ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
PH.D.

BY
AL QUTAISH, Rafa

SPQ^{MM}: A SOFTWARE PRODUCT QUALITY MATURITY MODEL USING ISO/IEEE STANDARDS, METROLOGY, AND SIGMA CONCEPTS

MONTRÉAL, JUNE 21, 2007

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain Abran, PhD, Thesis Supervisor
Department of Software Engineering and IT at the École de Technologie Supérieure

Mr. Witold Suryn, PhD, President of the Board of Examiners
Department of Software Engineering and IT at the École de Technologie Supérieure

Mr. François Coallier, PhD, Examiner
Department of Software Engineering and IT at the École de Technologie Supérieure

Mr. Luigi Buglione, PhD, External Examiner
Atos Origin Company, Rome, Italy

Mr. Houari A. Sahraoui, PhD, External Examiner
Department of Computer Science and Operational Research at the University of Montréal

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

JUNE 12, 2007

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I would like to seize this opportunity to convey my sincere thanks and deepest gratitude to my great supervisor Prof. Dr. Alain Abran for his support, guidance, encouragement, and assistance at so many levels throughout my study at the École de Technologie Supérieure. Also, I would like to thank everyone in the Software Engineering Research Laboratory (GÉLOG) and the Department of Software Engineering and IT.

I am deeply indebted to my committee members Prof. Dr. Witold Suryn, Dr. François Coallier, Dr. Luigi Buglione and Dr. Houari A. Sahraoui for their time and effort in reviewing this work.

Where would I be without my family? My parents deserve special mention for their inseparable support. My Father, in the first place is the person who put the foundations of my learning character, showing me the joy of intellectual pursuit ever since I was a child. My Mother, is the one who sincerely raised me with her caring and gently love. Khaled, Jamil, Haifa, Mohammad, Ali, Ahmad, Wafa, and Abdullah, thanks for being supportive and caring as siblings.

Words fail me to express my appreciation to my wife Lana and my small daughters Leen, Tala, and Sara for their moral support, encouragement, and caring during the writing of my thesis and during my study in the ÉTS.

Finally, I would like to thank everybody who was important to the successful realization of this thesis, as well as expressing my apologies that I could not mention personally one by one.

SPQ^{MM}: A SOFTWARE PRODUCT QUALITY MATURITY MODEL USING ISO/IEEE STANDARDS, METROLOGY, AND SIGMA CONCEPTS

AL QUTAISH, Rafa

ABSTRACT

In the software engineering literature, there are numerous maturity models for assessing and evaluating a set of software processes. By contrast, there is no corresponding maturity model for assessing the quality of a software product. The design of such a model to assess the quality of a software product therefore represents a new research challenge in software engineering.

Our main goal is to make available to industry (and consumers) a maturity model for assessing and improving the quality of the software product. This Software Product Quality Maturity Model (SPQ^{MM}) consists of three quality maturity submodels (viewpoints) that can be used not only once the software product has been delivered, but also throughout the life-cycle:

- Software Product Internal Quality Maturity Model – SPIQ^{MM},
- Software Product External Quality Maturity Model – SPEQ^{MM}, and
- Software Product Quality-in-Use Maturity Model – SPQiU^{MM}.

In this thesis, we introduce the Software Product Quality Maturity Model (SPQ^{MM}), which could be used from three different viewpoints: the software product internal quality, the software product external quality, and the software product quality in-use. This quality maturity model is a quantitative model, and it based on the ISO 9126 (software product quality measures), ISO 15026 (software integrity levels), IEEE Std. 1012 (software verification and validation) and on six-sigma concepts.

To build such a quality maturity model, we have combined the set of quality measures into a single number for each quality characteristic by assuming that all the measures for a single quality characteristic have an equal weight in the computation of a single value for that quality characteristic (they all make an equal contribution), yielding a quality level for that quality characteristic. The resulting quality level is then transformed based on the software integrity level into a sigma value positioned within a quality maturity level.

Keywords: ISO Standards, IEEE Standards, Software Measurement, Software Quality, Software Integrity Levels, Six Sigma, Quality Maturity Model, Metrology, SQuaRE.

SPQ^{MM}: UN MODÈLE DE LA MATURITÉ DE LA QUALITÉ DU PRODUIT LOGICIEL UTILISANT STANDARDS ISO/IEEE, MÉTROLOGIE, ET CONCEPTS SIGMA

AL QUTAISH, Rafa

RÉSUMÉ

Le logiciel est maintenant souvent essential pour donner un avantage concurrentiel à beaucoup d'organisations, et il devient progressivement un composant clé des systèmes d'entreprise, de leurs produits et de leurs services. La qualité de produits logiciels est donc maintenant souvent considérée un élément essentiel pour un succès commercial. L'industrie du logiciel a depuis plusieurs années consacré des efforts considérables pour essayer d'améliorer la qualité de leurs produits et le focus principal a porté jusqu'à maintenant sur l'amélioration du processus logiciel comme une approche indirecte pour atteindre cette qualité.

Dans la littérature du génie logiciel, il y a maintenant des nombreux modèles de maturité pour évaluer un ensemble de processus logiciels : par exemple, le CMMi, le modèle de la maturité de la maintenance de logiciel (S^{3M}) et le modèle de la maturité de la test (TMM). Par contraste, il n'y avait aucun modèle de maturité pour évaluer la qualité du produit logiciel. Par conséquent, la conception d'un tel modèle d'évaluation de la qualité d'un produit logiciel représentait un nouveau défi pour la recherche en génie logiciel.

Dans la littérature, il y a beaucoup de modèles de qualité et des centaines de mesures pour le produit logiciel. Sélectionner lesquels utiliser pour évaluer la qualité du produit logiciel est un défi. L'ISO a donc développé un consensus sur un modèle de la qualité et un inventaire de mesures pour évaluer la qualité d'un produit logiciel (ISO 9126). Cependant, même l'utilisation d'ISO 9126 et de ses mesures pour évaluer la qualité d'un produit logiciel représente un défi qui produit un ensemble de valeurs qui reflètent le niveau de la qualité de chaque mesure pour chaque caractéristique de la qualité. De plus, il est difficile d'interpréter ces nombres et de les intégrer dans un modèle de prise de décisions.

Le but principal dans cette thèse est rendre disponible à l'industrie (et éventuellement aux consommateurs) un modèle de la maturité pour évaluer et améliorer la qualité du produit logiciel en un utilisant un modèle de niveaux de la maturité de la qualité. Le modèle conçu et présenté dans cette thèse comprend une structure du modèle de la maturité de la qualité qui tient compte de trois points de vue distincts:

- Le produit logiciel entier,
- L'étape du cycle de la vie du produit logiciel (interne, externe et en-service), et
- La qualité du produit logiciel.

Du point de vue de l'étape du cycle de la vie du produit logiciel, cela consiste en trois sous-modèles de la maturité de la qualité qui peuvent être utilisés non seulement lorsque le produit logiciel a été délivré, mais aussi partout dans le vie cycle:

- Modèle de la maturité de la qualité interne du produit logiciel – $SPIQ^{MM}$,
- Modèle de la maturité de la qualité externe du produit logiciel – $SPEQ^{MM}$, et
- Modèle de la maturité de la qualité en service du produit logiciel – $SPQiU^{MM}$.

Une difficulté principale dans la conception d'un modèle de la qualité c'est la diversité des modèles décrits dans la littérature, la variété des vues sur qualité et les techniques détaillées : par exemple, le modèle de la qualité de McCall, le modèle de la qualité de Boehm, le modèle de la qualité de Dromey et le modèle de qualité FURPS. De plus, des centaines de mesures du logiciel -ou ce qui est appelé communément "métrique logiciel" - ont été proposées pour évaluer la qualité du logiciel, mais, malheureusement, sans consensus international large sur l'usage et les interprétations de ces modèles de la qualité et des mesures du logiciel.

Le développement d'un consensus dans les organisations de normalisation du génie logiciel (ISO et IEEE) a mené à quelques consensus sur le contenu du modèle de la qualité contenu ainsi que sur inventaires de mesures correspondantes.

Notre stratégie a été de construire un modèle de la maturité qui ne soit pas basé pas sur nos propres vue de la qualité du logiciel, mais sur le nouveau consensus dans la communauté dans les normes en génie logiciel. Par conséquent, le modèle de la maturité proposé est basé sur les documents suivants de l'ISO et de l'IEEE:

- ISO 9126-Partie 1: Modèle de la qualité,
- ISO 9126-Partie 2: Mesures internes,
- ISO 9126-Partie 3: Mesures externes,
- ISO 9126-Partie 4: Qualité en-service Mesure,
- ISO 15026: Système et Niveaux de l'Intégrité du Logiciel,
- Std IEEE. 1012: Standard pour Vérification du Logiciel et Validation,
- ISO VIM: Vocabulaire International des termes fondamentaux et généraux de Métrologie, et
- ISO 15939: Processus de la Mesure du logiciel.

En plus des sources précitées, les concepts six-sigma sont utilisés pour aligner la projection topographique entre le niveau de la qualité nivelle et niveau de la maturité de la qualité du produit logiciel.

La structure du Modèle de la Maturité de la Qualité du Produit logiciel (SPQ^{MM}) proposé dans cette thèse est basée sur deux ensembles de concepts qui existent dans industrie dans général qui est:

- Une approche quantitative à qualité du produit, et
- Les niveaux de qualité d'un produit.

Pour une approche quantitative à la qualité du produit, l'approche six-sigma a été sélectionnée pour construire le modèle de la maturité de la qualité présenté dans ce chapitre.

Pour les niveaux de qualité d'un produit, les cinq niveaux suivants de la maturité de la qualité ont été identifiés de l'observation d'industrie générale pratique à l'extérieur du domaine du logiciel:

- Garanti.
- Certifié.
- Neutre.
- Insatisfait.
- Complètement insatisfait.

Notre Modèle de la Maturité de la Qualité du Produit logiciel peut être utilisé pour déterminer la maturité de la qualité d'un produit logiciel. Spécifiquement, il peut être utilisé pour:

- Certifier un niveau de la maturité de la qualité pour un nouveau produit logiciel qui pourrait aider à l'encourager sur le marché;
- Repérer les produits logiciels existants aider de faire une sélection basée sur leur niveau de la maturité de la qualité;
- Répartir la qualité du produit logiciel pendant le cycle de la vie du développement (c.-à-d. intérieurement, extérieurement et en-service) enquêter sur les rapports entre les trois étapes et trouver toutes faiblesses pour améliorer le produit logiciel;
- Répartir la maturité de la qualité interne d'un produit logiciel qui peut être réutilisé dans les autres produits logiciels.
- Comparer les niveaux de la maturité de la vie cycle étape qualité (c.-à-d. interne, externe et en-service).

Le modèle de la maturité de la qualité du produit logiciel discuté dans cette recherche est actuellement limité au modèle de la qualité d'ISO 9126 (et SQuaRE dans la prochaine édition SIO) ainsi qu'à leurs ensembles des mesures que nous avons analysé de la perspective de la métrologie et les améliorations proposées au groupe ISO pour appuyer ces documents. Une autre limitation actuelle du modèle présenté est l'attribution de poids égaux pour toutes les mesures, toutes les caractéristiques et toutes les sous-caractéristiques. Pour lever cette contrainte, une organisation pourrait utiliser par exemple la technique statistique de l'analyse par composantes principales (PCA) pour déterminer un poids correspondant pour chaque mesure, caractéristique ou sous-caractéristique.

Mots-clés : Standards ISO, Standards IEEE, Mesure du Logiciel, Qualité du Logiciel, Niveaux de l'Intégrité du Logiciel, Modèle de la Maturité de la Qualité, Métrologie SQuaRE.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LISTE OF ABBREVIATIONS

ANSI	American National Standards Institute
BIPM	Bureau International des Poids et Mesures
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CMMI-SW	Capability Maturity Model Integration for Software engineering
COTS	Commercial Off-The-Shelf
CQL	Characteristic Quality Level
DFSS	Design For Six Sigma
DMAIC	Design, Measure, Analyse, Improve, and Control
DPMO	Defects Per Million Opportunities
EQL	External Quality level
ESA	European Space Agency
ESLOC	Executable Source Lines of Code
GDE	Global software Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFreq	Indicative Frequency
IQL	Internal Quality Level
ISO	International Organization for Standardization
IT	Information Technology
IUPAC	International Union of Pure and Applied Chemistry

IUPAP	International Union of Pure and Applied Physics
LOC	Line of Code
ML	Maturity Level
NORMSINV	Inverse Standard Normal Cumulative Distribution
OIML	Organisation Internationale de Métrologie Légale
OOP	Object-Oriented Programming
OSMM	Open Source Maturity Model
OSV	Original Sigma Value
PCA	Principal Component Analysis
QL	Quality Level
QM	Quality Measure
R6S	Rytheon Six Sigma
σ	Sigma
S^{3M}	Software Maintenance Maturity Model
SEI	Software Engineering Institute
$SPDQ^{MM}$	Software Product Documentation Quality Maturity Model
$SPEE^{MM}$	Software Product External Efficiency Maturity Model
$SPEF^{MM}$	Software Product External Functionality Maturity Model
$SPEiU^{MM}$	Software Product Effectiveness in-Use Maturity Model
$SPEQ^{MM}$	Software Product External Quality Maturity Model
$SPEM^{MM}$	Software Product External Maintainability Maturity Model
$SPEP^{MM}$	Software Product External Portability Maturity Model

$SPER^{MM}$	Software Product External Reliability Maturity Model
$SPEU^{MM}$	Software Product External Usability Maturity Model
$SPIE^{MM}$	Software Product Internal Efficiency Maturity Model
$SPIF^{MM}$	Software Product Internal Functionality Maturity Model
$SPIQ^{MM}$	Software Product Internal Quality Maturity Model
$SPIM^{MM}$	Software Product Internal Maintainability Maturity Model
$SPIntegQ^{MM}$	Software Product Integration Quality Maturity Model
$SPIP^{MM}$	Software Product Internal Portability Maturity Model
$SPIR^{MM}$	Software Product Internal Reliability Maturity Model
$SPIU^{MM}$	Software Product Internal Usability Maturity Model
$SPPiU^{MM}$	Software Product Productivity in-Use Maturity Model
$SPQiU^{MM}$	Software Product Quality in-Use Maturity Model
SPQ^{MM}	Software Product Quality Maturity Model
$SPSiU^{MM}$	Software Product Satisfaction in-Use Maturity Model
$SPSQ^{MM}$	Software Product Supporting Quality Maturity Model
$SPTQ^{MM}$	Software Product Training Quality Maturity Model
SC7	Subcommittee Seven
SQL	Subcharacteristic Quality Level
SQuaRE	Software Quality Requirements and Evaluation
SSV	Sigma Shift Value
TSLOC	Total Source Lines of Code
TMM	Testing Maturity Model

VIM	Vocabulaire International des termes fondamentaux et généraux de Métrologie – International Vocabulary of Basic and General Terms in Metrology
WG6	Working Group Six
WQL	Whole software product Quality Level

INTRODUCTION

Software is critical in providing a competitive edge to a great number of organizations and is progressively becoming a key component in business systems, products and services. The quality of software products is now considered to be a critical business success factor (Veenendaal and McMullan, 1997). The software industry is putting considerable effort in trying to improve the quality of their products; its main focus has been on software process improvement as an indirect approach to achieve software product quality (Veenendaal and McMullan, 1997).

Problem Statement

For the software product, in literature, there are hundreds of software measures – or what is commonly called 'software metrics' – proposed to assess and evaluate its quality, but unfortunately, without wide international consensus on their use and their interpretations. Furthermore, some of those measures do not meet key design criteria of measures in engineering and the physical sciences (Abran, Lopez and Habra, 2004). However, from 2001 to 2004, ISO came up with a set of documents referred to as the ISO 9126 series which consists of four parts: that is, one international standard and three technical reports. The first part of ISO 9126 (ISO, 2001b) describes in some details a software product quality model which should be used with the other three parts. The other three parts (ISO, 2003c; 2003d; 2004f) can be used to measure and evaluate the internal quality, external quality, and quality in-use attributes of a software product.

The ISO has recognized a need for further enhancement of ISO 9126, primarily as a result of advances in the information technologies (IT field) and changes in the IT environment. Consequently, the ISO is now working on the next generation of software product quality standards, which will be referred to as Software Product Quality

Requirements and Evaluation (SQuaRE – ISO 25000). Once completed, this series of standards will replace the current ISO 9126 and ISO 14598 series of standards.

One of the main objectives of the SQuaRE series and the current ISO 9126 series (which also constitutes the difference between them) is the coordination and harmonization of its contents with the ISO standard on software measurement process – ISO 15939 – which is based itself on the ISO International Vocabulary of Basic and General Terms in Metrology (VIM).

Within the ISO 9126, there are six characteristics and twenty seven subcharacteristics for the internal and external software product, and four characteristics for the in-use software product. For each of these subcharacteristics (and characteristics in case of quality in-use), there is a number of measures which could be used to assess and evaluate the software product from different points of view (i.e. internal functionality, external efficiency, in-use productivity, etc.). Thus, each characteristic/subcharacteristic is represented by a set of numbers.

In contrast, sometimes decision makers do not want to work with a set of numbers which reflect a specific quality characteristic or subcharacteristic of a particular software product. Instead, they prefer to deal with a single number to identify the quality level of such quality characteristic of that software product, and then to make an appropriate decision based on this single number. Such technique is already used in the evaluation of some of the software processes using capability and/or maturity models.

In the literature, there are many capability and maturity models to assess and evaluate a specific process to produce an equivalent maturity level (a single value to reflect the maturity of that process); generally speaking, industry and researchers use a five-scale level for their maturity models. Examples of such maturity levels are:

- Capability and Maturity Model – CMM (SEI, 2002a; 2002b).

- Software Maintenance Maturity Model – S^{3M} (April, Abran and Dumke, 2004; April et al., 2005).
- Testing Maturity Model – TMM (Burnstein, Suwanassart and Carlson, 1996a; 1996b).

However, these assessment models are about 'process' while we are interested in developing 'product quality' assessment model. The design of a software product maturity model to assess the quality of a software product, therefore, represents a new research challenge in software engineering.

