AN INTEGRATED GRAPHICAL ASSESSMENT FOR MANAGING SOFTWARE PRODUCT QUALITY

Luigi Buglione
Nihal Kececi
Software Engineering Management Research Laboratory
Université du Québec à Montréal
Montréal, Québec, CANADA

Alain Abran
Ecole de Technologie Supérieure
Université du Québec
Montréal, Québec, CANADA

ABSTRACT

Assessing software product quality has become more and more relevant and important to managers, even though it is still challenging to define and measure the detailed quality criteria and to integrate them into quality models. Software engineering standards can help establish a common language for these detailed criteria and, in parallel, implement a model of quality from its high-level concepts down to its lowest level of measurable details; in particular, the revised ISO/IEC 9126 suite of standards represents a useful taxonomy and framework for specifying software product quality. Several frameworks and techniques are being built on the basis of these standards. In particular, the GDQA (Graphical Dynamic Quality Assessment) framework and the QF^2D (Quality Factor through QFD) technique have been proposed to tackle software product quality analysis and measurement. This paper examines the structure of both and integrates them into an Integrated Graphical Assessment of Quality (IGQ) technique supporting quality assessments and related improvements through the full software lifecycle.

1. Introduction

Many software product assessment taxonomies have been proposed since the late '70s (McCall, 77) (Boehm, 78) including the ISO/IEC 9126 (ISO/IEC, 1991) standard on software product quality which, at that time, included only the ISO quality taxonomy and glossary. Some of the weaknesses of these initial taxonomies are the following:

- Their structure is hierarchical structure, and some have no standards for measurement or for causal (cause-effect) relationships;

- They have a static viewpoint, as they were initially structured for assessment at the end of the development process;

- In some instances, proposed checklists for identifying potential problems were based on the first two layers (characteristics and sub-characteristics) of the taxonomies, without taking into account the third layer (measures) for analyzing software quality and for investigating casual relationships in a quantitative manner.

Since then, the ISO in particular has developed, and is publishing in the 2001-2002 timeframe, a whole suite of improved 9126 standards, including a large body of measures specified from the ISO multiple viewpoints on software quality and on the basis of implicit high-level relationships across the phases of the development lifecycle. However, this new suite of ISO 9126 standards does not formally address the causal relationships of quality at the detailed level of measurement, either within one development phase or across phases.

In parallel, research has been carried out to represent and make use of the implicit causal quality relationships. This is done by setting up a graphical hierarchy to represent such relationships in specific quality assessments:

- *Graphical Dynamic Quality Assessment* (**GDQA**), and

- *Quality Factors through QFD* (**QF^{2}D**)

The aim of this paper is to investigate how these two pieces of research can be combined into a more robust approach for measuring and assessing overall software quality: **IGQ** (*Integrated Graphical Assessment of Quality*).

Section 2 presents a high-level review of some of the software product quality taxonomies, their commonalties and differences. Sections 3 and 4 introduce the GDQA framework and the QF^{2}D technique respectively, while Section 5 reviews their strengths and limitations, as well as their complementarities. The result of this analysis is summarized in an **IGQ** technique, supporting quality assessments and related improvements throughout the software lifecycle.

2 Software Product Quality Models

Software can be assessed using a series of predefined quality criteria (characteristics) by means of measurement. Since the end of the '70s, several quality models and taxonomies have been proposed (Table 1), some with 2 layers and others with 3:

- **2 layers** (Boehm, 1978) (McCall, 1977) (Dromey, 1995): these models and taxonomies comprise a set of characteristics, further subdivided into a set of sub-characteristics;

- **3 layers** (IEEE, 1992): this taxonomy comprises a set of characteristics and sub-characteristics, and a set of specific measures for each.

Table 1 illustrates how various authors, as well as ISO and IEEE standards-setting bodies, have used different terms and taxonomies for the various levels of their quality models. In Table 1, the parentheses around "Metrics" (lower row) indicate that the model architecture does not formally mention that layer, even though it is required for assessment.

Table 1. Terminology used in Software Quality Models

LAYER	MCCALL	BOEHM	ISO-9126:1991	IEEE1061	DROMEY
1	Factor	H-Level Charact.	Characteristic	Factor	H-Level Attribute
2	Criteria	Primitive Charact.	Sub-characteristic	Sub-factor	Subordinate Attribute
3	(Metrics)	(Metrics)	(Metrics)	Metrics	-

Another classification of QMs is based on how many relationships there are between the first two layers:

- **1:n relationship**, such as in ISO/IEC 9126:1991, where every characteristic has its own set of sub-characteristics;

- **n:m relationship**, such as in McCall's Factor-Criteria Model, where any quality sub-characteristic can be linked to one or more characteristics.

