pages 21–26, *ACM Press*, 2006.

[31] B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni, "The SQUID approach to defining a quality model," *Software Quality Journal*, 6:211–233, 1997.

[32] C. Frye, Focus on the product to improve quality, CMM founder, (June 2008).

[33] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Trans. Softw. Eng.*, 20(3):199–206, 1994.

[34] N.E. Fenton, and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Softw. Eng.*, 25(5):675–689, 1999.

[35] J. Musa, and A. Ackerman, "Quantifying software validation: when to stop testing? ," *IEEE Software*, 6(3):19–27, 1989.

[36] T. Gilb, *Principals of Software Engineering Management*, Addison-Wesley, Reading, Mass., 1987.

[37] R. G. Dromey, "Cornering the chimera," *IEEE Software*, Vol. 13, No. 1, pp. 33–43, 1996.

[38] R. S. Pressman, *Software engineering, a practitioner's approach*, Fourth Edition, McGraw-Hill Press, 1997.

[39] W. Pedrycz, and G. Succi, "Genetic granular classifiers in modeling software quality," *Journal of Systems and Software*, 76(3):277-285, 2005.

[40] Anders Henriksson, Uwe Aßman, James Hunt, "Improving Software Quality in Safety-Critical Applications by Model-Driven Verification," *Electronic Notes in Theoretical Computer Science*, Volume 133, Pages 101 - 117, 31 May 2005, available at: http://dx.doi.org/10.1016/j.entcs.2004.08.060

[41] F. Dongping, and L. Youcheng, “Structure Modeling and System Building of Self-adaptation Application Software System,” *Journal of Computer Engineering and Application*, 37(12), 2001.

[42] F. Dongping, Enterprise MIS System Component Developing Methodology Study, Beijing: Beihang University, 2001.

[43] T. Ying, Component Reuse Technique Research and Realization in the Development of Commercial Management Automation System, Beihang University, 2000.

[44] Li Jia, MIS System Development Method Study and Component Library Realization, Beijing: Beihang University, 2001.

[45] Z. Bosheng, X. Hong, and Z. Li, “Introduction to Process Engineering Principle and Process Engineering Environment,” *Software Journal*, Vol. 8, pp. 519-534, 1997.

[46] V. R. Basili, and M. W. Weiss, “A methodology for collecting valid software engineering data,” *Journal of IEEE Transactions on Software Engineering*, Vol.10 (No. 6), pp. 36-49, Nov. 1984.

[47] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach, “Practical Guidelines for Measurement-Based Process Improvement,” *Journal of Software Process Improvement and Practice*, Vol. 2(4), pp. 231-238, 1997.

[48] D. N. Card, "What makes for effective measurement,” *Journal of IEEE Software*, Vol. 10, pp. 94-95, Nov.1993.

[49] R. Solingen, and E. Berghout, “Integrating Goal- Oriented Measurement in Industrial Software Engineering: Industrial Experiences with and Additions to the Goal/Question/Metric Method (GQM),” *Proceedings of the 7th International Software Metrics Symposium*, pp. 178-186, 2001.

[50] J. Rosenberg, “Problems and prospects in quantifying software maintainability,” *Journal of Empirical Software Engineering*, 2(2): 173-177, 1997.

[51] N. Fenton and N. Maiden, “Making Decisions: Using BNs and MCDA,” Computer Science Dept., Queen Mary and Westfield College, London, 2000.

[52] P. C. Pendharkar, P. C. Subramanian, & J. A. Rodger, “A probabilistic model for predicting software development effort,” *IEEE Transactions on Software Engineering*, 31(7), 615–624, (2005).

[53] A. Baker, J. Bieman, N. Fenton, A. Melton, & R. Whitty, “A philosophy for software measurement,” *Journal of Systems and Software*, 12(3), 277–281, (1990).

[54] A. Melton, D. Gustafson, J. Bieman, & A. Baker, “A mathematical perspective for software measures research,” *IEE/BCS Software Engineering Journal*, 5(5), 246–254, (1990).

[55] B. Kitchenham, S. L. Pfleeger, & N. Fenton, “Towards a framework for software validation,” *IEEE Transactions on Software Engineering*, 21(12), 929–944, (1995).

[56] F. S. Roberts, “Measurement theory,” *encyclopedia of mathematics and its applications*, (Vol.7). Massachusetts: Addison-Wesley Publishing Company, (1979).