Goals of the Research

The first goal of our research lies in the building of an understanding of the designs and definition of the current proposed measures for software product quality to determine their strengths and weaknesses. In particular, we need to verify the ISO 9126 measures against the metrology concepts, and to build an ISO-based information model to address the harmonization issues in the ISO 25020 and ISO 25021 new standards.

While the second goal of this research project aims at building a maturity model based on software engineering standards rather than individual models of software product quality, in particular, it will be based on:

1. Measures based on sound metrological foundations.
2. Industry consensus on base measures for software product quality.
3. Industry consensus on software product quality models.
4. Industry consensus on software integrity levels.

The research objectives of the research described in this thesis are the verification of the ISO 9126 software product quality measures against the metrology concepts and the

building of a software product quality maturity model to assess the quality of the software product. These objectives are presented in more details in Chapter 5.

Such a maturity model has to assess the quality of the software product in order to produce quality maturity levels form each of the following three different views:

1. The ISO 9126 software product quality characteristics view.
2. Life-cycle stages (internal, external and in-use) view.
3. The whole software product view.

To get into the above research objectives, we have a two-phase research methodology. The first phase consists of three steps, that is, verifying the usability of the metrology concepts in the software measurement, verifying the ISO 9126 measures against the metrology concepts, and building an ISO-based information model to address harmonization issues in the ISO 25020 and ISO 25021 standards. While, the second phase consists of ten steps which are used to build the software product maturity model. Chapter 5 describes in more details this research methodology.

Contributions of the Research

The research contributions of this PhD thesis are:

- Verification of the applicability of using metrology concepts to software measures to investigate their design and definition. For instance, we verified the ISO 9126 measures against the metrology concepts. We noted that the metrology concepts could be applied to the available software measures to verify their consensus with the classical measurement in other engineering disciplines.
- Identification of some of the harmonization issues arising with the addition of new documents like the ISO 25020 and ISO 25021, in particular, with respect to previously published measurement standards for software engineering, such as ISO 9126, ISO 15939, ISO 14143-1 and ISO 19761.

- Identification of a list of base measures needed to evaluate the ISO 9126 part-2, part-3, and part-4 derived measures. For each of these base measures, the measurement unit has been identified. In addition, a cross-reference list between the base measure and the related characteristics/subcharacteristics has been built.
- Applying the sigma concepts to the measured quality levels of the software product, by mapping the quality level to the corresponding sigma value; this sigma value can then be used to derive the maturity level.
- Building of a maturity model to assess the maturity level of the software product quality from different views (the characteristic view, the life cycle stage view, and the whole software product view) based on a set of ISO and IEEE standards. For this quantitative maturity model, the sigma concepts have been incorporated into the software integrity levels to obtain the appropriate values of the quality maturity levels of different types of software products (real-time software, application software, etc.)

A number of outcomes of this thesis have been published/accepted/submitted in the following journals, books or conferences.

- Published:
    1. Al-Qutaish, Rafa E. and Alain Abran. 2005. "An Analysis of the Design and Definitions of Halstead's Metrics", 15th International Workshop on Software Measurement - IWSM'2005. (Montreal (Que), Canada, 12-14 Sept. 2005), p. 337-352. Aachen, Germany: Shaker Verlag.
    2. Abran, Alain; Rafa E. Al-Qutaish and J. M. Desharnais. 2005. "Harmonization Issues in the Updating of the ISO Standards on Software Product Quality", Metrics News Journal, vol. 10, no 2, p. 35-44.
    3. Abran, Alain; Rafa E. Al-Qutaish; Jean-Marc Desharnais and Naji Habra. 2005. "An Information Model for Software Quality Measurement with ISO Standards", International Conference on Software Development - SWDC-

REK. (Reykjavik, Iceland, 27 May - 1 Jun. 2005), p. 104-116. Reykjavik, Iceland: University of Iceland Press.

4. Abran, Alain; Rafa E. Al-Qutaish and Juan J. Cuadrado-Gallego. 2006. "Investigation of the Metrology Concepts within ISO 9126 on Software Product Quality Evaluation", 10th WSEAS International Conference on Computers - ICComp'2006. (Vouliagmeni (Athens), Greece, 13-15 Jul. 2006), p. 864-872. Athens, Greece: WSEAS Press.
5. Abran, Alain; Rafa E. Al Qutaish and Juan J. Cuadrado-Gallego. 2006. "Analysis of the ISO 9126 on Software Product Quality Evaluation from the Metrology and ISO 15939 Perspectives", WSEAS Transactions on Computers, vol. 5, no 11, p. 2778-2786.

- Accepted:
  6. Abran, Alain; Al-Qutaish, Rafa. E.; Desharnais, Jean-Marc and Habra, Naji. 2007. "ISO-Based Models to Measure Software Product Quality", Accepted as a Chapter in "Software Quality Measurement" – a book to be Edited by G. Vijay, FCAI - the Institute of Chartered Financial Analysts of India, Hyderabad, India: FCAI University Press.
- Submitted:
  7. Al-Qutaish, Rafa E. and Abran, Alain. 2007. "Assessment of Software Product Quality: Determining the Maturity Levels", Submitted to the ASQ Software Quality Professional Journal, American Society of Quality, USA.

Organization of the Thesis

This thesis consists of twelve chapters (including the introduction and the conclusion) and five appendices. The current chapter outlines the problem, the research goals, and a summary of the main contributions of this thesis.

Chapter one gives an overview of the field of software measurement, different definitions of the software measures (or as some times called metrics) and software quality, and a brief explanation of the potential benefits of using the software measurements. In addition, it shows the different classifications of the software measures and describes the software product quality measurement. Finally, at the end of this chapter some examples of software product quality measures are presented.

Chapter two presents a general overview of the ISO related standards on software product quality measurement. Then, it shows what the standard is and gives a brief list of its benefits. In addition, it contains a general description of the contents of the ISO 9126, ISO 14598, ISO 25020, ISO 25021, ISO 25051, ISO 15939, and the ISO guide on the international vocabulary of basic and general terms in metrology (VIM). Finally, it identifies the strengths and weaknesses of those ISO standards.

Chapter three presents a classification of the maturity models and describes a number of the software engineering maturity models. In particular, it discusses three of the maturity models used to assess the quality of the software process and introduces the only two maturity models which we have found in the literature for the software product. Finally, it illustrates strengths and weakness of the discussed product maturity models.

Chapter four gives a brief overview of the concepts to be used to build the proposed quality maturity models. Particularly, it explains the contents of five quality models from the software engineering literature, sigma concepts, and software integrity levels standards.

Chapter five presents the research objectives and research methodology. The research methodology is divided into four phases. In this chapter, the details of each phase are presented.

Chapter six firstly presents a brief overview of the Halstead's measures (as a case study for other measures in ISO 9126) and the analysis framework which we will use to analyze these measures. In addition, it presents the details of our analysis of the design and definitions of Halstead's measures. Finally, at the end of this chapter, a discussion on this analysis and a summary of our observations are described.

Chapter seven presents the analysis of the ISO 9126-4 on the quality in-use measures from the metrology and ISO 15939 perspectives. Specifically, it presents the analysis of the effectiveness, productivity, safety and satisfaction measures, respectively. Finally, a set of comments and suggestions for a potential improvement of the ISO 9126-4 are presented.

Chapter eight presents some of the harmonization issues arising from introducing the ISO 25020 and ISO 25021 new standards with respect to previously published measurement standards for software engineering, and proposes ways to address them using the measurement information model of ISO 15939 on software measurement process.

Chapter nine shows the structure of our proposed software product quality maturity model. More specifically, it presents the used quality maturity levels and the quantitative approach to the product quality. In addition, it describes the contents of the different views of the software product quality maturity model.

Chapter ten illustrates the different steps which should be followed to get the quality maturity level from any of the different views.

Finally, this thesis contains five appendices. The first two, Annex I and Annex II describe the quantities and units metrology concepts in the quality in-use safety measures and satisfaction measures, respectively. The third one, Annex III provides a

comparison between ISO 9126 and ISO 25021 with respect to the availability of the characteristics, subcharacteristic and measures in both. The fourth one, Annex IV gives a list of the base measures from ISO 9126 parts 2, 3 and 4. The last one, Annex V presents a cross-reference table between the base measures and the characteristic / subcharacteristic in which they could be used.

CHAPTER 1

SOFTWATRE MEASUREMENT: AN OVERVIEW

1.1 Introduction

Measurements have a long tradition in natural sciences. At the end of the 19th century, the physicist, Lord Kelvin, formulated the following about measurement:

> "When you can measure what you are speaking about, and express it into numbers, you know some thing about it. But when you can not measure it, when you can not express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to stage of science." (Pressman, 2004, p. 79)

In addition, Roberts (1979, p. 1) points out in his book about measurement theory that: "A major difference between 'well-developed' sciences such as physics and some of the less 'well-developed' sciences such as psychology or sociology is the degree to which things are measured."

In the area of software engineering, the concept of software measurement (or what is commonly called software 'metrics') is not new. Since 1972, a number of so-called software 'metrics', or measures, have been developed. From the wide range of software measures, four basic theories have been the source of the majority of the research conducted on software measurement. Some of these measures have been defined by McCabe (1976), Halstead (1972; 1977), Albrecht (1979), and DeMarco (1986).

The importance of the costs involved in software development and maintenance increases the need of a scientific foundation to support programming principles and

management decisions by measurement. Already in 1980, Curtis (1980, p. 1144) pointed out that:

> "Rigorous scientific procedure must be applied to studying the development of software systems if we are to transform programming into an engineering discipline. At the core of these procedures is the development of measurement techniques and the determination of cause-effect relationships."

In the software engineering literature, some researches use the term 'metrics' while others use the term 'measures' to refer to the same concept.

The definition of a 'measure' is "an empirical objective assignment of a number or a symbol to an entity to characterize a specific attribute" (Fenton and Pfleeger, 1997, p. 28). Moreover, Ince *et al.* (1993, p. 22) have defined the software 'metrics' as "numerical values of quality which can be used to characterize how good or bad that the product is in terms of properties such as its proneness to error". In addition, 'metrics' is defined in (IEEE, 1990, p. 47) as "quantitative measures of the degree to which a system, component or process possesses a given attribute", while within the ISO 15939, the term 'measure' was defined as a "variable to which a value is assigned as a result of measurements" (ISO, 2002, p. 3). Finally, it has been defined in ISO 14598-1 as "the number or category assigned to an attribute of an entity by making a measurement" (ISO, 1999a, p. 3)

In 2002, ISO produced the ISO 15939 international standard (ISO, 2002), that contains the definitions of the terms to be used in the software measurement process, including the term 'measure' instead of the term 'metrics'. Therefore, in this thesis, we will use the term 'measure' – whenever it is possible – in order to be aligned with this international standard.

Essentially, the software measures are used to measure the quality of the software product or process. However, there are several definitions for the 'software Quality' expression. For example, it is defined by the IEEE (1990, p. 60) as "the degree to which a system, component or process meets specified requirements and customer (user) needs (expectations)". Pressman (2004, p. 199) defines it as "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software". By contrast, the ISO defines 'quality' in ISO 14598-1 (ISO, 1999a, p. 4) as "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs", and Petrasch (1999, p. 2) defines it as "the existence of characteristics of a product which can be assigned to requirements".

The rest of this chapter is organized as follows: Section 1.2 presents the general benefits of the use of the software measurement. Section 1.3 shows the different classification of the software measures, and particularly, the software product measures. Section 1.4 gives an overview of the software product measurement. Finally, section 1.5 illustrates some examples from the literature on software product measures from the design and coding perspectives.

1.2 Software measurement benefits

Software measurement helps in two ways. First, it helps individual developers understand what they are doing and provides insight into areas that they might improve. For example, the measurement of code complexity gives information about which code is overly complex and might be improved by additional modularisation, and the measurement of numbers and types of bugs gives information on what errors a developer is prone to make and thus what he should be watching out for. Second, software measurement gives an organisation information about where it is, and about the effect of things it is trying to use (Sharp, 1993).

Grady and Caswell (1987) have summarised the advantages of the software measurement. They determined that software measurement helps the developers to:

- Understanding the software development process better.
- Measuring progress.
- Providing common terminology for key controlling elements of the process.
- Identifying complex software elements.
- Making software management more objective and less subjective.
- Estimating and scheduling better.
- Evaluating the competitive position better.
- Understanding where automation is needed.
- Identifying engineering practices which lead to the highest quality and productivity.
- Making critical decisions earlier in the development process.
- Eliminating fundamental causes of defects.
- Encouraging the use of software engineering techniques.
- Encouraging the definition of long-term software development strategy based upon a measured understanding of current practices and needs.

1.3 Classifications of the software measurement

Fenton and Pfleeger (1997) have classified the software measures into three classes: product measures, process measures, and resource measures. In more details, they based their classification on the following definitions:

- Products: any outcomes, deliverable, or documents that are emerged from the processes.
- Processes: any activities which are related to the software itself.
- Resources: the items that are input to the processes.

Anything we are likely need to measure or predict in software is an attribute of some entity of the three classes (product, process or resource). Fenton and Pfleeger (1997) have made a distinction between these attributes, which are internal or external:

- Internal attributes of a product, process, or resource are those that can be measured totally in terms of the product, process, or resource itself.
- External attributes of a product, process, or resource are those which can only be measured with respect to how the product, process, or resource relates to its environment, that is, during the execution.

In addition, the measures could be classified to be either direct or indirect measures based on their dependability on other measures or not (Fenton and Pfleeger, 1997):

- Direct measures: are the measures which could be calculated based on the attribute itself and without using any other measures.
- Indirect Measures: are the measures which could be evaluated based only on other measures.

1.4 Software product measurement

The 'product' measures are one of the three classes of the software measures as classified by Fenton and Pfleeger (1997). The other two classes are 'process' measures and 'resource' measures. The product measures are the measures related to the software product itself (Conte, Dunsmore and Shen, 1986), and are divided into the following three types:

1. Specification measures.
2. Design measures.
3. Source-code measures.

On the other hand, Kafura and Canning (1985) have divided the product measures into three types:

1. Code measures: are based on implementation details.
2. Design measures: are based on an analysis of the software's design structure.
3. Hybrid measures: are the combination of the above two measures.

For example, the design measures are the measures which address the design deliverables in the software lifecycle. Thus, these measures can give the developer early information about the software project at much earlier stage than source code measures. Most of the design measures are used for dividing a system into modules and the interrelation between these modules. Design measures can be derived either from the source code or from a system design documents. Examples of design measures are the information-flow measures (Kafura and Canning, 1985) and the call graph measures (Yin and Winchester, 1978).

Whereas, the source code measures are the measures which address the source code itself. These measures can be applied and collected during the coding (implementation) phase of the software lifecycle. Examples of source-code measures are Lines of Code – LOC – measures, McCabe's measure (McCabe, 1976), and Halstead's measures (Halstead, 1977).

Moreover, Dumke and Foltin (1998) have classified the software product measures into five types:

1. Size measures.
2. Architecture measures.
3. Structure measures.
4. Quality measures.
5. Complexity measures.

Finally, Andersson (1990) has classified the software product measures into the following five classes:

1. Productivity measures, such as the Lines of Code (LOC) and Halstead's measures.
2. Complexity measures, such as the McCabe's and Information Flow measures.
3. Reliability measures, such as the Mean Time between Failures (MTBF) and availability measures.
4. Readability measures, such as Fog-Index measure.
5. Error prediction measures.

1.5 Examples on software product measures

In this subsection, we provide some examples of the software product measures based on the classification introduced by Kafura and Canning (1985) in which they classified the software product measures into three classes, that is, design, code and hybrid measures.

1.5.1 Design measures

The design measures can give the developer information about the software product much earlier than the source code measures. Most of the design measures are used for dividing a system into modules and the interrelation between these modules. The design measures can be derived either from source code or from a system design documents. In this section, we discuss two of the design measures, that is, information-flow measures and call-graph measures.

1.5.1.1 Information-flow measures

The information-flow measures were first suggested by Henry and Kafura (1981). Their measures are derived from the following direct measures of specific information-flow attributes:

- Fan-in of a module A is the number of local flows that terminate at A, plus the number of data structures from which information is read by A.

- Fan-out of a module A is the number of local flows that emanated from A, plus the number of data structures that are updated by A.

Moreover, Henry and Kafura (1981) measures depend on the complexity of the module code and the complexity of the modules' connections to their environment. The complexity of a module code is simply a length of a module in which it is the number of lines in source code for the module. The value of the complexity of each module is defined as:

$$\boxed{Length \times (Fan\text{-}in \times Fan\text{-}out)^2} \tag{1.1}$$

Regarding the calculation of the information-flow measures (finding the fan-in and fan-out), the information flows need to be identified and then may be classified as a direct local flow, an indirect local flow, or a global flow. The definitions of these classes are as the following:

1. Direct local flow is defined to exist if a module invokes a second module and passes information to it, or the invoked module returns a result to the caller module. For example, module A calls module B and passes parameters to it.
2. Indirect local flow is defined to exist if the invoked module returns information, which is subsequently passed, to a second invoked module. For example, module C calls module A and module B and passes the return value from module A to module B.
3. Global flow is defined to exist if there is a flow of information from one module to another via a global data structure. For example, module A writes to a data structure and module B reads from the data structure.

Ince and Shepperd (1989), and Shepperd (1990) illustrate that there are a number of problems with these measures. The problems can be summarized as follow:

- Henry and Kafura do not formally describe the connection between two modules.

- The definition would seem only to cover indirect local flow over the scope of two levels, without giving a reason why indirect local flows with more than two levels should not covered.
- The indirect local flow between two called modules *B* and *C* can not be detected without having the internal functionality of the calling module *A*.
- A module that is used many times will receive a high value of complexity.
- The non-availability of the length of the module during the design phase.

The information-flow measures are difficult to be obtained, partly because a large-scale software requires a large amount of calculations, and because many of Henry and Kafura's definitions are not clear. To solve those difficulties, Ince and Shepperd (1989) make a number of assumptions, which are:

- Recursive module calls treated as normal calls.
- Any variable shared by two or more modules treated as a global data structure.
- Compiler and library modules ignored.
- Indirect local flow only counted across one hierarchical level.
- Indirect local flow ignored, unless the same variable is both imported and exported by the controlling module.
- No attempt made to make dynamic analysis of module calls.
- Duplicate flows ignored.