The 2001-2002 revision of ISO/IEC 9126 proposes three viewpoints of quality (Figure 1): internal quality, external quality and quality in use. For each quality viewpoint, a significant number of measures is proposed. This revision recognizes that there can be multiple quality viewpoints in the assessment of software. Each of these viewpoints can be associated with different groups of stakeholders, and can be taken into account concurrently in comprehensive assessment (the Manager, User and Developer groups) in order to determine the most important quality product attributes within each of their respective viewpoints[1]. Of course, other viewpoints could also be taken into account. However, the ISO 9126 model does not propose any technique for handling more than one viewpoint at the same time.

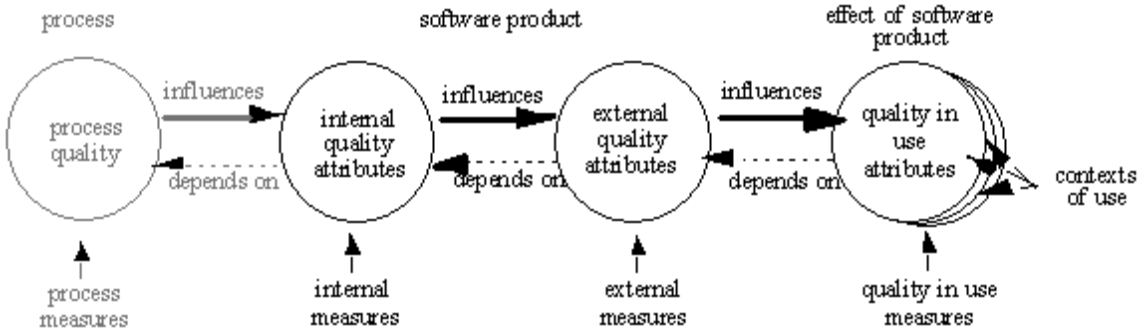


Figure 1. ISO/IEC 1926:2002 Software Product Quality Model

[1] An example in a totally different field is *wine evaluation*, in which "organolectic analysis" is used. Before expressing a final normalized value (on a 100 maximum point rating), three viewpoints must be taken into account at the same time: sight (20%), smell (28%), taste (52%), each classified into three or four subcriteria, to be rated on a 5-level scale (from 0 to 4) and then multiplied by a corrective coefficient.

3 GDQA: Graphical Dynamic Quality Assessment

3.1 Overview

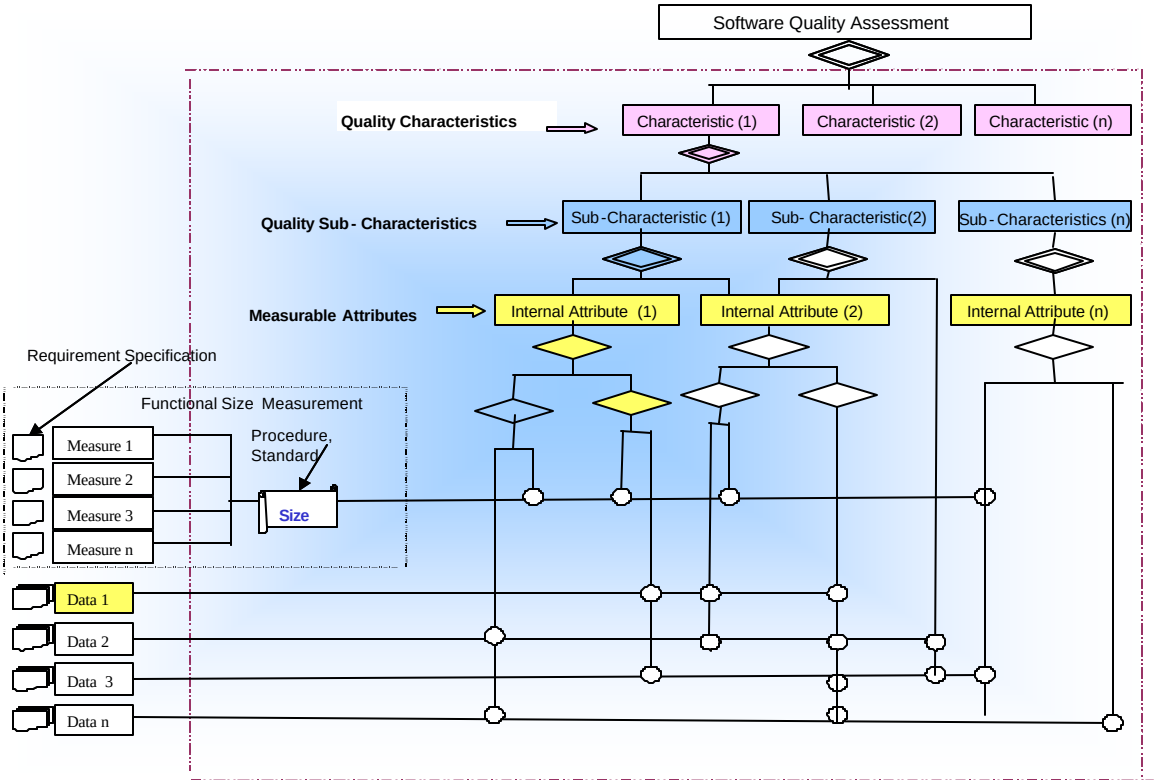
GDQA is a framework (Kececi, 2001) in which it is assumed that any high-level software quality requirement can be a function of many variables of whole-system characteristics; that is, there are causal relationships across the various quality characteristics and sub-characteristics. In any specific quality assessment, the relevant set of selected software quality characteristics, which can be process-, product-, management- and/or human-related, are classified into a hierarchical tree structure, as illustrated in Figure 2. The highest level of this structure consists of quality characteristics and the lowest level consists of measurable software quality attributes. The main focus in this approach is the identification of the relationships between the high-level quality characteristics and the primary data required to observe them in a quantitative manner (the top and left-hand sides of Figure 2 respectively). This representation of the implementation of the measures for the ISO 9126 model is described in Figure 2 and is based on three sets of concepts:

- **The selection of a quality model** (taxonomy and topology): The GDQA framework provides a hierarchical representation of the quality requirements of an existing quality model taxonomy, and topology: this is represented in the top portion of Figure 2 by the quality characteristics and quality sub-characteristics, and by the selection of specific measurable attributes of what must be assessed (including the identification and selection of quality requirements from multiple viewpoints).

- **Primary data** (direct measure/single numerical value): Primary data are generally single values collected from process documentation or system/software specifications, testing reports, etc. (e.g. number of errors, review effort in hours, etc.). The data can also come from the applications of well-documented measurement methods, such as is the case for the measurement of the functional size of the software (ISO/IEC 19761: 2002) (Abran, 2001). In the GDQA framework, the dots represent the numerical values derived from the primary data, which can then be used for calculating the numerical value of one or more attributes, as illustrated in Figure 2.

- **Functions**. Functions are defined as the transformation of some measures to represent the expected causal relationship of quality. They can be simple functions – simple ratios, for instance, based on primary data. They can also be complex functions with parameters combined according to predefined formulas. The functions represent the relationships in the hierarchies between the quality requirements, the measurable attributes and their values in specific instantiations. With GDQA, the relationships can be characterized as logical mathematical operations, as illustrated in Figure 2: the mathematical operators being represented by logical connectors, and their inputs by the dots indicating which primary datum contributes as a parameter in any specific function.

Figure 2.The Graphical Dynamic Quality Analysis (GDQA) framework



The GDQA framework has been derived from the core concepts of functional modeling techniques to identify the relationships between quality characteristics and primitive measures (data). The GDQA framework, as applied here to a software engineering context, captures core concepts of many of the 'systems' engineering approaches that are being widely used for analyzing complex engineering systems; as in system engineering. GDQA makes extensive use of hierarchy theory, the success/failure mechanism and functional modeling. Of course, multiple types of functional modeling from the system engineering of complex systems are available, such as the Goal Tree Success Tree (GTST), Dynamic Master Logic Diagram (DMLD) and GTST-DML (Kececi, 2001) (Modarres, 1999) (Kececi, 1998) (Kececi, 1999-a, b, and c).

3.2 Strengths

GDQA provides clear and dynamic views to stakeholders on how quality requirements, as specified in a specific quality model and taxonomy, can be defined and represented quantitatively. Such a graphical framework representation facilitates:

1) An understanding of the quality requirements, as defined in any quality model taxonomy;

2) The design of corresponding quality assessment strategies;

3) The use of a broad range of quality characteristics, sub-characteristics and attributes, and their measures – direct/indirect, external/internal – for the whole system;

4) Identification of the interrelationships between software-, hardware- and human-related characteristics which have an influence on the quality of the product;

5) Identification, through the dynamic nature of GDQA, of trends in quality by observing the time behavior of the variables;

The identification of common measures used to compute more than one quality attribute.