[57] H. E. Kyburg, *Theory and measurement*, Cambridge, UK: Cambridge University Press, (1984).

[58] D. Coleman, D. Ash, D. Lowther, & P. Oman, “Using metrics to evaluate software systems maintainability,” *IEEE Computer*, 27(8), 44–49, (1994).

[59] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in software quality,” Vols. I-III, Rome Air Development Centre, Italy, AD/A-049-014/015/055, *Nat'l Tech. Information Service*, Springfield, November 1977.

[60] T. Gilb, *Software metrics*, Cambridge, Mass: Winthrop, (1977).

[61] R. West, & K. R. Lehman, “Automated summative usability studies: An empirical evaluation,” In *CHI 2006 Proceedings—Automatic Generation and Usability* (pp. 631–639), Montreal, Quebec, Canada: ACM, 22–27 April 2006.

[62] L. Yu, S. R. Schach, K. Chen, & J. Offutt, “Categorization of common coupling and its application to the maintainability of the linux kernel,” *IEEE Transactions on Software Engineering*, 30(10), 694–706, (2004).

[63] D. Garvin, “What Does Product Quality Really Mean?,” *Sloan Management Review*, pp. 25-45, fall 1984.

[64] ISO 9001 - Quality Systems - Model for Quality Assurance in Design/Development, Production, Installation and Servicing, International Organisation for Standardisation, Geneva, 1994.

[65] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, “Capability Maturity Model, Version 1.1,” *IEEE Software*, Vol. 10, No. 4, pp. 18-27, July 1993.

[66] ISO 8402 - *Quality Management and Quality Assurance - Vocabulary*, International Organisation for Standardization, Geneva, 2nd Edition, 1994.

[67] S. R. Maier, *Organisational concepts and measures for the evaluation of data modelling*, in: S. Becker (Ed.), Developing Quality Complex Database Systems: Practices, Techniques and Technologies, Idea Group Publishing, Hershey, USA, 2001.

[68] V. R. Basili, G. Caldiera, H. D. Rombach, “Goal question metric paradigm,” in: *Journal of Encyclopedia of Software Engineering*, Marciniak (Ed.), John Wiley & Sons, Vol. 1, pp. 528–532, 1994.

[69] V. R. Basili, H. D.Rombach, “Tailoring the software process to project goals and environments,” in: *Proceedings of the 9th International Conference on Software Engineering*, Monterey, CA, USA, 1987.

[70] E. Yourdon, “When Good Enough Software is Best,” *IEEE Software*, pp. 79-81, May 1995.

[71] L. A. Arthur, Measuring Programmer Productivity and Software Quality, Wiley Interscience, Chichester, 1985.

[72] W. E. Deming, *Out of the crisis*, Cambridge, Mass.: Massachusetts Institute of Technology, World of W. Edwards Deming (Washington, DC: CEEPress Books, 1988).

[73] Michael S. Deutsch, and Ronald R. Willis, *Software quality engineering: a total technical and management approach*, Prentice-Hall, Inc., 1988.

[74] B. Boehm, & H. In, “Identifying quality-requirement conflicts,” *IEEE Software*, 13(2), 25–35, (1996).

Nermine M. Khalifa holds a PhD degree in Engineering & Information Science (2010) from Middlesex University, London, UK. She is specialized in E-business, E-commerce application and Supply Chain Management. She had many publications in the field of E-commerce, E-Supply Chain, Enterprise Resource Planning, E-Customer Relationship Management, Performance measures, System dynamic & Simulation, RFID and Global Supply Chain and Software Engineering. These papers have been published in international conferences, well-known journals and book chapters. She contributed as a reviewer in international conferences, Journals and book chapters such as: IADIS Conferences, IBIMA Conferences, ICOSCM & ICSCMIS conferences, System Dynamics Conferences, International Journal of Supply Chain, Journal of Supply Chain, IGI publications.

Mona M. Abd Elghany graduated from College of Management & Technology in Arab Academy for Science & Technology in Alex, Egypt, with B.Sc. in Business Administration (marketing and finance) and earned a M.Sc. in Total Quality Management, and a Ph.D. in Performance Measures. Current occupation is an Assistant Professor in Finance & Accounting department (FAD), CMT, AAST. Abd Elghany’s career path has encountered operation management researches, small-to-medium enterprises practices and policies, and she was formerly engaged in ISO consultancies. She participated in a number of academic papers in ranked conferences, journals and published books.