1.5.1.2 Call-graph measure

The call-graph measures are derived from the module call graph. The call graph is usually modeled by representing modules as nodes, and calls as edges in the graph. The graph impurity (Yin and Winchester, 1978) is one of the measures that can be derived from the module call-graph. This measure determines how the graph departs from a pure tree. If *e* is the total number of edges and *n* is the total number of nodes for a given call

graph, then the complexity of the connections within the design can be measured by the definition of the graph impurity, which is:

$$\boxed{c = e - n + 1} \tag{1.2}$$

Berg and Broek (1995) have classified the call-graphs. Hence, the following four classes of call graphs are distinguished:

- General call-graph: the customary graph with calls between the three types of modules (locals, globals, and library modules).
- Global call-graph: calls between top level modules and library modules (directly and indirectly via local modules).
- Local call-graph: for each top level module, the calls between this module and other top level modules, library modules, and local modules which are in scope of the top level modules in the root.
- Include call-graph: in this call graph, there are no module dependencies, but calls between scripts (via the include construct).

1.5.2 Source code measures

The source code measures are the measures which are concerned with the source code. These measures can be applied and collected during the coding (implementation) phase of the software life-cycle. In this section, we will discuss three various measures of this type, which are Source Lines of Code (SLOC) measures, McCabe's measure (McCabe, 1976), and Halstead's measures (Halstead, 1977).

1.5.2.1 Source lines of code measures

The most familiar software measure and the simplest is the count of the source lines of code (LOC). It is used for measuring the program size or length. The LOC measure can

be used as a predictor for productivity, quality, and effort. Unfortunately, there are many ways to measure the source lines of code for a particular piece of source code; one may count the total number of lines, including comments and blank lines, or one may ignore comments and blank lines, and one may or may not count declaration lines.

1.5.2.2 McCabe's measure

McCabe's measure or 'cyclomatic complexity' (McCabe, 1976) is a measure of the number of linearly independent paths through a program. The number of paths can be infinite if the program has a backward branch. Therefore, the cyclomatic measure is built on the number of basic paths through the program.

The cyclomatic complexity (*V(G)*) is derived from a control graph, which is modeled from a source code, and is mathematically computed using graph theory. In the control graph, edges represent the flow of control and nodes represent the statements. For a control graph G with e edges, n nodes, and p connected components, the cyclomatic complexity (*V(G)*) is calculated from:

$$\boxed{V(G) = e - n + 2p} \tag{1.3}$$

Simply speaking (as McCabe has observed), cyclomatic complexity is found by determining the number of decision statements in a program, and is calculated as the following (McCabe, 1976):

$$\boxed{V(G) = N + 1} \tag{1.4}$$

where N is the number of decisions statements in a program.

By counting the decision statements, the complexity of a program can be calculated. However, many decision statements contain compound conditions. An example is a compound '*IF*' statement, that is, *IF A=B AND C=D THEN.*

If the decision statements are counted in this example, cyclomatic complexity is equal to two (one *IF* statement plus one). If compound conditions are counted, the statement could be interpreted as: *IF A= B THEN IF C= D THEN.* Therefore, the cyclomatic complexity would be three (two *IF* statements plus one). Cyclomatic complexity makes the assumption that compound decision statements increase program complexity and integrates individual conditions in order to calculate *V(G).* An upper limit of ten for program complexity is proposed because grater complexity would be less manageable and testable (McCabe and Butler, 1989).

1.5.2.3 Halstead's measures

In Halstead's measures, commonly called 'software science' (Halstead, 1977), a computer program is considered to be a collection of tokens that can be classified as either operators or operands. A program can be thought of as a sequence of operators and their associated operands. All Halstead's measures are functions of the counts of these tokens.

The design and definition of the Halstead's measures are described and analysed in details through Chapter 6 of this thesis.

CHAPTER 2

SOFTWARE PRODUCT QUALITY MEASUREMENT IN ISO STANDARDS

2.1 Introduction

Many organizations are developing standards for software engineering; for example, the European Space Agency (ESA), the Institute of Electrical and Electronic Engineers (IEEE), the American National Standards Institute (ANSI), and the International Organization for Standardization (ISO). In particular, the ISO organization represents the international consensus and agreement from a number of member countries from around the world. Many of these countries might participate in the editing process of the standards and they must participate in the ballot of each stage of the standards development.

A 'standard' may be defined as an agreement between a number of – not necessarily all – players within a certain area of technology. That is to say, the word 'standard' is only used in cases where recognition has been granted by one or more standardisation bodies.

Schmidt (2000) has summarised the following benefits of the use of the software engineering standards:

1. Help in achieving greater conformance to software requirements, reduce the number of software defects, mitigate risks associated with the software, and decrease software maintenance costs.
2. Provide a framework for systematic, incremental software process improvements, and help to reduce the number of defects introduced during early project phases. This reduces the cost and schedule of the testing, installation, and maintenance phases.
3. Help in satisfying governmental regulations and industry quality standards as they relate to software, and is essential for passing audits and achieving certification.

The need to achieve compliance is a hard business reality for companies in a number of industries.

4. Provide enhanced accuracy of project planning, detailed means of tracking projects, early measures of software quality, and improved repeatability of success stories.

In addition to the above-mentioned benefits, standards are designed to promote the efficient use of technology, and can be seen as structured and pre-packaged, agreed-upon best practices for specific technologies (Abran, 1996).

In this chapter, we present in some details the contents of the ISO standards which are related to the software product quality. In particular, the following standards are discussed:

1. ISO 9126 on software product quality.
2. ISO 14598 on software product evaluation.
3. ISO 25020 on measurement reference model and guise (as a part of the SQuaRE).
4. ISO 25021 on quality measure elements (as a part of the SQuaRE).
5. ISO 25051 on requirements for quality of commercial off-the-shelf (COTS) software product and instructions for testing.
6. ISO 15939 on software measurement process.

In addition to the above-mentioned standards, the ISO VIM (on international vocabulary of basic and general terms in metrology) is discussed since this standard will be used – in Chapters 6 and 7 – along with ISO 15939 as a guide to investigate the availability of the metrology concepts within the ISO 9126 standards.

This chapter is structured as follows: Section 2.2 describes the contents of the ISO 9126 on software product quality. Section 2.3 presents the ISO 14598 on software product evaluation. Section 2.4 introduces the SQuaRE series of standards, in particular, it explains the ISO 25020 and ISO 25021 new standards. Section 2.5 presents the ISO

25051 on the requirements for quality of Commercial Off-The-Shelf (COTS) software product and instruction for testing. Section 2.6 gives a brief description of the contents of the ISO guide on the international vocabulary of basic and general terms in metrology (VIM). Finally, strengths and weakness of these ISO standards are identified and discussed.

2.2 ISO 9126: software product quality

In 1991, the ISO published its first international consensus on the terminology for the quality characteristics for software product evaluation. This standard was called Software Product Evaluation – Quality Characteristics and Guidelines for Their Use (ISO, 1991). From 2001 to 2004, the ISO published an expanded version, containing both the ISO quality models and the inventories of proposed measures for these models. The current version of the ISO 9126 series now consists of one International Standard and three Technical Reports:

1. ISO 9126 – Part 1: Quality Model (ISO, 2001b).
2. ISO TR 9126 – Part 2: External Measures (ISO, 2003c).
3. ISO TR 9126 – Part 3: Internal Measures (ISO, 2003d).
4. ISO TR 9126 – Part 4: Quality in-Use Measures (ISO, 2004f).

Within ISO 9126, there are six quality characteristics and twenty-seven quality subcharacteristics for the internal and external software product, and four quality characteristics for the in-use software product. For each of these subcharacteristics (and characteristics in the case of quality in-use), there are a number of measures that could be used to assess and evaluate software product quality from different viewpoints (e.g. internal functionality, external efficiency, in-use productivity, etc.).

The first part of ISO 9126 (ISO, 2001b) describes in some details a software product quality model which should be used with the other three parts. It is defined as a

framework which explains the relationship between different approaches to quality (ISO, 2001b), and considers software product quality from three distinct viewpoints (stages); that is, internal quality, external quality and quality in-use:

- Internal quality is "the totality of the characteristics of the software product from an internal view" (ISO, 2001b, p. 5). It can be realized by measuring the internal properties of the software product without executing it.
- External quality is "the totality of the characteristics of the software product from an external view" (ISO, 2001b, p. 5), that is, measuring the external quality properties of the software product during its execution.
- Quality in-use is "the user's view of the quality of the software product when it is used in a specific environment and a specific context of use" (ISO, 2001b, p. 5). It is related to the quality of the in-use software product, that is, during the operation and maintenance phases.

Figure 2.1 shows the ISO view of the expected relationships between internal quality, external quality and quality in-use attributes. The internal quality attributes influence the external quality attributes, while the external attributes influence the quality in-use attributes. Furthermore, quality in-use depends on external quality, while the external quality depends on the internal quality (ISO, 2001b).

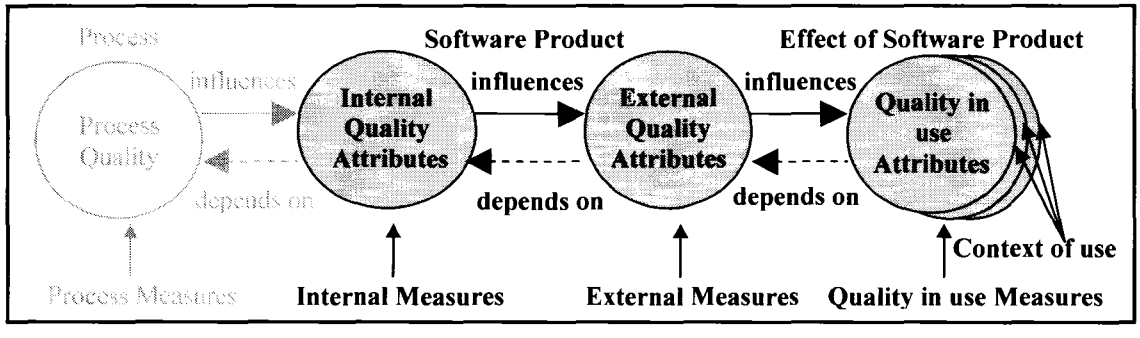

Figure 2.1 ***Quality in the life-cycle***
(ISO, 2001b)

Moreover, this document (ISO 9126-1) – Quality Model – contains a two-part quality model for software product quality (ISO, 2001b), that is:

1. Internal and external quality model;
2. Quality in-use model.

The first part of the two-part quality model determines six characteristics in which they are subdivided into twenty-seven subcharacteristics for internal and external quality, as in Figure 2.2 (ISO, 2001b). These subcharacteristics are a result of internal software attributes and are noticeable externally when the software is used as a part of a computer system. The second part of the two-part model indicates four quality in-use characteristics, as in Figure 2.3 (ISO, 2001b).

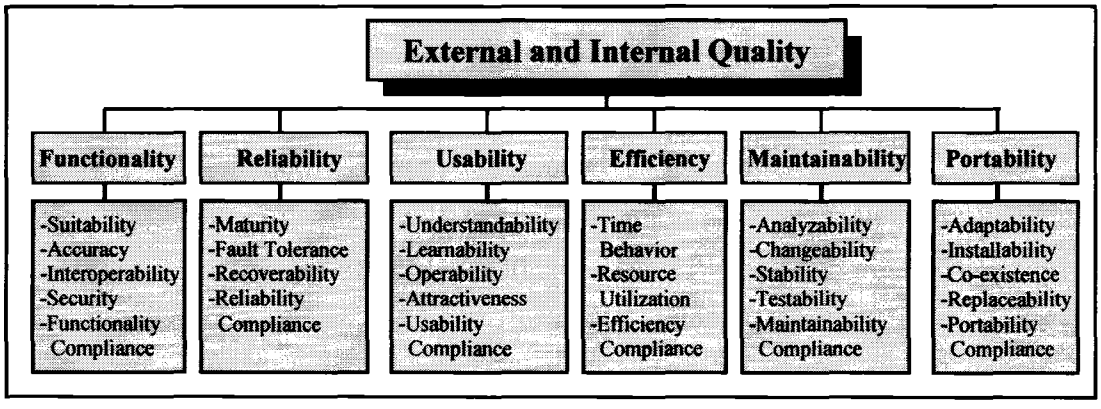

Figure 2.2 ***ISO 9126 quality model for external and internal quality (characteristics and subcharacteristics)***
(ISO, 2001b)

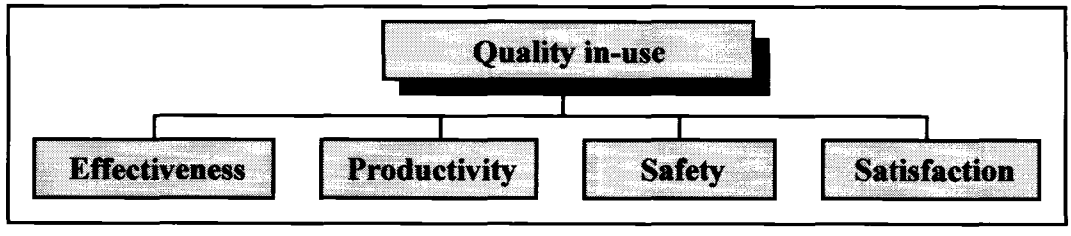

Figure 2.3 ***ISO 9126 quality model for quality in-use (characteristics)***
(ISO, 2001b)

In addition, Figure 2.4 shows the different views of product quality and associated measures at different stages in the software lifecycle (ISO, 2001b).

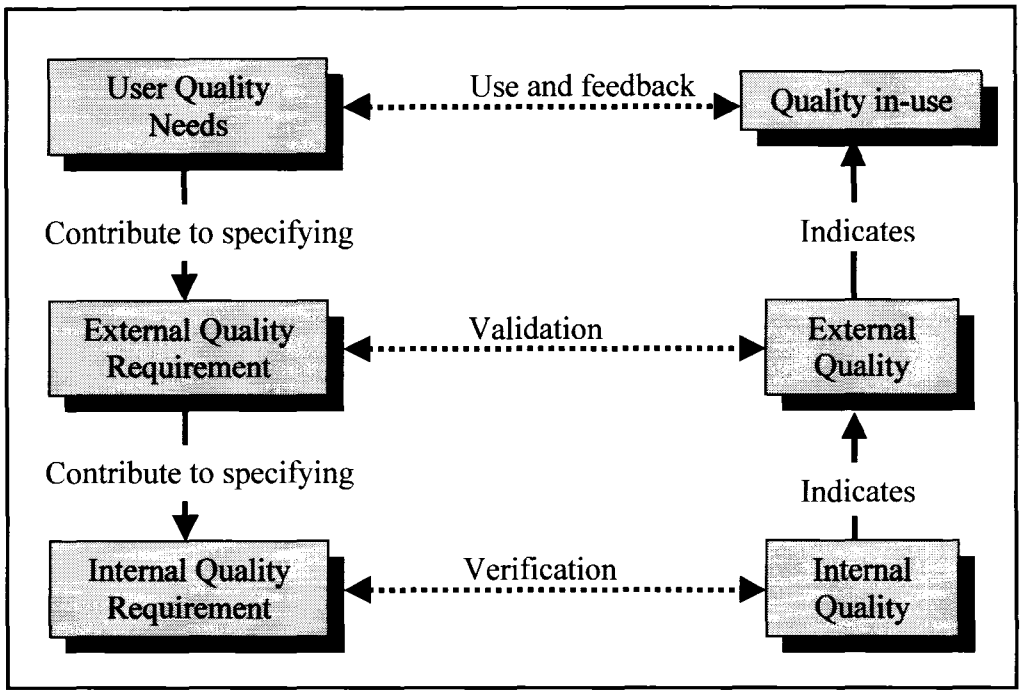

Figure 2.4 ***The quality during the software lifecycle***
(ISO, 2001b)

The second document of the ISO 9126 series – external measures – contains a basic set of measures for each external quality subcharacteristic, explanations of how to apply and use software external quality measures, and examples of how to apply these measures during the software product lifecycle (ISO, 2003c). The external measures are classified according to the characteristics and the subcharacteristics defined in ISO 9126-1.

The third document of the ISO 9126 series – internal measures – contains an inventory of measures for each internal quality subcharacteristic, explanations of the application of these measures, and examples of how to use these measures in the software product lifecycle (ISO, 2003d). Also, the internal measures are classified according to the characteristics and the subcharacteristics defined in ISO 9126-1.

Finally, the fourth document of the ISO 9126 series – quality in-use measures – contains a basic set of measures for each quality in-use characteristic, explanations of how to apply them, and examples of how to use them in the software product lifecycle (ISO, 2004f). The quality in-use measures are classified by the characteristics defined in ISO 9126-1.

Furthermore, this set of ISO standards could be used by the following intended users during the software development life cycle (ISO, 2001b):

1. Developers.
2. Quality managers.
3. Maintainers.
4. Evaluators.
5. Acquirers.
6. Suppliers.
7. Users.

2.3 ISO 14598: software product evaluation

In addition to the four documents of the ISO 9126 series, the ISO also published a set of documents for guidelines on how to apply ISO 9126, which is called ISO 14598 and named as software product evaluation. The ISO 14598 series of standards consists of six parts:

1. ISO 14598 – Part 1: General Overview (ISO, 1999a).
2. ISO 14598 – Part 2: Planning and Management (ISO, 2000a).
3. ISO 14598 – Part 3: Process for Developers (ISO, 2000b).
4. ISO 14598 – Part 4: Process for Acquirers (ISO, 1999b).
5. ISO 14598 – Part 5: Process for Evaluators (ISO, 1998a).
6. ISO 14598 – Part 6: Documentation of Evaluation Modules (ISO, 2001a).

The part-1 of the ISO 14598 series of standards – general overview – contains an overview of the contents and the objectives of the other parts, defines a number of terms used in the other parts, and illustrates the relationship between the other five parts (ISO, 1999a). In addition, it clarifies the relationship between the quality model in the ISO 9126 part-1 and the ISO 14598 series of standards, includes the general requirements for the specification and the evaluation of the software quality, and presents a framework to evaluate the quality of all types of software product (ISO, 1999a).

Figure 2.5 illustrates the relationship between the parts 3, 4, and 5 (evaluation process) and the parts 2 and 6 (evaluation support) of the ISO 14598 series of standards (ISO, 1999a).