3.3 Limitations

Although GDQA focuses on the integration of quality characteristics and their measurable characteristics, as well as necessary data collection, it does not specify how they could be useful for different interest groups. Furthermore, multiple stakeholders can build their own individual and distinct quality models; GDQA does not tackle the issue of integrating multiple viewpoints of quality and, in such instances, does not provide for a consolidated assessment of the quality of a software product.

3.4 An example: Analyzing internal reliability measures using GDQA

In the ISO 9126 standard, software 'reliability', a characteristic of software quality, is subdivided into 6 sub-characteristics, one of which is labeled as 'maturity'; a number of distinct candidate measures are proposed to derive a numerical value for this sub-characteristic (see Table 2). ISO 9126, however, provides only a sequential, albeit extensive, inventory of candidate measures within its quality model; it is basically a catalogue of candidate measures. For a specific quality assessment, candidate measures from this ISO inventory must then be selected and the GDQA approach applied for the hierarchical representation of the quality requirements

Figure 3 illustrates an application of the GDQA framework for the 'reliability' characteristic of ISO-9126:2002. ISO 9126 defines the following as the sub-characteristics of reliability: maturity, fault tolerance and recoverability. The numerical functions proposed by ISO 9126 for these sub-characteristics are listed in Table 2. It can be observed in Table 2 that the measurable attribute defined as the "number of faults detected in review" is needed to compute three different measurable attributes, and their relevant measurement functions (i.e.: fault exposure rate in review, fault removal and remaining fault density). Each measurable attribute contributes in a distinct way to system reliability. This contribution is usually represented by a weight derived from the quality requirements. By contrast, the definitions of the measurement functions are defined by conventions in the selected quality models. Once both have been selected in a specific context of quality assessment, GDQA provides a graphical representation to visualize the expected complex relationships between the quality characteristics, sub-characteristics, measurable attributes, measurement functions and the base detailed measures required for its instantiation. This representation helps stakeholders (and developers) to develop a project measurement plan and to monitor software quality at any time during the development phases.

Table 2. Candidate measures for the 'maturity' sub-characteristic of the 'reliability' characteristic

Measurable attribute			**Data or Measures**	
Name	*Identifier*	*Measurement Function*	*Identifier*	*Data Description*
Fault exposure rate in review	MX1	MX1 = A1/B1	A1	Number of faults detected in review
			B1	Number of estimated faults to be detected in review
Fault Removal	MX2Y2	MX2Y2=MX2/MY2 MX2=-A2/B2 MY2=A2/B3	A2	Numbers of corrected faults in design/c oding….
			B2	Number of estimated faults to be detected in review…
			B3	Number of faults detected in review
Remaining Fault Density	MW3	MX3=(A3-A4)/B4	A3	Number of estimated faults to be detected in review
			A4	Number of faults detected in review
			B4	Estimated program size
Test Coverage	MX4	MX4=A5/B4	A5	Number of test items, which designed test cases covered, conformed in review
			B4	Number of test items which should be covered by adequate test cases