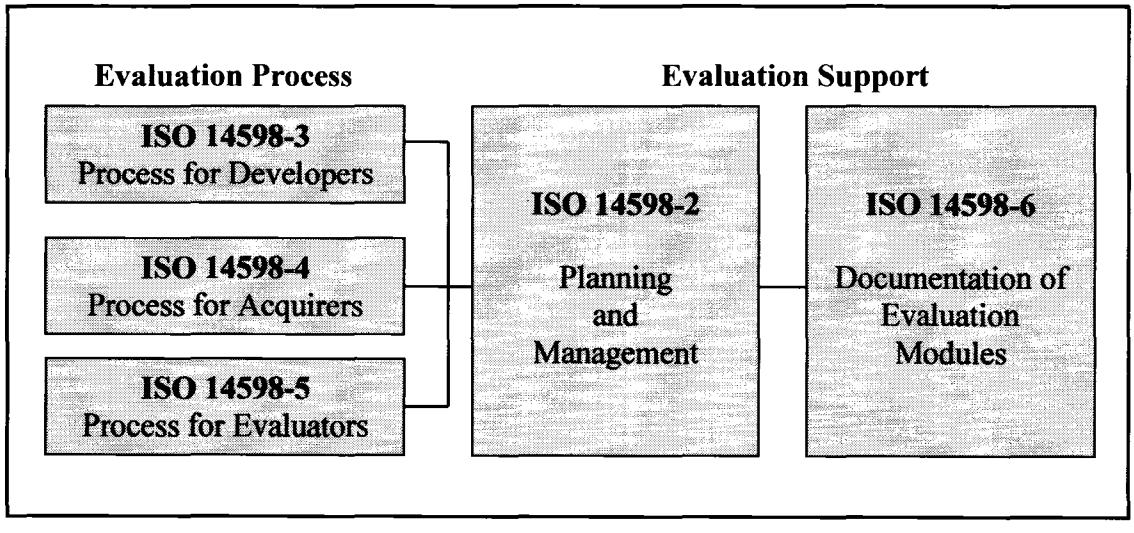

Figure 2.5 ***The relationship between the evaluation process and the evaluation support***
(ISO, 1999a)

The ISO 14598 part-2 – planning and management – presents details about the planning and management requirements that are associated with the software product evaluation, and it defines the requirements which should be provided by the organization in order to ensure the success of the evaluation process (ISO, 2000a).

The part-3 of the ISO 14598 series of standards – process for developers – may be used to apply the concepts explained in the ISO 9126 series of standards and the ISO 14598 parts 1, 2, and 6 (ISO, 2000b). It gives recommendations and requirements for the practical implementation of the software product evaluation, in parallel with the development, by the developer (ISO, 2000b). This part of the ISO 14598 series of standards may be used by the following (ISO, 2000b):

1. Project manager.
2. Software designer.
3. Quality assurance audit.
4. Maintainer.
5. Software acquirer.

The part-4 of the ISO 14598 series of standards – process for acquirers – includes requirements, recommendations and guidelines for the systematic measurement, assessment and evaluation of the software product quality (ISO, 1999b). The evaluation process explained in this part of the ISO 14598 series of standards helps to meet the objectives and the goals of deciding on the acceptance of a single product or for selecting a product (ISO, 1999b). This part-4 of the ISO 14598 series may be used by (ISO, 1999b):

1. Project manager.
2. System engineers.
3. Software engineering staff.
4. End users.

The ISO 14598 part-5 – process for evaluators – may be used to apply the concepts explained in ISO 9126 series of standards by providing requirements and recommendations for the practical implementation of the software product evaluation when several parties need to understand, accept, and trust the evaluation results (ISO, 1998a). The evaluation process explained in this part of the ISO 14598 series defines the

activities needed to analyze the evaluation requirements, to specify, design, and perform the evaluation actions and to conclude the evaluation of any kind of software product (ISO, 1998a). This part of the ISO 14598 series may be used by (ISO, 1998a):

1. Testing laboratory evaluators.
2. Software suppliers.
3. Software acquirer.
4. Software users.
5. Certification bodies.

Finally, the part-6 of the ISO 14598 series of standards – documentation of evaluation modules – clarifies and defines the contents, the formation, and the structure of the documentation to be used to illustrate an evaluation module (ISO, 2001a). This part of the ISO 14598 series may be used by testing laboratories, research institutions and organizations, and any others who need to produce new evaluation modules (ISO, 2001a).

2.4 SQuaRE series of standards

The ISO has now recognized a need for further enhancement of ISO 9126 and ISO 14598 series of standards, primarily as a result of advances in the fields of information technologies and changes in environment (Azuma, 2001). Therefore, the ISO is now working on the next generation of software product quality standards (Suryn, Abran and April, 2003), which will be referred to as Software Product Quality Requirements and Evaluation (ISO SQuaRE – ISO 25000 series). This series of standards will replace the current ISO 9126 and ISO 14598 series of standards. The ISO SQuaRE series consists of five divisions, as in Figure 2.6 (ISO, 2005):

1. ISO 2500n: Quality Management Division.
2. ISO 2501n: Quality Model Division.
3. ISO 2502n: Quality Measurement Division.

4. ISO 2503n: Quality Requirements Division.
5. ISO 2504n: Quality Evaluation Division.

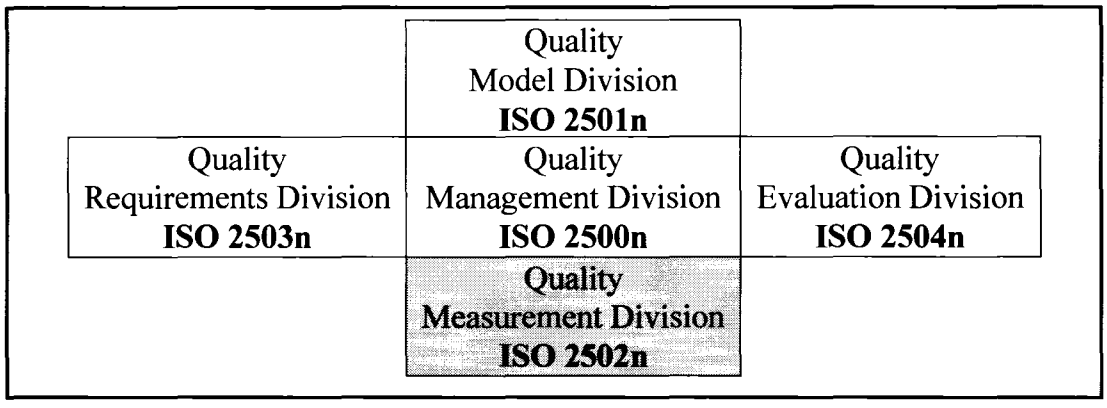

Figure 2.6 ***Organization of the ISO SQuaRE series of standards***
(ISO, 2005)

One of the main objectives of (and differences between) the ISO SQuaRE series of standards and the current ISO 9126 series of standards is the coordination and harmonization of its contents with ISO 15939 (ISO, 2005). Figure 2.7 shows the structure of the quality measurement division (ISO 2502n) series that is to replace the current four-part ISO 9126 series of standards (ISO, 2007a). This quality measurement division (ISO 2502n) consists of five standards (ISO, 2007a):

1. ISO 25020: Measurement Reference Model and Guide.
2. ISO 25021: Quality Measure Elements.
3. ISO 25022: Measurement of Internal Quality.
4. ISO 25023: Measurement of External Quality.
5. ISO 25024: Measurement of Quality in-Use.

In particular, ISO has recently finalized two documents from the quality measurement division standards, that is, ISO 25020 and ISO 25021. These two new standards will be analyzed in details in Chapter 7.

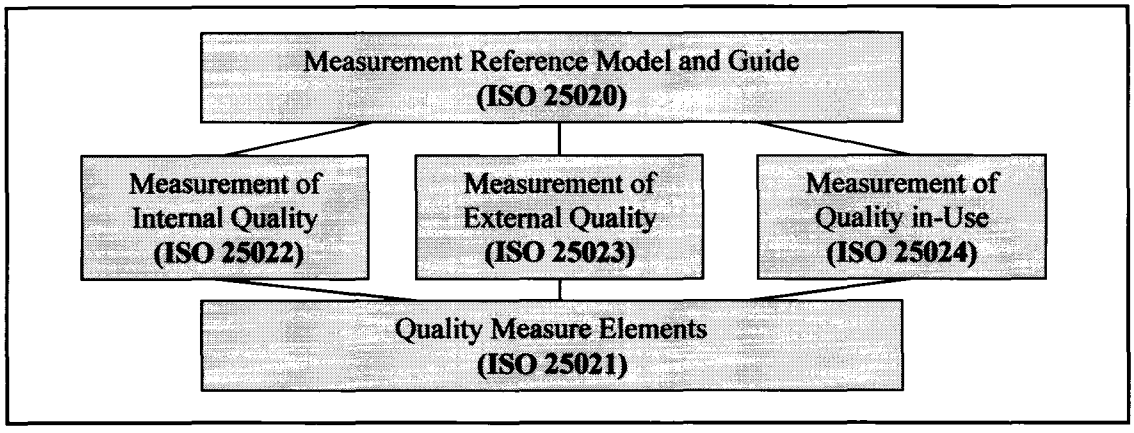

Figure 2.7 ***Structure of the measurement division (ISO 2502n)*** *(ISO, 2007a)*

The ISO 25020 and ISO 25021 introduced the following new terms to be used in all of the related SQuaRE standards (i.e. ISO 25022, ISO 25023 and ISO 25024) (ISO, 2007a; 2007b):

1. Quality measure element category.
2. Quality measure element.
3. Quality measure.

In ISO 25020, the quality measure is defined as the measure of internal software quality, external software quality or software quality in-use, and the quality measure element is defined to be a base or derived measure which is used to construct the software quality measures (ISO, 2007a). Although, the quality measure element categories are introduced in ISO 25021, they are never defined nor mentioned in ISO 25020 which is the guide for the users of the quality measurement division standards. In addition, the list of quality measures in ISO 25021 arises from a set of surveys involving several commercial and academic institutions (ISO, 2007b).

These two standards may be used by the following intended users to evaluate the quality of a specific software product (ISO, 2007a):

1. Developers.
2. Acquirers.
3. Independent evaluators.

2.5 ISO 25051: requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing

This standard was initially published in 1994 as ISO 12119 – Software Packages: Quality Requirements and Testing – (ISO, 1994). In 2006, it was updated and republished to be a part of the SQuaRE series of standards as ISO 25051: Requirements for Quality of Commercial Off-The-Shelf (COTS) Software Product and Instructions for Testing (ISO, 2006b). The ISO 25051 international standard provides requirements for COTS software product, requirements for test documentation, and instructions for conformity evaluation, including requirements for product description requirements for user documentation, and quality requirements for software (ISO, 2006b). In Annex C of this international standard, it provides guidance and recommendations for safety or business critical COTS software products (ISO, 2006b).

The quality requirements for the COTS software product itself consist of the following product quality characteristics (ISO, 2006b):

1. Functionality.
2. Reliability.
3. Usability.
4. Efficiency.
5. Maintainability.
6. Portability.
7. Quality in-use.

In addition, it can be used to assess the software product user documentation through the following quality requirements (ISO, 2006b):

1. Completeness.
2. Correctness.
3. Consistency.
4. Understandability.
5. Learnability.
6. Operability.

Furthermore, it contains a set of requirements for the testing process. These requirements can be summarised as follows (ISO, 2006b):

1. Requirements for the test plan.
2. Requirements for the testing description.
3. Requirements for the test results.

This international standard could be used by the following potential users to assess the quality of the COTS software products (ISO, 2006b):

1. Suppliers.
2. Certification bodies.
3. Testing laboratories.
4. Accreditation bodies.
5. Acquirers.
6. End users.

2.6 ISO 15939: software measurement process

The goal of this international standard is to identify the activities and tasks which are necessary to identify, define, select, apply, and improve the software measurement successfully. In addition, it contains a set of definitions for the commonly used terms in the field of software measurement (ISO, 2002).

Furthermore, this international standard can be used to implement a measurement process on an overall project or organizational measurement structure. The following activities and tasks should be followed for this implementation (ISO, 2002, p. 10):

1. Establish and sustain measurement commitment, it consists of the following tasks:
    a. Accept the requirements for measurement.
    b. Assign resources.
2. Plan the measurement process, it includes the following tasks:
    a. Characterise organisational unit.
    b. Identify information needs.
    c. Select measures.
    d. Define data collection, analysis, and reporting procedures.
    e. Define criteria for evaluating the information products and the measurement process.
    f. Review, approve, and provide resources for measurement tasks.
    g. Acquire and deploy supporting technologies.
3. Perform the measurement process.
    a. Integrate procedures.
    b. Collect data.
    c. Analyse data and develop information products.
    d. Communicate results.
4. Evaluate the measurement process.
    a. Evaluate information products and the measurement process.
    b. Identify potential improvements.

Within ISO 15939, ISO produced an information model – as in Figure 2.8 – to help in determining what has to be specified during measurement planning, performance and evaluation (ISO, 2002).

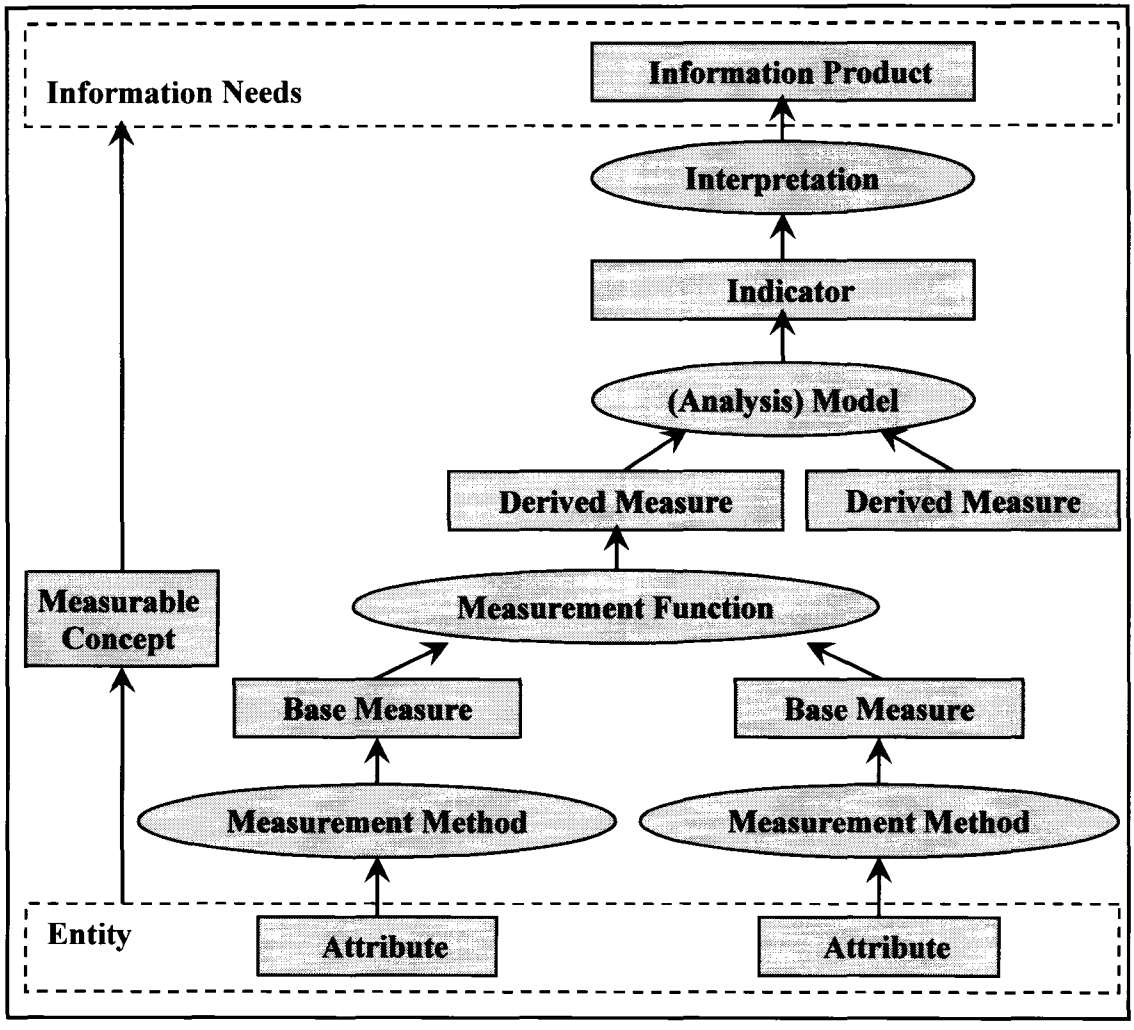

Figure 2.8 ***Measurement information model from ISO 15939***
(ISO, 2002)

Figure 2.8 shows that a specific measurement method is used to collect a base measure for a specific attribute. Then, the values of two or more base measures can be used within a computational formula (by means of a measurement function) to produce and construct a specific derived measure. These derived measures are then used in the context of an analysis model to achieve an indicator which is a value, and to interpret the indicator's value to explain the relationship between it and the information needed, in the language of the measurement user, to produce an Information Product for his Information Needs (ISO, 2002).

Abran *et al.* (2005) have divided the ISO 15939 Measurement Information Model into three sections, that is, 'Data Collection', 'Data Preparation', and 'Data Analysis' sections, as described in Figure 2.9.