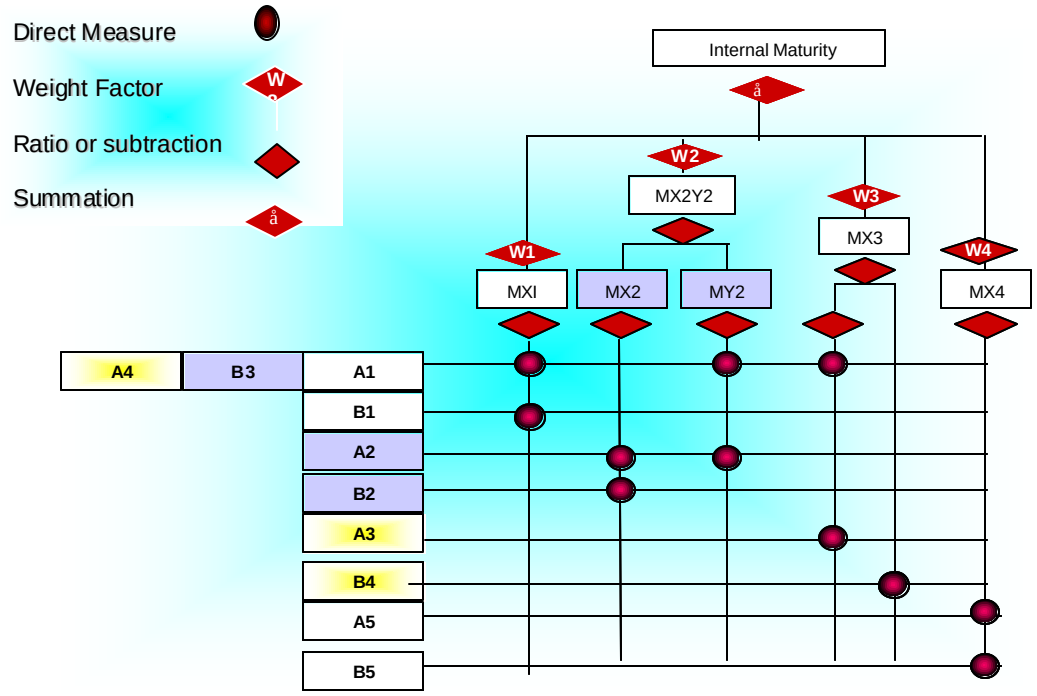


Figure 3. The GDQA framework for the ISO 9126 'maturity' measurement function

4 QF^{2}D: Quality Factor through QFD

4.1 Overview

The original QF technique (Buglione, 1999) allows the integration of multiple sets of quality assessment measures from several organizational viewpoints to derive a consolidated quality value; the next QF version (Buglione, 2001) makes available source data or intermediate results in addition to the consolidated final quality value. This provides the quality analyst with multiple levels of detail to enable him to understand the relationships across the many characteristics and sub-characteristics that impact quality. For his causal analysis, the quality analyst then needs to access all the sources of data for each different viewpoint which had been taken into account in the final consolidated result of the quality assessment. With QF^{2}D, it is possible to take into account several levels of analysis, depending of the objectives of the assessment and the level of granularity desired. QF^{2}D uses the key concepts of the House of Quality (HoQ) and of its related matrices to keep track of the intermediate results and calculations. QF^{2}D also uses Quality Function Deployment (QFD), a well-known method in the manufacturing field for better management of the initial product requirements prior to the production phase.

QF^{2}D provides for a normalized quality value and profile both prior to and following development of a software product, based on the assessment of a number of stakeholder viewpoints, and throughout the software lifecycle, along two main phases we called the D (development) and I (improvement) phases, as shown in Figure 4. The first matrix allows the targets (e.g. requirements) for the quality of the software product to be quantified and defined from the viewpoints (M, U, D)[2] on the basis of the ISO product quality characteristics, and corresponding quality sub-characteristics. These quality goals and quality indications will be taken into account in the development of the software product. Next, the software is analyzed, designed, coded, tested and assessed.

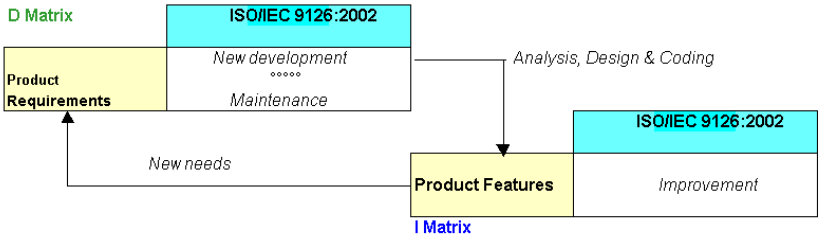


Figure 4. Development (D) and Improvement (I) matrices and QF^{2}D lifecycle

In the second matrix, the list of product features delivered is matched to the ISO sub-characteristics selected as target for the assessment of the product. Moving values between the two matrices provides feedback based on testing (from D to I matrix). In this way, QF^{2}D gives a company the opportunity to monitor the quality of a software product in a dynamic way throughout its lifecycle.

Figure 5 illustrates the structure of the two matrices with the three viewpoint dimensions (M, U, D) on the left (*n* possible people per group), together with their sets of quality requirements (Targets DES_i). Then, for each, a priority is assigned on a Likert scale (from 1 to 5), and then a rating is assigned for each of the sub-characteristics.

[2] M = Managers; U = Users; D = Developers

Table 3 presents the elements in the QF^{2}D matrices, which may be different depending on the lifecycle phases. For example, in the rows of the development/maintenance matrices, the requirements are the objects of assessment, while in the Improvement matrices, it is the features of the software product implemented that are being assessed. By contrast, for both sets of matrices, the elements of the columns, the list of ISO/IEC 9126 standard sub-characteristics (parts 2, 3 and 4) are the same. All targets specified in the requirements for the quality sub-characteristics are expressed using the ISO/IEC 14598-1 scale (from 0 to 3), rather than the usual QFD symbols used in the HoQ.

Table 3. Elements of the Development (D) and Implementation (I) matrices

	DEVELOPMENT (D) MATRIX	IMPROVEMENT (I) MATRIX
Software Lifecycle phases	• Requirements • Maintenance	• Testing (V&V Activities)
Object of analysis	Software Product (via process) through a TQM approach	
Whats (*rows*)	Targets of Requirements	Product Features implemented (after coding)
How's (*columns*)	ISO/IEC 9126:2002 sub-characteristics	ISO/IEC 9126:2002 sub-characteristics

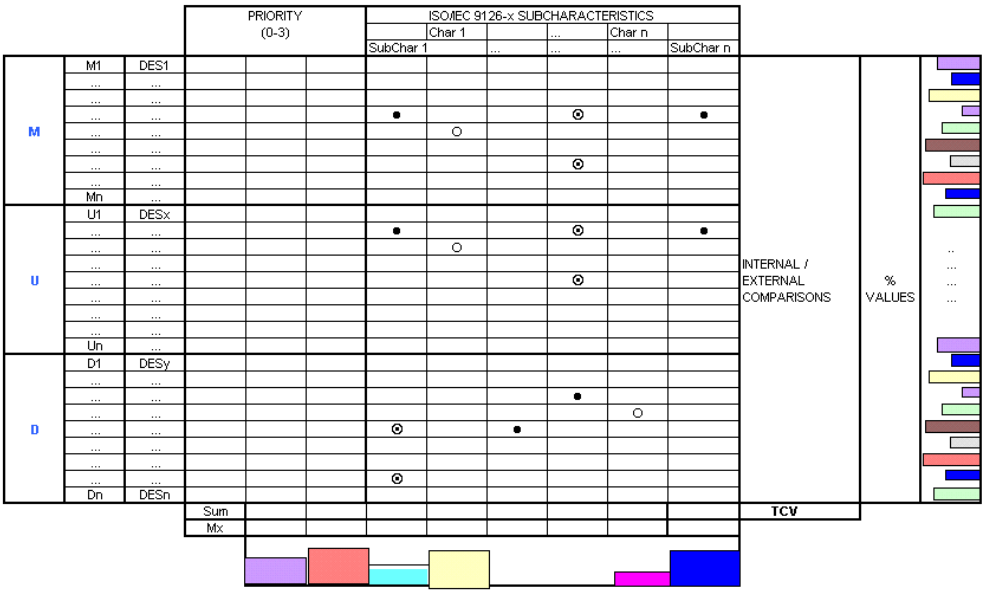


Figure 5 Example of the application of the HoQ for the Development Matrix

4.2 Strengths

The main QF^{2}D strengths are:

- Quality assessment profile available throughout the software lifecycle;
- Multiple-viewpoint assessment (M, U, D) of software quality with possible extension to more than three viewpoints;

- Good tailorability in terms of number of requirements (SRS/FUR[3]) and groups of stakeholders;

- Use of the updated version of ISO/IEC 9126 (ISO/IEC, 2000) (with 7 quality characteristics and 31 sub-characteristics)[4];

- Can be used to focus on a particular perspective and a much more in-depth analysis when required. For example, for an in-depth assessment of usability, another standard, such as ISO 9241-11, would be used, considering every characteristic as a separate dimension;

- Can be used with multidimensional performance models (Buglione, 2001-b).

4.3 Limitations

QF^{2}D was designed initially in the context of qualitative assessments based on expert judgment, both for targets of requirements and for achievements. Its implementation did not depend on the implementation of measurement programs. While QF^{2}D uses the key concepts of ISO 9126, it does not use the detailed measures proposed in the new version of ISO 9126, which had not been published at that time.

5 Integrated Graphical Assessment of Quality

How to integrate the graphical framework and the QF^{2}D technique

The strengths of the graphical framework and the QF^{2}D technique can be combined to integrate multiple viewpoints taken into account when assessing software quality. We refer to this as IGQ (Integrated Graphical Quality). For simplicity's sake, the integration of the two original methods into IGQ are presented here in a high-level view, referring to a single application phase, even though the approach can be easily expanded to the desired number of SLC phases.

Four main tasks to be performed:

- Selection of the quality requirements: selection of their most relevant systems and software functional requirements, from several stakeholder viewpoints;

- Identification and selection of quality characteristics and sub-characteristics, and of their priority in the implementation of requirements;

- Identification of the measurable attributes to use for monitoring the software project until its delivery to the customer;

- Calculation of the related numerical value for quality.

[3] Software Requirement Specification / Functional User Requirement.
[4] Considering ISO/IEC 9126:2002, parts 2 and 3, and also part 4 on “Quality in Use”.

Neither GDQA nor QF^{2}D can accomplish all these tasks, and they cannot do so from all the possible viewpoints stressed, as illustrated in Table 4.