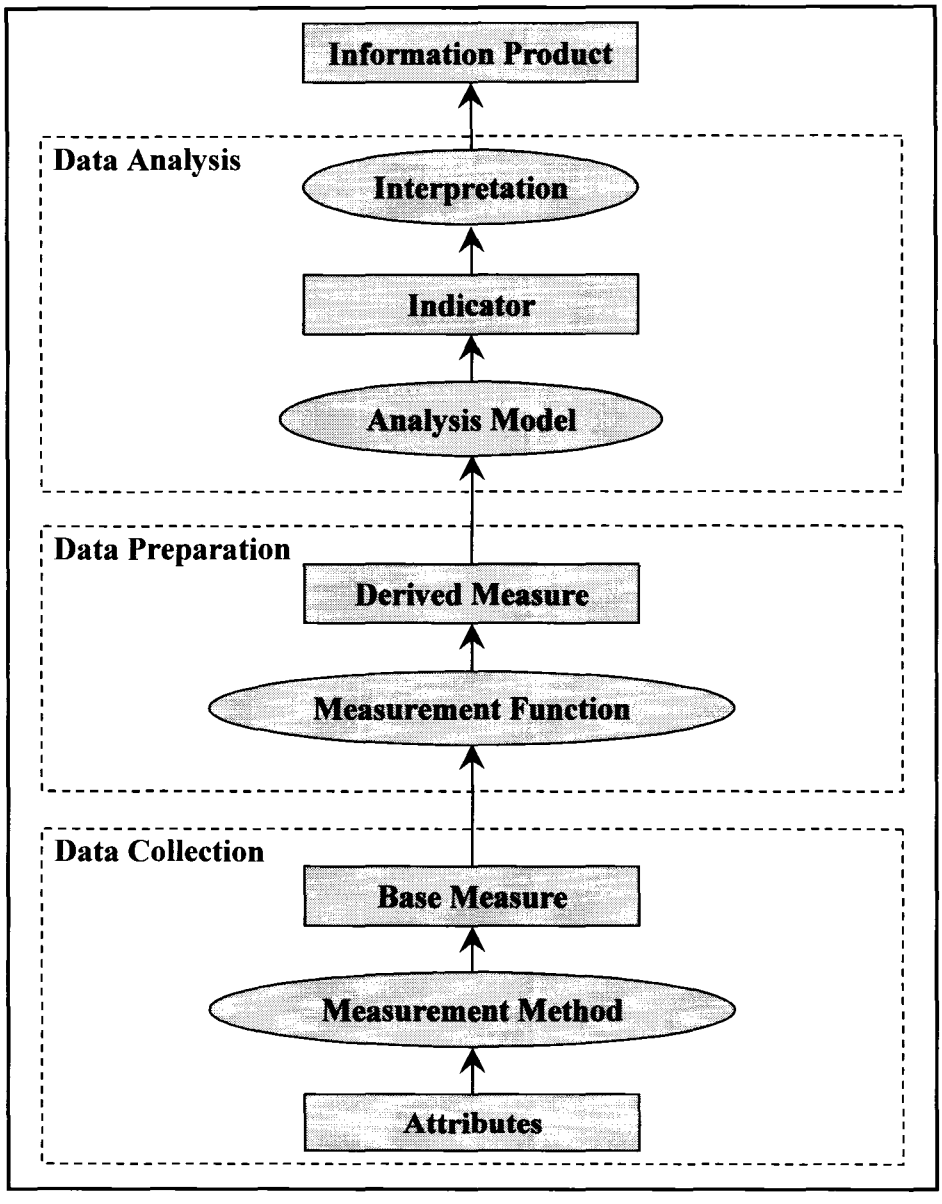

Figure 2.9 ***ISO 15939 measurement information model – three different sections***

2.7 ISO VIM: international vocabulary of basic and general terms in metrology

Metrology is the basis of all measurement related concepts in natural sciences and engineering, and to each of the different interpretations of a software 'metrics' is associated a related distinct metrology term with related metrology criteria and relationship to other measurement concepts. In 1984, the ISO with other participating organizations (BIPM, IEC, and OIML) published their first edition of the international consensus on the basic and general terms in metrology (ISO VIM) (ISO, 1984). Later on, in 1993, this publication was reviewed, and then the ISO in collaboration with six participating organizations (BIPM, IEC, OIML, IUPAC, IUPAP, and IFCC) published the second edition of this document (ISO, 1993). ISO is now working on its third edition of this document to integrate – in particular – concepts related to measurement uncertainty and measurement traceability.

The term 'metrology' is defined in the ISO International Vocabulary of Basic and General Terms in Metrology as the field of knowledge dealing with measurement (ISO, 1993). More specifically, it has been defined by Simpson (1981) as the portion of measurement science used to provide, maintain, and disseminate a consistent set of units; to provide support for the enforcement of equity in trade by weights and measurement laws; or to provide data for quality control in manufacturing.

This ISO document consists of 120 terms classified into six categories (in parentheses, the number of terms within each category):

1. Quantities and Units (22 terms).
2. Measurements (9 terms).
3. Measurement Results (16 terms).
4. Measuring Instruments (31 terms).
5. Characteristics of the Measuring Instruments (28 terms).
6. Measurement Standards – Etalon (14 terms).

The ISO VIM standard on metrology presents its terms, within each category, in increasing order of complexity and describes in textual format each term individually. To facilitate understanding of these more than hundred related terms, Abran and Sellami (2002a) proposed a modeling of all the sets of measurement concepts documented in this ISO document.

Two of the categories of terms in the VIM deal with some aspects of the design of measurement methods, that is, 'quantities and units' and 'measurement standards – etalon'. The other four categories are related to the application of a measurement design with a measuring instrument and to the quality characteristics of the measurement results provided by this measuring instrument (Abran and Sellami, 2002a). In this thesis, we use the modeling proposed by Abran and Sellami (2002a) of the measurement concepts in the ISO vocabulary of terms in metrology. In particular, we will use the first category which deals with the design of the measurement methods, that is, 'quantities and units' to analyse ISO 9126 (see Chapter 7 for this analysis).

Table 2.1, shows the detailed structure of the 'quantities and units' category; in particular, the system of quantities consists of two types of quantities, that is, base and derived quantities (in ISO 15939, the 'base quantities' and 'derived quantities' terms were replaced by the following two equivalent terms: 'base measures' and 'derived measures'.)

Table 2.1

Detailed structure of the 'quantities and units' category
(Abran and Sellami, 2002a)

Quantities and Units			
System of Quantities	**Dimensions of Quantities**	**Units of Measurement**	**Value of a Quantity**
- Base Quantity - Derived Quantity	- Quantity of Dimension one (Dimensionless Quantity)	- Symbol of a Unit - System of Units - Coherent (Derived) Unit - Coherent System of Units - International System of Units (SI) - Base Unit - Derived Unit - Off-System Unit - Multiple of a Unit - Submultiple of a Unit	- True Value - Conventional True Value - Numerical Value - Conventional Reference Scale (Reference-Value Scale)

Table 2.2 presents the related terms along with their definitions adopted in this thesis (particularly, in Chapter 7) from the 'quantities and units' category of the ISO VIM on International Vocabulary of Basic and General Terms in Metrology (ISO, 1993).

While metrology has a long tradition of use in physics and chemistry, it is rarely referred to in the software engineering literature. Carnahan *et al.* (1997) are amongst the first authors to identify this gap in what they referred to as 'IT metrology' and to discuss the challenges and opportunities arising from the application of the metrology concepts to information technology. In addition, they proposed logical relationships between metrology concepts, as in Figure 2.10. Moreover, Gray (1999) discussed the applicability of metrology to information technology form the software measurement point of view.

Table 2.2

Definitions of some metrology terms
(ISO, 1993)

Term	Definition
Quantity	Attribute of a phenomenon, body or substance that may be distinguished qualitatively and determined quantitatively.
System of Quantities	A set of quantities, in the general sense, among which defined relationships exist.
Unit (of Measurement)	A particular quantity defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitudes relative to that quantity.
Base Quantity	One of the quantities that, in a system of quantities, are conventionally accepted as functionally independent of one another.
Derived Quantity	A quantity defined, in a system of quantities, as a function of base quantities of that system.
Quantity of Dimension one (Dimensionless Quantity)	A quantity in the dimensional expression of which all the exponents of the dimensions of the base quantities reduce to zero.
System of Units	A set of base units, together with derived units, defined in accordance with given rules, for a given system of quantities.
Coherent (Derived) Unit	A derived unit of measurement that may be expressed as a product of powers of base units with the proportionality factor one.
Coherent System of Units	A system of units of measurement in which all of the derived units are coherent.
International System of Units (SI)	The coherent system of units adopted and recommended by the General Conference on Weights and Measures (CGPM).
Base Unit	A unit of measurement of a base quantity in a given system of quantities.
Derived Unit	A unit of measurement of a derived quantity in a given system of quantities.
Off-System Unit	A unit of measurement that does not belong to a given system of units.
Multiple of a Unit	A larger unit of measurement that is formed from a given unit according to scaling conventions.
Submultiple of a Unit	A smaller unit of measurement that is formed from a given unit according to scaling conventions.
Value of a Quantity	Magnitude of a particular quantity generally expressed as a unit of measurement multiplied by a number.
True Value	A value consistent with the definition of a given particular quantity.
Conventional True Value	A value attributed to a particular quantity and accepted, sometimes by convention, as having an uncertainty appropriate for a given purpose.
Numerical Value	A quotient of the value of a quantity and the unit used in its expression.
Conventional Reference Scale	For particular quantities of a given kind, an ordered set of values, continuous or discrete, defined by convention as a reference for arranging quantities of that kind in order of magnitude.

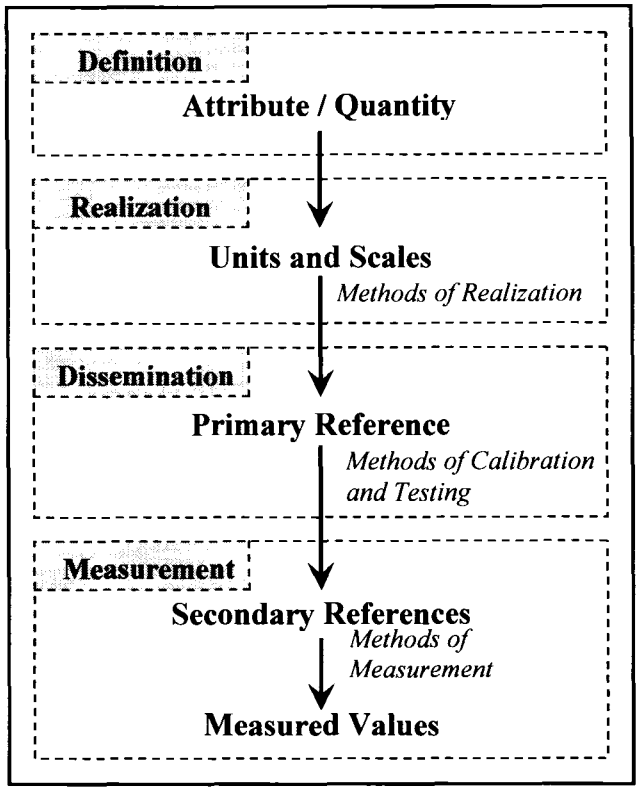

Figure 2.10 ***Logical relationships among metrology concepts in standardizing measurements***
(Carnahan et al., 1997)

Abran (1998) highlighted some high-level ambiguities in the domain of software measurement and proposed to substitute the appropriate metrology terms to the current ambiguous and peculiar 'software metrics' terminology unique to the domain of software engineering. In addition, the availability of the metrology concepts in software engineering has been investigated in (Abran and Sellami, 2002b; Abran, Sellami and Suryn, 2003; Bourque et al., 2004). Abran and Sellami (2004) documented the metrology concepts addressed in the ISO 19761 (COSMIC-FFP), both in the design of this measurement method and in some of its practical use. Moreover, Sellami and Abran (2003) investigated the contribution of metrology concepts to understand and clarify a framework for software measurement validation proposed by Kitchenham *et al.* (1995).

2.8 Identification of the strengths and weaknesses

The ISO 9126 series of standards on software product quality evaluation proposes a set of 197 metrics (measures) for measuring the various characteristics and subcharacteristics of software quality (ISO, 2003c; 2003d; 2004f). However, as it is typical in the software engineering literature, the set of so-called metrics in ISO 9126 refers to multiple distinct concepts which, in metrology, would have distinct labels (or naming conventions, e.g. terms) to avoid ambiguities. Therefore, to help in understanding and clarifying the nature of the 'metrics' proposed in ISO TR 9126, each needs to be analysed from a metrology perspective and mapped to the relevant metrology concepts. Such a mapping will also contribute to identifying the measurement concepts that have not yet been tackled in the ISO 9126 series of documents. Each of these gaps represents an opportunity for improvement in the design and documentation of the measures proposed in ISO 9126.

Recently, the ISO has come up with the ISO 25020 and ISO 25021 documents as apart of the quality measurement division of the SQuaRE series of standards (ISO, 2007a; 2007b). Within these two documents, the associated working group (WG6) of software engineering subcommittee seven (SC7) has introduced three new terms namely: quality measure elements, quality measure element categories and quality measures. However, there already exists a very mature measurement terminology, and it is well documented in the ISO International Vocabulary of Basic and General terms in Metrology (ISO VIM) (ISO, 1993). This terminology is widely accepted and used in most fields of science, and has been adopted in ISO 15939 (ISO, 2002) as the agreed upon measurement terminology for software and system engineering related ISO standards. Thus, using well-defined terms – such as in ISO 15939 and ISO VIM – is more efficient and usable than adding of new terms for the new SQuaRE series of standards – such as in ISO 25020 and ISO 25021. In addition, as a main goal of the new ISO 25020 and ISO 25021 to be aligned with the ISO 15939 international standard, thus, an investigation is

needed to ensure that the ISO 15939 is completely and correctly mapped to the new two standards.

The ISO 25051 – which is a part of the SQuaRE series of standard – is completely dedicated to the COTS software product (ISO, 2006b). However, since the COTS is a type of software product, thus, some of contents of this standard are already included in the other SQuaRE documents. In addition, the ISO 25051 contains some materials that is related to the requirement for user documentation which is already available as a single ISO standard, as in ISO 18019 (ISO, 2004d).

CHAPTER 3

CAPABILITY AND MATURITY MODELS IN THE SOFTWARE ENGINEERING LITERATURE

3.1 Introduction

A maturity model is a structured collection of elements that describe the characteristics of effective processes or products. In addition, a maturity model can be used as a benchmark for assessing different processes or products for equivalent comparison.

In the software engineering literature, there are many capability and maturity models. These capability and maturity models could be classified into process or product capability and maturity models based on what aspect of the software they could be applied.

This chapter is structured as follows: Section 3.2 presents some examples of the many available process maturity models related to software engineering. By contrast, only two maturity models are identified as dealing with the software product. In Section 3.3, we discuss these two product maturity models, that is: the Open Source Maturity Model (OSMM) (Golden, 2004) and the Software product Maturity Model (Nastro, 1997). At the end of this chapter – in Section 3.4 – we discuss the strengths and weaknesses of maturity models which are related to the software product rather than to the software process.

3.2 Process maturity models

The maturity models of software development processes are useful because they indicate different levels of process performance and, consequently, the direction in which a

software process should improve (McBride, Henderson-Sellers and Zowghi, 2004). In this section we discuss the following process capability and maturity models:

1. Capability Maturity Model Integration for Software Engineering – CMMI-SW.
2. Testing Maturity Model – TMM.
3. ISO Process Assessment Model – ISO 15504.

3.2.1 Capability maturity model integration for software engineering

The Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) of Carnegie Mellon University in response to a request to provide the U.S. Department of Defense with a method for assessing its software contractors (SEI, 1993). Within the updated Capability Maturity Model Integration (CMMI) version are currently embedded four bodies of knowledge (disciplines) (SEI, 2002a; 2002b):

1. System Engineering.
2. Software Engineering.
3. Integrated Product and Process Development.
4. Supplier Sourcing.

In this subsection, we provide an overview of only the Capability Maturity Model Integration for Software Engineering (CMMI-SW).

The components of the CMMI-SW are process areas, specific goals, specific practices, generic goals, generic practices, typical work products, subpractices, notes, discipline amplifications, generic practice elaborations and references (SEI, 2002b). The CMMI-SW can organize process areas in a staged or a continuous representation. The staged representation includes five maturity levels to support and guide process improvement; in addition, it groups process areas by maturity level, indicating which process areas to implement to achieve each maturity level (SEI, 2002b). Figure 3.1 presents the CMMI-SW maturity level architecture.

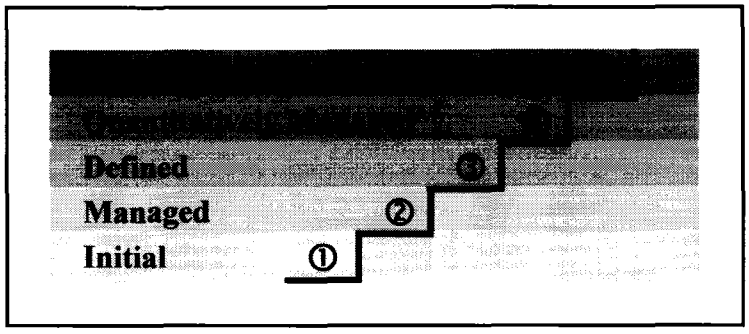

Figure 3.1 ***CMMI-SW maturity levels***
(SEI, 2002b)

Furthermore, Figure 3.2 shows the CMMI-SW model components in a staged representation, and illustrates the relationships between those components (SEI, 2002b).

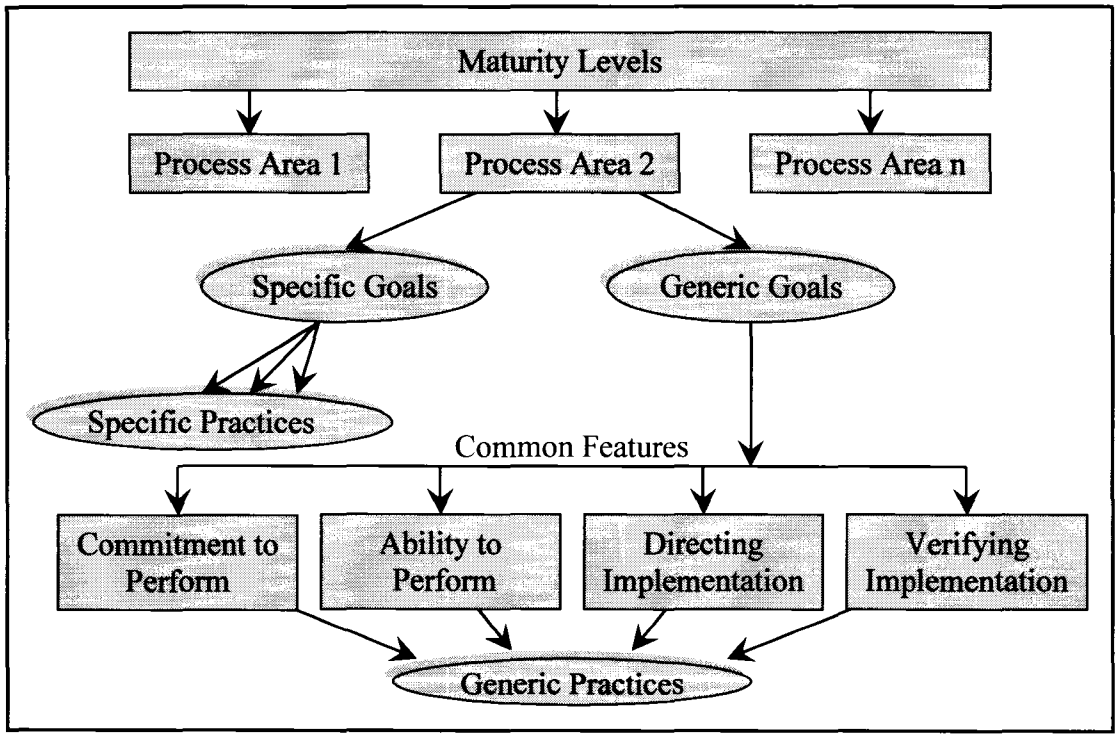

Figure 3.2 ***CMMI-SW components***
(SEI, 2002b)

3.2.2 Testing maturity model

In addition to the SEI maturity model (CMMI-SW), there are a number of maturity models in the literature dealing with the software process. For example, the Testing Maturity Model (TMM) (Burnstein, Suwanassart and Carlson, 1996a; 1996b) which is related to the software testing process. This maturity model is a quantitative model, that is, it is based on measurements. It was developed by Burnstein *et al.* (1996a; 1996b) at the Illinois Institute of Technology. The TMM consists of five levels of testing maturity (see Figure 3.3), each with maturity goals identifying testing improvements that should be addressed to achieve the next level of maturity (Burnstein, Suwanassart and Carlson, 1996a).

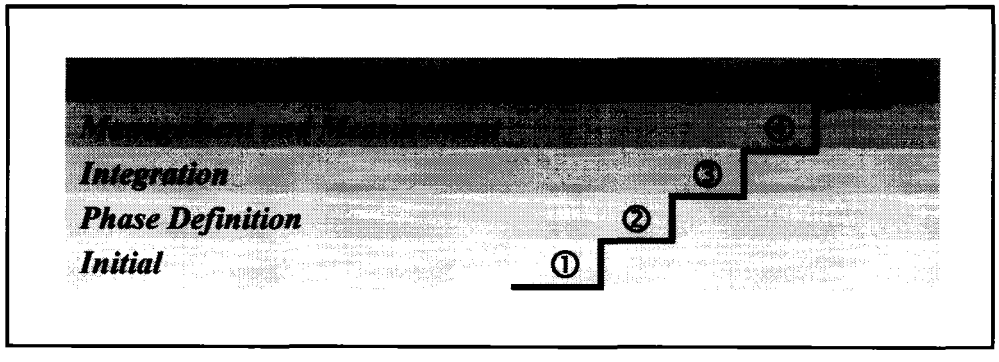

Figure 3.3 ***Testing maturity levels***

3.2.3 ISO 15504: software process assessment

ISO 15504 consists of a set of documents related to Software Process Assessment. It was first published in 1998 as a series of 9 Technical Reports. During 2003 to 2005, ISO has re-published this international standard as a 5-part series:

1. ISO 15504-1: Concepts and Vocabulary (ISO, 2004a).
2. ISO 15504-2: Performing an Assessment (ISO, 2003a).
3. ISO 15504-3: Guidance on Performing an Assessment (ISO, 2004b).

4. ISO 15504-4: Guidance on use for Process Improvement and Process Capability Determination (ISO, 2004c).
5. ISO 15504-5: An Exemplar Process Assessment Model (ISO, 2006a).

The first Part – Concepts and Vocabulary – is an entry point into ISO 15504. It gives an introduction to the concepts of this international standard, and defines a number of related terms (ISO, 2004a). In addition, this part describes how the other four parts fit together, and provides guidance for their selection and use (ISO, 2004a). Figure 3.4 shows a potential roadmap for users of this international standard (ISO, 2004a).

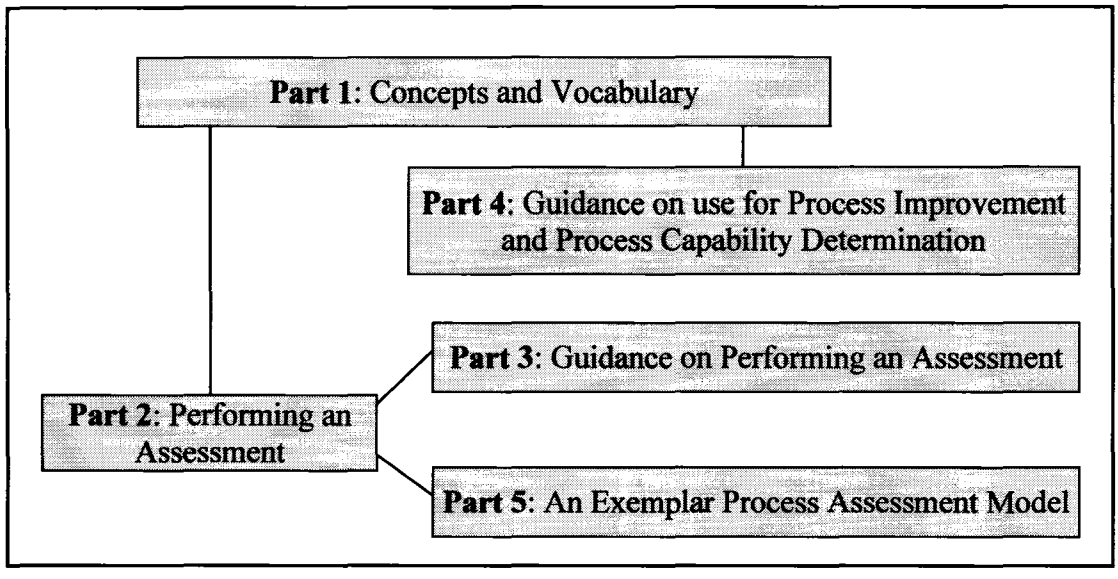

Figure 3.4 ***A potential roadmap for the users of ISO 15504***
(ISO, 2004a)

The second Part – Performing an Assessment – of this international standard contains normative requirements for process assessment and for process models in an assessment, and defines a measurement framework for evaluating process capability. The measurement framework defines nine process attributes that are grouped into six process capability levels that define an ordinal scale of capability that is applicable across all

selected processes. In addition, this part describes the relationships between the components of the process assessment model, as in Figure 3.5 (ISO, 2003a).

Figure 3.6 illustrates the relationships between the process attributes and their ratings and the corresponding capability levels. In this figure 3.6, the capability levels start at level one, that is, level zero is excluded since it indicates that the process is not implemented, or fails to achieve its process purpose.

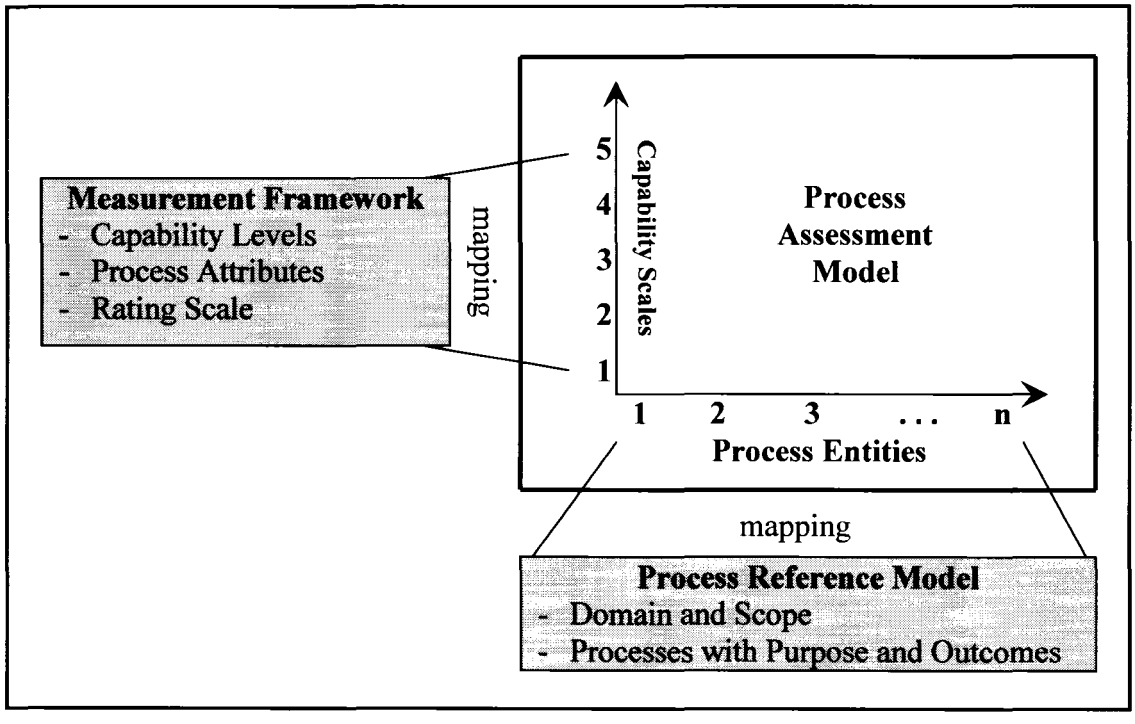

Figure 3.5 ***Process assessment model relationships***
(ISO, 2003a)

Furthermore, this Part 2 – Performing an Assessment – of the ISO 15504 introduces the following rating categories to be used in order to rate each of the process attributes (ISO, 2003a):

- N: Not achieved (0% - 15% achievement).
- P: Partially achieved (15% - 50% achievement).

- L: Largely achieved (50% - 85% achievement).
- F: Fully achieved (85% - 100% achievement).

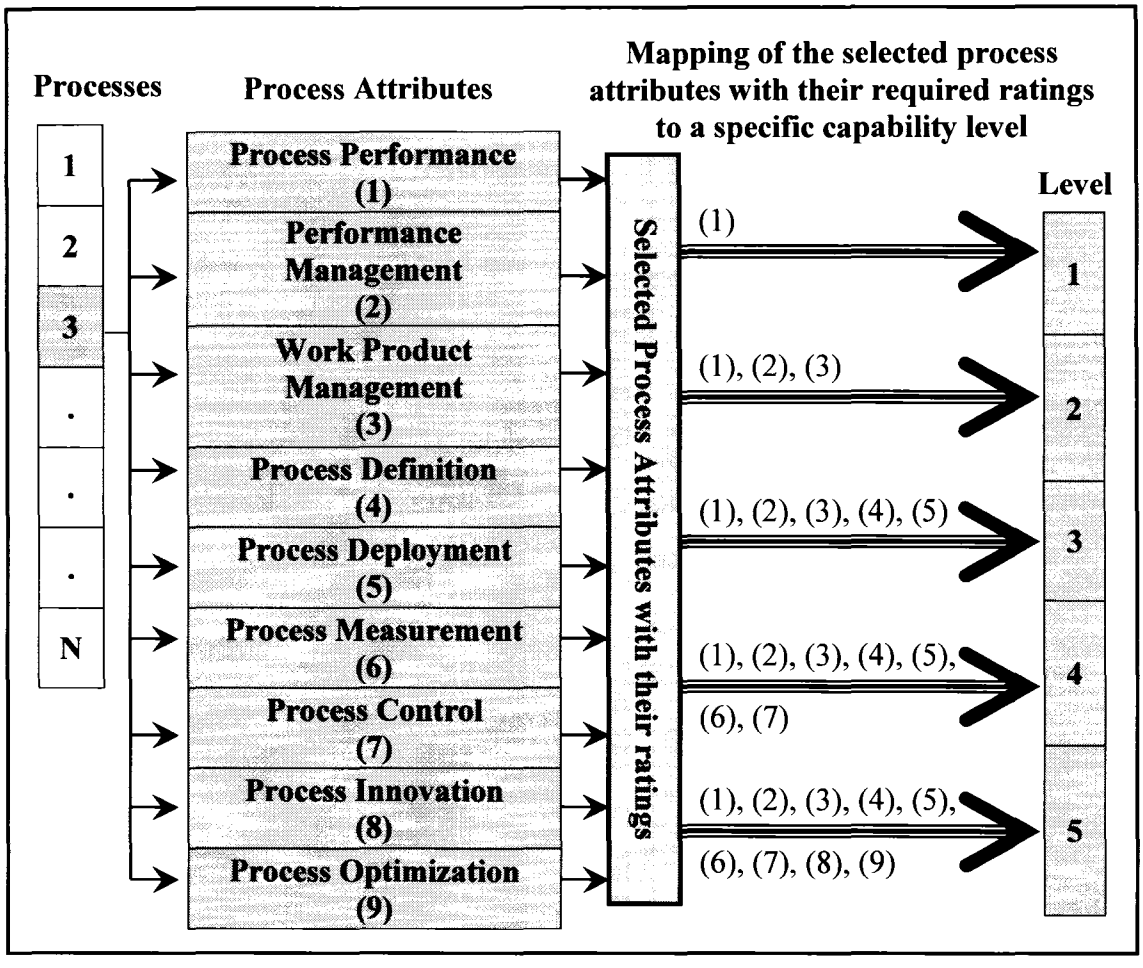

Figure 3.6 ***The relationships between the process attributes, their ratings and the corresponding capability levels***

The third Part – Guidance on Performing an Assessment – provides guidance on how to meet the minimum set of requirements for performing an assessment contained in the second part – Performing an Assessment – of this standard (ISO, 2004b). It provides an overview of process assessment and interprets the requirements through the provision of guidance on (ISO, 2004b):

1. Performing an assessment.

2. Measurement framework for process capability.
3. Process reference models and process assessment models.
4. Selecting and using assessment tools.
5. Competency of assessors.
6. Verification of conformity.

In addition, this part also provides an exemplar documented assessment process in its Annex A (ISO, 2004b).

The fourth Part – Guidance on use for Process Improvement and Process Capability Determination – provides guidance on how to utilize a conformant process assessment within a process improvement program or for process capability determination (ISO, 2004c). Within a process improvement context, process assessment provides a means of characterizing an organizational unit in terms of the capability of selected processes. Analysis of the output of a conformant process assessment against an organizational unit's business goals identifies strengths, weaknesses and risks related to the processes. In addition, this can help determine whether the processes are effective in achieving business goals, and provide the drivers for making improvements. Process capability determination is concerned with analyzing the output of one or more conformant process assessments to identify the strengths, weaknesses and risks involved in undertaking a specific project using the selected processes within a given organizational unit (ISO, 2004c).

Finally, the fifth Part – An Exemplar Process Assessment Model – provides an exemplar model for performing process assessments that is based upon and directly compatible with the Process Reference Model in ISO 12207 Amendment 1 and Amendment 2 (ISO, 2006a). The process dimension is provided by an external Process Reference Model, which defines a set of processes, characterized by statements of process purpose and process outcomes (ISO, 2006a). The capability dimension is based upon the

Measurement Framework defined in Part 2 – Performing an Assessment – of this standard. The assessment model(s) extend the Process Reference Model and the Measurement Framework through the inclusion of a comprehensive set of indicators of process performance and capability (ISO, 2006a).

The potential users of this set of standards are the following (ISO, 2004a):

1- Assessors.

2- Acquirers.

3- Suppliers.

3.3 Product maturity models

In the software engineering literature, we find only the following two maturity models which are related to the software product:

- Open Source Maturity Model – OSMM (Golden, 2004).
- Software Product Maturity Model (Nastro, 1997).

It must be noted, however, that these two models of software product do not address the quality of these products.

Within this section, we provide a brief description for these two models.

3.3.1 Open source maturity model

The Open Source Maturity Model (OSMM) (Golden, 2004) is designed to help organizations successfully implement open-source software. The OSMM is a three-phase process, and performs the following tasks, as in Figure 3.7:

1. Assessment of the maturity element.
2. Assignment of the weighting factor.
3. Calculation of the product maturity score.

	Phase 1: Assess Maturity Element				Phase 2	Phase 3
	Define Requirements	Locate Resources	Assess Element Maturity	Assign Element Score	Assign Weighting Factor	Calculate Product Maturity Score
Product Software						
Support						
Documentation						
Training						
Product Integration						
Professional Services						

Figure 3.7 ***The OSMM three-phase evaluation process***
(Golden, 2004)

The first phase consists of the following steps:

1. Define the requirements,
2. Locate the resources,
3. Assess the element maturity
4. Assign the element score.

The second phase involves assigning the objective weighting factors that are provided as default weightings and that can be changed by individual organizations to reflect their particular needs (Golden, 2004).

The last phase involves calculating the product maturity score by multiplying the score of each element by its weight, and then summing the results to obtain the output of the OSMM assessment as a numeric score between zero and 100. This score may be compared against recommended levels for different purposes, which vary according to whether an organization is an early adopter or a pragmatic user of information technology (Golden, 2004), see Table 3.1 for the recommended minimum OSMM scores.

Table 3.1

Recommended minimum OSMM scores
(Golden, 2004)

Purpose of Use	Type of User	
	Early Adopter	Pragmatist
Experimentation	25	40
Pilot	40	60
Production	60	70

Using the key software concept of maturity (i.e., how far along a product is in the software lifecycle, which dictates what type of use may be made of the product), the OSMM assesses the maturity level of the following key product elements (Golden, 2004):

1. Software.
2. Support.
3. Documentation.
4. Training.
5. Product integration.
6. Professional Services.

The OSMM is designed to be a lightweight process which can evaluate an open-source product's maturity in two weeks or less (Golden, 2004).

3.3.2 Software product maturity model

In addition to the OSMM, Nastro (1997) developed a maturity model for the software product. His maturity model consists of three core elements and two sub-elements, the sub-elements may be applied to specific software applications (Nastro, 1997).

The core elements of Nastro's (1997) model are the following:

1. Product capability.
2. Product stability.
3. Product maintainability

And the sub-elements are:

1. Product repeatability.
2. Product compatibility.

Based on the computed maturity level of each of the core and sub-elements, Nastro (1997) proposed the following equation to calculate the product maturity level of an embedded, real-time or signal processing system (Nastro, 1997):

$$\boxed{PC * (PS + PR + PM) / 3} \tag{3.1}$$

where:

- *PC* is the Product Capability maturity level,
- *PS* is the Product Stability maturity level,
- *PR* is the Product Repeatability maturity level, and
- *PM* is the Product Maintainability maturity level.