Table 4. Handling of multiple viewpoints in QF^{2}D and GDQA, in each task

Task Viewpoint	1. Quality Requirements Selection	2. Quality (sub) Characteristics Selection	3. Measurable Attributes Selection	4. Quality Value Calculation
Managers	QF^{2}D	QF^{2}D	…	QF^{2}D
Users	QF^{2}D	QF^{2}D		QF^{2}D
Developers	QF^{2}D / GDQA	QF^{2}D / GDQA	GDQA	QF^{2}D

On the one hand, GDQA provides a means for building a structured and visual model of the quality requirements on the basis of measurements, and then integration of the various measures, from the bottom up, through functions. This is usually done from a single viewpoint; that is, from the Developer viewpoint.

On the other hand, QF^{2}D allows for viewpoints other than the Developer viewpoint (D) to be taken into account, and then for their integration into a single consolidated value for quality (with all the sub-levels for analysis). In addition, with QF^{2}D, the inputs to the quality models do not necessarily need to come from measurement programs; QF^{2}D allows the collection of *'opinions'* about the values of the sub-characteristics, and these opinions are collected using questionnaires (and the processing, in a quantitative manner, of the values collected in these questionnaires).

The integration of GDQA and QF^{2}D is illustrated in Figure 6 with an IDEF0 diagram (NIST, 1993).

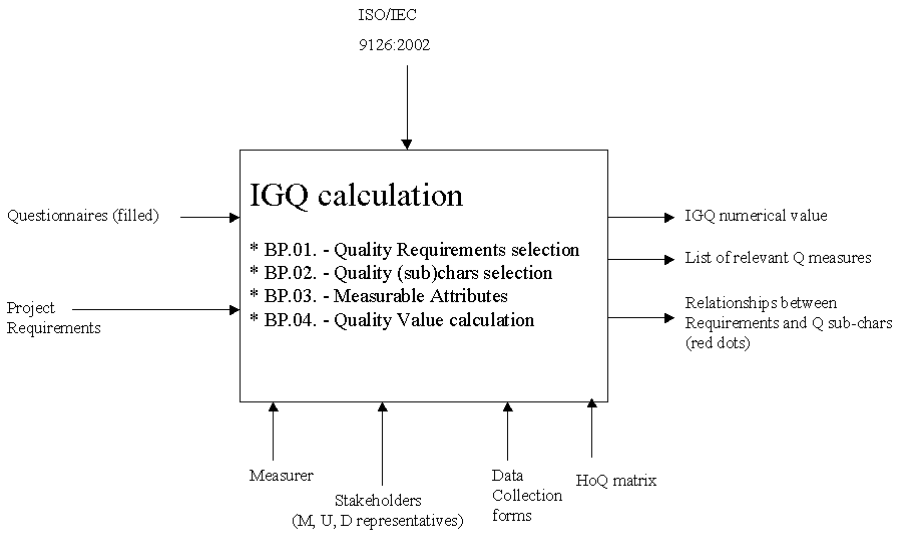


Figure 6. High-Level IDEF0 diagram for the "IGQ calculation" procedure

IDEF0 methodology takes into account four basic element types, the so-called ICOM (Input-Control-Output Mechanism), graphically represented with a series of arrows, into and out of the process box. The main *inputs* are the software requirement

specifications from the assessed project and the questionnaires completed by stakeholders on their quality priorities and targets. The *Mechanisms* used are the data collection forms, filled in by the stakeholders from the Manager, User and Developer viewpoints, and the HoQ matrix, managed by the quality analyst (e.g. the measurer). The *Control* is given by the standard chosen for software product quality, the new four-part ISO/IEC 9126:2002. Finally, the *Outputs* are the IGQ numerical values, expressed as percentages, the most relevant relationships between project requirements and the 9126 sub-characteristics, in order to derive the quality measures for monitoring the implementation of the project.

The repeatability of these main tasks in each SLC phase makes it possible to manage the quality of a software project in a practical way, with a graphical analytical tool, starting from the ISO quality model and determining the more relevant measures from this model.

6 Conclusions & Next Steps

Assessing software product quality has become more and more relevant and important to managers, even though it is still challenging to define and measure the detailed quality criteria and to integrate them into quality models. One of the main challenges is to define and use them taking into account multiple stakeholders at the same time, balancing different – and sometime opposing – requirements and trying to maximize the organizational target goals, always keeping in mind the "big picture".

Software engineering standards can help encourage the use of a common language for the detailed criteria and, in parallel, implement a model of quality from its high-level concepts down to its lowest level of measurable details; in particular, the ISO/IEC 9126:2002 suite of standards represents a useful model and taxonomy for specifying and measuring software product quality. Several tools and techniques are being built on the basis of this standard. In particular, the GDQA framework and the QF^2D technique have been proposed to tackle software product quality analysis and measurement. Their main strengths are – respectively – the dynamic analysis of quality through quality measures from ISO 9126 and the management of quality using a QFD-like structure.