In the above equation, *PC* has the highest weight among all the core and sub-elements, because of its criticality for this application (Nastro, 1997).

3.4 Identification of the strengths and weaknesses

In this thesis, we are interested in the development of maturity models to assess the quality of software product rather than the process. Therefore, in this section, we focus on the strengths and weakness of the two product-related maturity models; we did not find any other maturity models related to the software product.

3.4.1 OSSM model

When the OSMM is used to assess the maturity of open source software, it takes into account other elements rather than only the software product itself, that is: the support, documentation, training, product integration and professional services. Indeed, these elements can affect the quality of the software product since – for example – a high quality software with low quality documentation might be interpreted as a poor quality software product from the customer point of view.

The OSMM is designed to be used with the open source software products when they are completed, i.e. they are ready for the release. In addition, it is mostly useful when an organization or an individual needs to choose between a variety of open source software products. Furthermore, it is not based on any quality model.

3.4.2 Nastro model

The second product maturity model – the Nastro software product maturity model – has the following limitations:

1. It is for an executable software product. Therefore, it can only be used with an incremental life-cycle which provides multiple releases (versions) of an executable software product.
2. It is not based on any comprehensive quality model, but only on a small number of product quality characteristics (there are five of them).
3. It is designed for the software product itself, rather than the quality of the software product.
4. For each element (core or sub-), there is only one measure.
5. It has been built to track and report the software development effort during an incremental life-cycle.

CHAPTER 4

RELATED CONCEPTS TO THE DEVELOPMENT OF THE SOFTWARE PRODUCT QUALITY MATURITY MODEL - SPQ^{MM}

4.1 Introduction

In this chapter, we introduce the concepts which will be used to build the quality maturity model. In particular, we presents the selected software quality model, the sigma concepts, ISO 15026 on software integrity levels, and IEEE Std. 1012 on software verification and validation.

This chapter is structured as follows: Section 4.2 presents five quality models from the software engineering literature. Section 4.3 introduces the sigma concepts. Section 4.4 gives a general description of the contents of the ISO 15026 on software integrity levels. Finally, Section 4.5 presents some information about the IEEE Std. 1012 on the software verification and validation.

4.2 Software quality models

There are a number of quality models in software engineering literature, each one of these quality models consists of a number of quality characteristics (or factors, as called in some models). These quality characteristics could be used to evaluate the quality of the software product from the view of that characteristic. Selecting which one of the quality models to use is a real challenge. In this subsection, we present the contents of the following quality models:

1. McCall's Quality Mode.
2. Boehm's Quality Model.

3. Dromey's Quality Model.
4. FURPS Quality Model.
5. ISO 9126 Quality Model.

4.2.1 McCall's quality mode

McCall quality model (also known as the General Electrics Model of 1977) is one of the most quoted quality models in the software engineering literature. It has been presented in 1977 by Jim McCall *et al.* (1977). This model originates from the US military and is primarily aimed towards the system developers and the system development process (McCall, Richards and Walters, 1977). Using this model, McCall attempts to bridge the gap between users and developers by focusing on a number of software quality factors that reflect both the users' views and the developers' priorities (McCall, Richards and Walters, 1977).

The structure of the McCall quality model consists of three major perspectives (types of quality characteristics) for defining and identifying the quality of a software product, and each of these major perspectives consists of a number of quality factors. Each of these quality factors has a set of quality criteria, and each quality criteria could be reflected by one ore more metrics, as in Figure 4.1. The three major perspectives and their related factors are the following (McCall, Richards and Walters, 1977):

1. Product Revision: it is about the ability of the product to undergo changes, and it includes:
    a. Maintainability: the effort required to locate and fix a fault in the program within its operating environment.
    b. Flexibility: the ease of making changes required by changes in the operating environment.
    c. Testability: the ease of testing the program, to ensure that it is error-free and meets its specification.

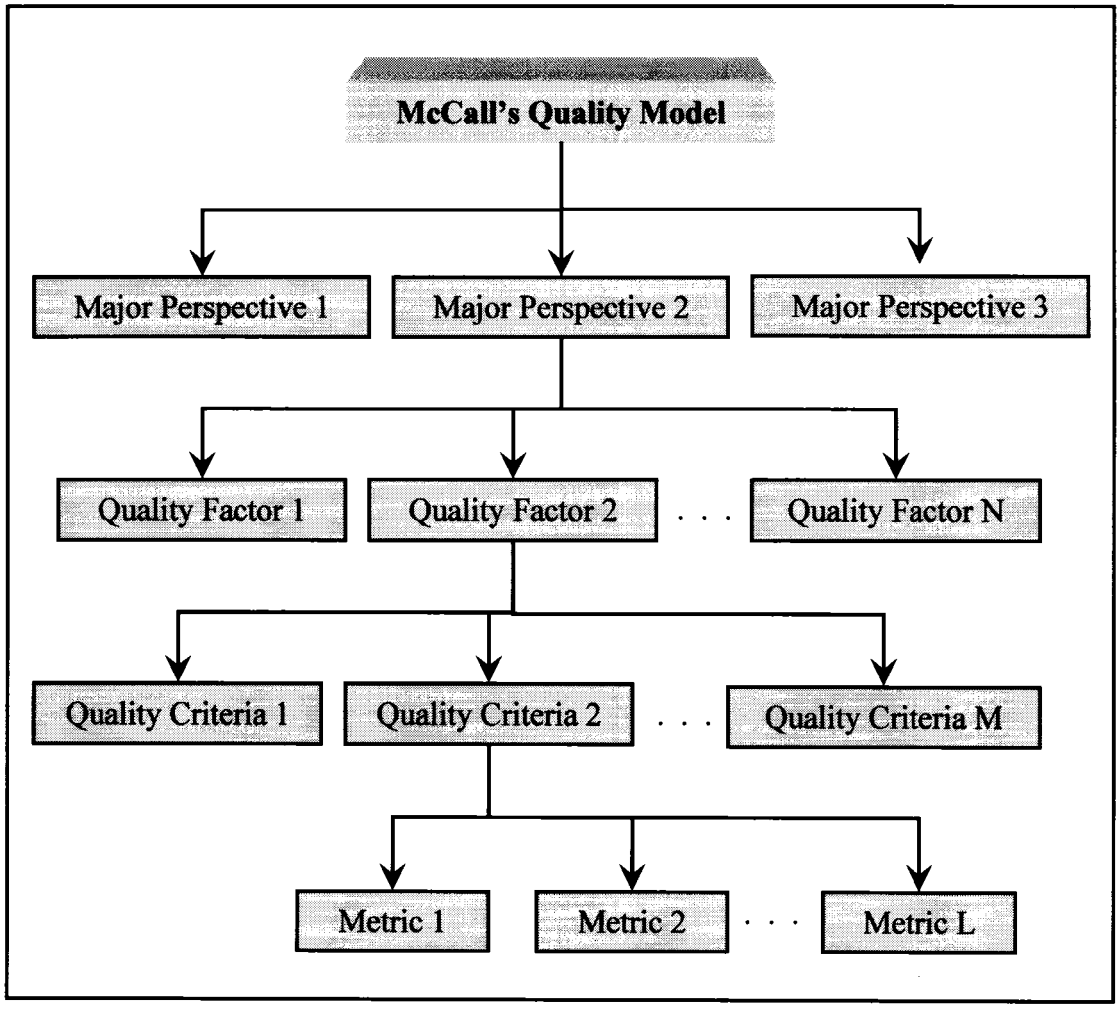

Figure 4.1 ***The structure of McCall's quality model***

2. Product Operations: it is about the characteristics of the product operation. The quality of the product operations depends on:
   a. Correctness: the extent to which a program fulfils its specification.
   b. Reliability: the systems ability not to fail.
   c. Efficiency: it is further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage.
   d. Integrity: the protection of the program from unauthorized access.
   e. Usability: the ease of the use of the software.

3. Product Transition: it is about the adaptability of the product to new environments. It is all about:
    a. Portability: the effort required to transfer a program from one environment to another.
    b. Reusability: the ease of reusing software in a different context.
    c. Interoperability: the effort required to couple the system to another system.

In more details, McCall's quality model consists of 11 quality factors to describe the external view of the software (from the users' view), 23 quality criteria to describe the internal view of the software (from the developer's view) and a set of 'metrics' which are defined and used to provide a scale and method for measurement. Table 4.1 presents the three major perspectives and their corresponding quality factors and quality criteria.

The main objective of the McCall's quality model is that the quality factors structure should provide a complete software quality view (Kitchenham and Pfleeger, 1996). The actual quality metric is computed by answering 'yes' and "no" questions.

4.2.2 Boehm's quality model

Boehm's quality model (Boehm et al., 1978; Boehm, Brown and Lipow, 1976) is introduced to quantitatively evaluate the quality of software. This model attempts to qualitatively define the quality of software by a predefined set of attributes and metrics. It consists of high-level characteristics, intermediate-level characteristics and lowest-level (primitive) characteristics which contribute to the overall quality level (see Figure 4.2).

Table 4.1

The content of McCall's quality model

Major Perspectives	Quality Factors	Quality Criteria
Product revision	Maintainability	Simplicity
		Conciseness
		Self-descriptiveness
		Modularity
	Flexibility	Self-descriptiveness
		Expandability
		Generality
	Testability	Simplicity
		Instrumentation
		Self-descriptiveness
		Modularity
Product operations	Correctness	Traceability
		Completeness
		Consistency
	Efficiency	Execution efficiency
		Storage efficiency
	Reliability	Consistency
		Accuracy
		Error tolerance
	Integrity	Access control
		Access audit
	Usability	Operability
		Training
		Communicativeness
Product transition	Portability	Self-descriptiveness
		Software-system independence
		Machine independence
	Reusability	Self-descriptiveness
		Generality
		Modularity
		Software-system independence
		Machine independence
	Interoperability	Modularity
		Communication commonality
		Data commonality
3 Perspectives	11 Factors	23 Distinct Criteria

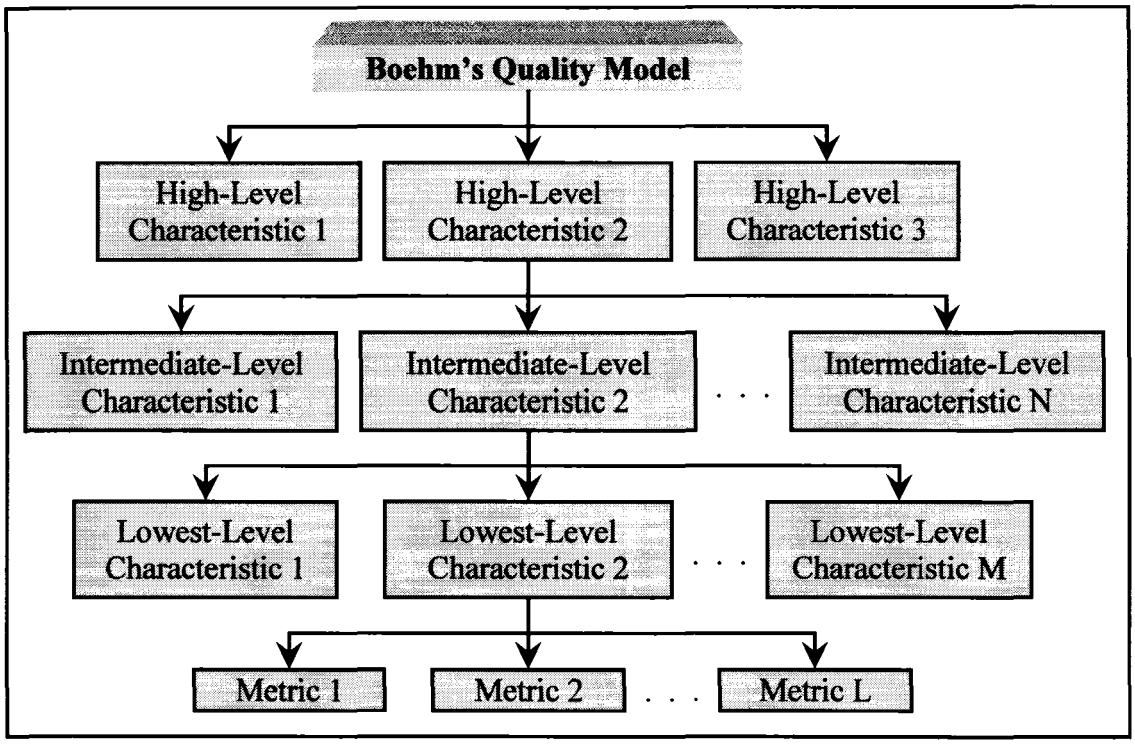

Figure 4.2 ***The structure of Boehm's quality model***

In this model, the high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put. In its high-level, there are three characteristics, that is (Boehm et al., 1978; Boehm, Brown and Lipow, 1976):

- As-is utility: to address how well, easily, reliably and efficiently can I use the software product as-is?
- Maintainability: to address how easy is it to understand, modify and retest the software product?
- Portability: to address if the software product can still be used when the environment has been changed?

Table 4.2 shows the contents of the Boehm's quality model in the three levels, high-level, intermediate-level and lowest-level characteristics. In addition, it is noted that there is a number of the lowest-level characteristics which can be related to more than

one intermediate-level characteristics, for example, the 'Self Containedness' primitive characteristic could be related to the 'reliability' and 'portability' primitive characteristics.

Table 4.2

The content of Boehm's quality model

High-Level Characteristics	Intermediate-Level Characteristics	Primitive Characteristics
As-is Utility	Reliability	Self Containedness
		Accuracy
		Completeness
		Robustness/Integrity
		Consistency
	Efficiency	Accountability
		Device Efficiency
		Accessibility
	Human Engineering	Robustness/Integrity
		Accessibility
		Communicativeness
Portability		Device Independence
		Self Containedness
Maintainability	Testability	Accountability
		Communicativeness
		Self Descriptiveness
		Structuredness
	Understandability	Consistency
		Structuredness
		Conciseness
		Legibility
	Modifiability	Structuredness
		Augmentability
3 High-Level Characteristics	7 Intermediate-Level Characteristics	15 Distinct Primitive Characteristics

In the intermediate level characteristic, there are seven quality characteristics that together represent the qualities anticipated from a software system (Boehm et al., 1978; Boehm, Brown and Lipow, 1976):

- Portability: the software can be operated easily and well on computer configurations other than its current one.
- Reliability: the software can be expected to perform its intended functions satisfactorily.
- Efficiency: the software fulfills its purpose without waste of resources.
- Usability: the software is reliable, efficient and human-engineered.
- Testability: the software facilitates the establishment of verification criteria and supports evaluation of its performance.
- Understandability: the software purpose is clear to the inspector.
- Flexibility: the software facilitates the incorporation of changes, once the nature of the desired change has been determined.

The primitive characteristics can be used to provide the foundation for defining quality metrics; this use is one of the most important goals established by Boehm when he has constructed his quality model; one or more metrics are supposed to measure a given primitive characteristic. Boehm *et al.* (1978, p. 13) defined the 'metric' as "a measure of extent or degree to which a product possesses and exhibits a certain (quality) characteristic."

4.2.3 Dromey's quality model

This quality model has been presented by Dromey (1995; 1996). It is a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to be applied to different systems (Dromey, 1995).

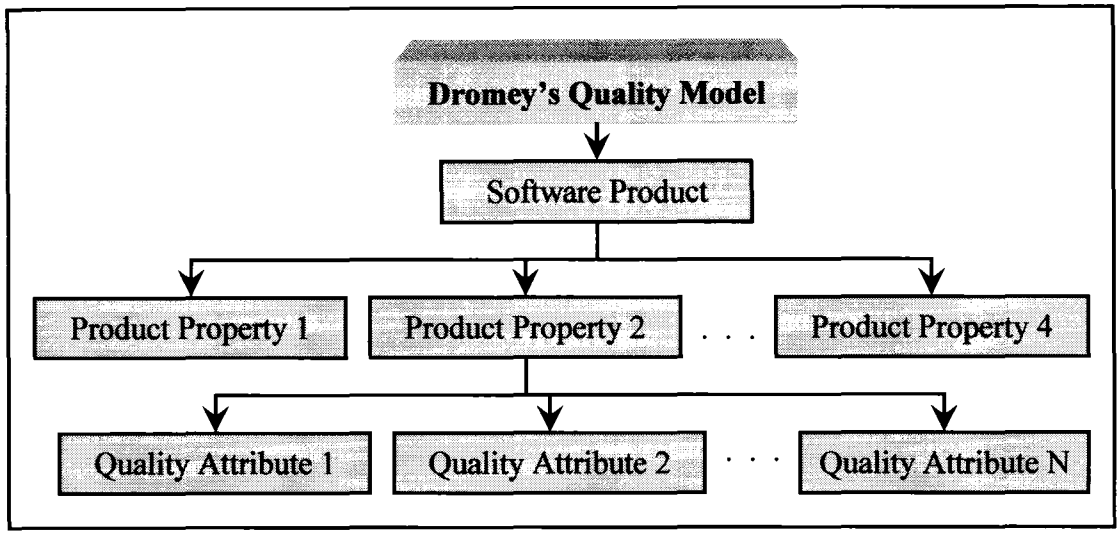

Figure 4.3 ***The structure of Dromey's quality model***

Furthermore, Figure 4.3 shows that it consists of four software product properties and for each property there is a number of quality attributes. Figure 4.4 shows the contents of the Dromey's quality model.