This paper has examined the structure of both and proposed a way in which to integrate them into what we call an IGQ (an integrated graphical assessment of quality), which supports quality assessments and related improvements throughout the full software lifecycle. The main improvements are given by the utilization of the quality measures of the quality sub-characteristics chosen in each HoQ matrix for controlling the development for continuous improvement. IGQ, as QF^2D, can be used alone or in combination with other quantitative frameworks for software performance assessment, such as QEST/LIME. IGQ can be easily implemented throughout the software lifecycle, using the data analysis from the development phase as a main input for the maintenance phase, and again, data analysis from the latter as a main input for the next iterative development phase.

References

Abran A., Desharnais J.M., Oligny S., St-Pierre D. & Symons C. (2001) COSMIC-FFP Measurement Manual version 2.1, May 2001, www.lrgl.uqam.ca/cosmic-ffp

Boehm B.W., Brown J.R., Lipow H., MacLeod G.J. & Merrit M.J. (1978). Characteristics of Software Quality, Elsevier North-Holland.

Buglione L. & Abran A. (1999). A Quality Factor for Software, Qualita99 (3rd International Congress on Quality and Reliability), Paris, France, March 25-26, 1999,ISBN 2-900-781-43-4,335-344

Buglione L. & Abran A. (2000). QF^2D: a different way to measure Software Quality, included in: New Approaches in Software Measurement, 10th International Workshop on Software Measurement, IWSM 2000, Berlin, Germany, October 4-6, 2000. Proceedings, Dumke R. & Abran A. (Eds.), ISBN 3-540-41727-3.

Buglione L. & Abran A. (2001-a). QF^2D: Quality Factor through QFD application, Qualita2001 (4th International Congress on Quality and Reliability), Annecy, France, March 22-23, 2001, ISBN 2-9516453-0-0, 34-39.

Buglione L. & Abran A. (2001-b). Multidimensionality in Software Performance Measurement: the QEST/LIME models, SSGRR 2001 (2nd International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet), L'Aquila, Italy.

Dromey R.G. (1995), A Model for Software Product Quality, IEEE Transactions on Software Engineering, Vol. 21, No. 2, 146-162.

IEEE, Std 1061-1992 (1992). Standard for a Software Quality Metrics Methodology.

ISO/IEC, International Standard 9126 (1991). Information Technology - Software product evaluation – Quality characteristics and guidelines for their use, Geneva.

ISO/IEC-9126 (2000). Software Engineering- Software product quality, -Part 1: Software quality model Geneva.

ISO/IEC 9126 (2002) Software Engineering- Software product quality, , Part 2: External metrics Part 3: Internal metrics, Part 4: Quality In use, (to be published), Geneva.

ISO/IEC 19761 (2002) Software Engineering – COSMIC-FFP: A Functional Size Measurement Method (to be published), Geneva.

Kececi N. & Modarres M. (1998). Software Development Life Cycle Model to Ensure Software Quality, International Conference on Probabilistic Safety Assessment and Management, New York City, NY.

Kececi N., Li M. & Smidts C. (1999-a). Function Point Analysis: An Application to a Nuclear Reactor Protection System, International Topical Meeting on Probabilistic Safety Assessment –PSA'99, Washington, DC.

Kececi N., Modarres M., and & Smidts C. (1999-b). System Software Interface for Safety-Related Digital I&C Systems, European Safety and Reliability – ESREL'99 Conference, TUM Munich- Garching.

Kececi N. & Abran A. (2001), Analysing, Measuring & Assessing Software Quality in a Logic Based Graphical Model, Qualita2001 (4th International Congress on Quality and Reliability), Annecy, France, March 22-23, 2001, ISBN 2-9516453-0-0.

McCall J.A., Richards P.K. & Walters G.F. (1977). Factors in Software Quality, Voll. I, II, III: Final Tech. Report, RADC-TR-77-369, Rome Air Development Center, Air Force System Command, Griffiss Air Force Base, NY.

Modarres M., Y-S. Hu. (1999). Applying Fuzzy-Logic-Based Hierarchy for Modeling Behaviors of Complex Dynamic Systems. System & Software Computing in Nuclear Engineering. Da Ruan ed., Springer-Verlag.

NIST, Integration Definition for Function Modeling (IDEF0), Draft Federal Information Processing Standard Publication 183, Dec 21 1993.