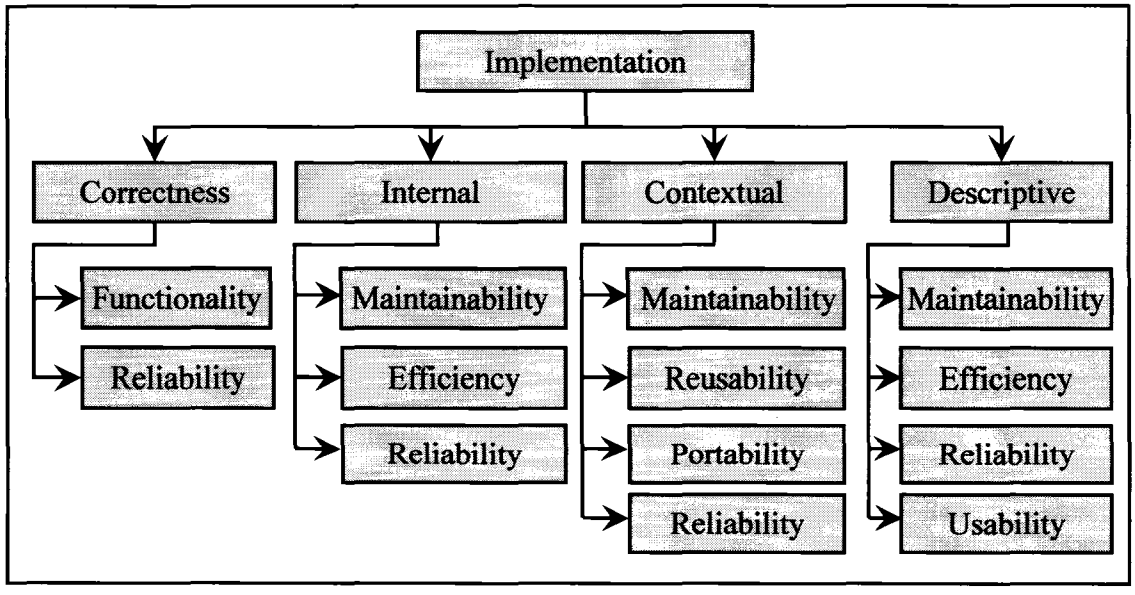

Figure 4.4 ***The content of Dromey's quality model***

4.2.4 FURPS quality model

The FURPS model was originally presented by Robert Grady (Grady, 1992); it has been later extended by IBM Rational Software (Jacobson, Booch and Rumbaugh, 1999; Kruchten, 2000) into FURPS+, where the '+' indicates such requirements as design constraints, implementation requirements, interface requirements and physical requirements (Jacobson, Booch and Rumbaugh, 1999). In this quality model, the FURPS stands for (Grady, 1992), as in Figure 4.5:

- Functionality: it may include feature sets, capabilities, and security.
- Usability: it may include human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials.
- Reliability: it may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
- Performance: it imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage.
- Supportability: it may include testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability.

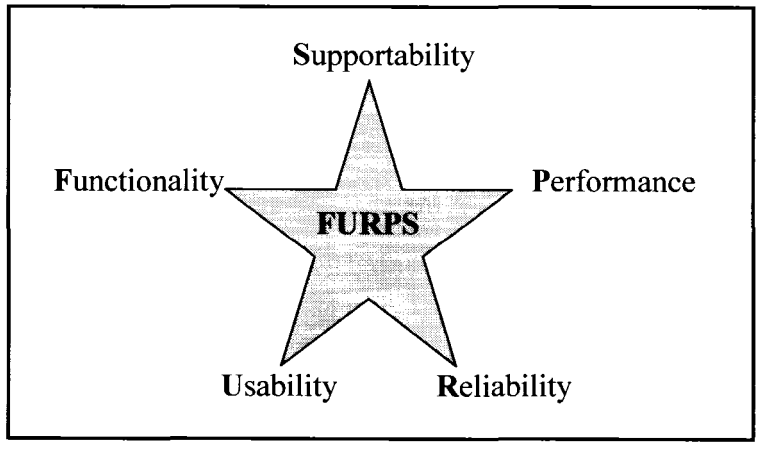

Figure 4.5 ***The contents of FURPS quality model***

4.2.5 ISO 9126 quality model

As mentioned in Chapter 2, the first document of the ISO 9126 series – quality model – contains two-parts quality model for software product quality (ISO, 2001b):

1. Internal and external quality model.
2. Quality in-use model.

Each internal and external quality characteristic and its corresponding subcharacteristics are defined in ISO 9126-1 (ISO, 2001b, p. 7) as follows:

- Functionality: "the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions". It contains the following subcharacteristics:
  - Suitability: "the capability of the software product to provide an appropriate set of functions for specified tasks and user objectives".
  - Accuracy: "the capability of the software product to provide the right or agreed results or effects with the needed degree of precision".
  - Security: "the capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them".
  - Interoperability: "the capability of the software product to interact with one or more specified systems".
  - Functionality Compliance: "the capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality".
- Reliability: "The capability of the software product to maintain a specified level of performance when used under specified conditions". It includes the following subcharacteristics:
  - Maturity: "the capability of the software product to avoid failure as a result of faults in the software".

- Fault tolerance: "the capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface".
- Recoverability: "the capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure".
- Reliability Compliance: "the capability of the software product to adhere to standards, conventions or regulations relating to reliability".

- Usability: "the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". It contains the following subcharacteristics:
  - Understandability: "the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use".
  - Learnability: "the capability of the software product to enable the user to learn its application".
  - Operability: "the capability of the software product to enable the user to operate and control it".
  - Attractiveness: "the capability of the software product to be attractive to the user".
  - Usability Compliance: "the capability of the software product to adhere to standards, conventions, style guides or regulations relating to usability".
- Efficiency: "the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions". It includes the following subcharacteristics:
  - Time behaviour: "the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions".

- Resource behaviour: "the capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions".
- Efficiency Compliance: "the capability of the software product to adhere to standards or conventions relating to efficiency".

- Maintainability: "the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications". It contains the following subcharacteristics:
  - Analyzability: "the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified".
  - Changeability: "the capability of the software product to enable a specified modification to be implemented".
  - Stability: "the capability of the software product to avoid unexpected effects from modifications of the software".
  - Testability: "the capability of the software product to enable modified software to be validated".
  - Maintainability Compliance: "the capability of the software product to adhere to standards or conventions relating to maintainability".
- Portability: "the capability of the software product to be transferred from one environment to another". It includes the following subcharacteristics:
  - Adaptability: "the capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered".
  - Installability: "the capability of the software product to be installed in a specified environment".
  - Co-existence: "the capability of the software product to co-exist with other independent software in a common environment sharing common resources".

- Replaceability: "the capability of the software product to be used in place of another specified software product for the same purpose in the same environment".
- Portability Compliance: "the capability of the software product to adhere to standards or conventions relating to portability".

Furthermore, each of the quality in-use characteristic is also defined in ISO 9126-1 as follows (ISO, 2001b, p. 12):

- Effectiveness: it is "the capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use."
- Productivity: it is "the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use."
- Safety: it is "the capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use."
- Satisfaction: it is "the capability of the software product to satisfy users in a specified context of use."

4.3 Sigma concepts

4.3.1 What is six-sigma?

The six-sigma (6-σ) approach has been designed by Motorola Company in 1986 and defined as a measure of the defects to improve the quality (Motorola, 2007). The term six-sigma has its origins in statistical process control, and recently, it has become better known as the name of a wide ranging set of data based process improvement techniques (Shelley, 2003). In more details, six-sigma means six standard deviations. A standard deviation is a parameter that characterizes a set of measurements, just as the average can

characterize such a set. One standard deviation is a value such that roughly two thirds of all values in a set fall within the range from one standard deviation below average to one standard deviation above average. Sets of values that can be characterized by the average and standard deviation can be modeled by the normal distribution, also known as the 'bell-shaped curve'. With a larger coefficient for sigma (1-σ, 2-σ, 3-σ, 4-σ, 5-σ, or 6-σ) more of the set is included, corresponding to a larger area under the bell curve (Breyfogle, 2003). Moreover, six-sigma is about measuring defects in a value chain process in order to systematically reduce them and, therefore, the corresponding cost factors; where a process that executes on 6σ level will yield results within the tolerance interval with 99.99966% probability, this means that we could have only 3.4 defects per one million defect opportunities (DPMO) (Fehlmann, 2004).

In the literature, there are many definitions of six-sigma. For example, Brue (2005) defines it as a method for improving productivity and profitability; and it is a disciplined application of statistical problem solving tools to identify and quantify waste and indicate steps for improvement. In other words, six-sigma focuses on executive sponsorship, driving out defects, and measurable improvements (Pickerill, 2005). While, Hefner and Sturgeon (2002, p. 4) define it as "a management philosophy based on meeting business objectives by striving for perfection; and it is a disciplined data-driven methodology for decision making and process improvement". In addition, they identified the content of the six-sigma to be five integrated methods, that is:

1. Process management.
2. Voice of the customer.
3. Change management.
4. Tools for measuring variation and change.
5. Business metrics.

Kumar (2005, p. 1) defines six-sigma as "a compelling method for breakthrough improvements for delivering world class processes with a defect rate of less than 3.4

parts per million". Whereas, Schofield (2006) defines it as a disciplined, structured, and data-driven methodology to solving problems.

However, Motorola (2004) identified six reasons why leaders love six-sigma, that is:

1. It impacts the bottom line:
2. It drives strategy execution.
3. It generates robust and flexible business processes.
4. It improves human performance across the enterprise.
5. It is highly scalable.
6. It is a low risk investment.

Design for six-sigma (DFSS) is a rigorous approach to designing products and services to meet customer expectations. Companies implementing six-sigma find that many defects are actually created during the design process. DFSS facilitates redesign of processes and ensures that end products are producible using existing technology. In addition, DFSS makes engineering or process designer aware of the need for concurrent product and process design, thereby eliminating defects before they can occur. In addition, DFSS can be seen as a subset of six-sigma focusing on preventing problems, instead of just fixing them (Mazur, 2003).

4.3.2 Six-sigma in software engineering

Six-sigma concepts are entering software engineering literature and practice (Biehl, 2004) and software practitioners and researchers are exploring ways to apply six-sigma techniques to improve software and systems development (Heinz, 2004). The essence of six-sigma for software is to prevent software from producing defectiveness in spite of their defects rather than to build software without defects (Biehl, 2004).

Since six-sigma is new in the software and systems development domains, many organizations are working hard to implement it; common issues raised by organizations include (Heinz, 2004):

- How does six-sigma compare with other improvement approaches, and how does it fit with my organization's other software process improvement initiatives?
- What evidence is there that six-sigma is applicable to software and systems engineering?
- What will it take for me to implement six-sigma in my organization, and how do I get started?
- How do I train software engineers in six-sigma methods when six-sigma training is largely focused on manufacturing?

In software engineering, controls can be implemented to take advantages of the improvement zone between 3σ and 6σ process performance; this can be done by building critical customers measures into software solutions, for example, response times, cycle times, transaction rates, access frequencies, and user defined thresholds. Then, these measures can make the applications self-correcting by enabling specific actions when process defects surface within the improvement zone; these actions do not always need sophisticated technical solutions to be beneficial (Biehl, 2004). Controls can be as simple as an email notifying support personnel of defects above the 3σ level or a periodic report-highlighting activity in the 3σ to 6σ zone (Biehl, 2004).

Fehlmann (2004) identifies a six-sigma approach (DMAIC) to software development (by software development, he does not mean only writing new software, but also software integration, deployment, and maintenance), the DMAIC stands for:

1. Define: Set the goal.
2. Measure: Define the measures.
3. Analyze: Measure where you go.

4. Improve: Improve your processes while you go.
5. Control: Act immediately if going the wrong path.

In addition, Fehlmann (2004) mentions three principles based on the experience of implementing six-sigma for software:

- Principle 1: Evaluate customer related measures only (use combinatory metrics to cover all topics).
- Principle 2: Adjust to moving targets (your goals may need change; accept change and manage it accordingly).
- Principle 3: Enforce measurement (do not enforce meeting targets).

Siviy and Forrester (2004) conducted a research project to investigate the use of six-sigma to accelerate the adoption of CMMI. They concluded the following:

- Six-sigma helps integrate multiple improvement approaches to create a seamless, single solution.
- Rollouts of process improvement by six-sigma adopters are mission-focused as well as flexible and adaptive to changing organizational and technical situations.
- Six-sigma is frequently used as a mechanism to help sustain (and sometimes improve) performance in the midst of reorganizations and organizational acquisitions.
- Six-sigma adopters have a high comfort level with a variety of measurement and analysis methods.
- Six-sigma can accelerate the transition of CMMI:
    - Moving from CMMI Maturity Level (ML) three to five in nine months, or from CMMI-SW ML one to five in three years (the typical move taking 12-18 months per level)
    - Underlying reasons are strategic and tactical

- When six-sigma is used in an enabling, accelerating, or integrating capacity for improvement technologies, adopters report quantitative performance benefits, using measures they know are meaningful for their organizations and clients.

VanHilst *et al.* (2005) proposed that the Global software Development Environments (GDEs) can be extended with a DMAIC framework (methodology) to interactively provide required metrics and analyses.

In the course of its work with hundreds of six-sigma companies and through its own experience, Microsoft has identified six key steps to ensure six-sigma success (Microsoft, 2006), that is:

1. Establish Leadership Support and Engagement.
2. Align Goals with six-sigma Activities.
3. Establish six-sigma Infrastructures.
4. Identify Opportunities to Improve.
5. Match People with Projects.
6. Ensure Execution and Accountability.

Raytheon started its six-sigma (also called R6S) in early 1999; as a result of their process improvement, they reported $1.8 billion in gross financial benefits, $500 million improvement in operating profit, and generated $865 million in cash flow (Hayes, 2003). This is an example of what other organizations got by implementing six-sigma to software process improvement (Hayes, 2003).

Furthermore, Head (1994) used the cleanroom software engineering technique as a methodology to implement six-sigma to software in order to produce six-sigma quality software.

4.3.3 Uncertainty of applying six-sigma in software engineering

The success stories of applying six-sigma in manufacturing during the last years have encouraged the practitioners and researchers to explore the applicability of six-sigma in software industry. Based on the results of this exploration, the researchers have been divided into two groups, that is, for or against this idea. In particular, some researchers came up with a number of uncertainties on applying six-sigma to software. Throughout this subsection, we present some of these uncertainties.

Binder (1997; 2001) stated that six-sigma does not make sense for software based on the following reasons:

1. Software processes are fuzzy. Every part of the software is produced by a process that defies the predictable mechanization assumed for physical parts.
2. Software characteristics of merit cannot be expressed accurately as ordinal tolerance; that which makes software correct or incorrect cannot usually be measured as simple distances, weights, or other physical units.
3. Software is not mass produced. Even if software components could be designed to ordinal tolerance, they would still be one-off artifacts.

The six-sigma software process could be interpreted as 3.4 failures per million lines of code, that is, 0.0034 failures per thousand lines of code; this would require a software process which is twice better than the current best practices (Binder, 2001). It is difficult to imagine how this could be achieved since the average cost of such low rate of failures in the source code is reported to be $1000 per line (Joyce, 1989).

Jacowski (2006) mentioned that the big question is whether six-sigma can indeed be applied in the software industry as successfully as it was applied to manufacturing; this is still being argued. The real challenge is to see if it can be implemented to the software

process without reinventing the wheel. There is also disagreement among leaders in the software industry about the need for six-sigma.

In addition, a software development process - which is defined as a set of software engineering activities to transform user requirements into software product – is completely different than other types of processes such as manufacturing. The distinctiveness attributes for a software development process which are not available for other types of processes are as follows (Hong and Goh, 2003):

1. Unlike other types of processes such as manufacturing, the software development process is not repetitiveness. However, each software product needs its own process. In addition, the software process produces one software product which could be duplicated with high precision. Then, once the software is duplicated, it produces the exact functionality as the original copy.
2. The inputs and outputs of the software development process are different in each instance of the process. It does not make sense to produce exactly the same piece of software twice. Therefore each instance of software process deals with one different set of user requirements, and outputs of different software modules form part of the final software product.
3. In contrast to a manufacturing process, each transformation of a user requirement to a software module is cognition intensive; while, most of the manufacturing activities are targeted to minimize cognition.
4. As a result, software development is an intellectual process that needs visualization (e.g., documentation) before six-sigma implementation (Card, 2000). In software development, data relationships can be discovered via documentations, interviews and process mappings. Data flow diagrams, entity relationship diagrams, and object models are tools commonly used to represent the data relationship that the six-sigma approach needs for problem definition.
5. Software development is an intellectual process that needs visualization (e.g., documentation) before six-sigma implementation

6. Different sets of external factors affect the software development process, such as changes of developers, knowledge level, programming skills, and so on. This is unlike the manufacturing process which is affected by many sources of variation such as temperature, raw materials, equipment wear, and human interaction; these factors are hardly valid for software process.

Moreover, Hong and Goh (2003) stated that a misuse of six-sigma at the other extreme would be emphasizing the unique features of software process and never attempt to manage and improve the existing process. Some software engineering activities, such as the process life cycle models and quality measurements, are evidence of effort towards producing stable software processes.

4.4 ISO 15026: software integrity levels

The ISO 15026 international standard on system and software integrity levels establishes the requirements for determining the level of system and software integrity. By identifying the requirements for determining the system and software level, the software integrity requirements can be determined (ISO, 1998b). This international standard defines the concepts associated with integrity levels, defines the processes for determining integrity level requirements and imposes requirements on each process (ISO, 1998b). It is a standard that can only be applied to software. Moreover, the integrity level of the system and the integrity levels of the hardware components are only required in this international standard to determine the integrity levels of the software components (ISO, 1998b).

The software integrity level is an assignment of either a degree of reliability of providing a mitigating function, or a limit on the frequency of failure that could result in a threat, that is, the degree of confidence that the overall system will not fail (ISO, 1998b). In

addition, a software integrity level refers to a range of values of a software property necessary to maintain system risks within acceptable limits (ISO, 1998b).

More specifically, this ISO standard provides an example of a risk matrix (see Table 4.3) which could be used to calculate the risk associated with each threat. This calculation relates the frequency of occurrence of an initiating event to the severity of the consequences of that initiating event. The result will be the risk class, which could be one of the following: high, intermediate, low or trivial (ISO, 1998b). Using this result, the system integrity level could be determined by mapping the risk class to the corresponding system integrity level using Table 4.4.

Table 4.3

Example of a risk matrix
(ISO, 1998b)

Frequency of Occurrence	**Indicative Frequency (per year)**	**Severity of Consequences**			
		Catastrophic	**Major**	**Severe**	**Minor**
Frequent	**> 1**	High	High	High	Intermediate
Probable	**$1 - 10^{-1}$**	High	High	Intermediate	Low
Occasional	**$10^{-1} - 10^{-2}$**	High	High	Low	Low
Remote	**$10^{-2} - 10^{-4}$**	High	High	Low	Low
Improbable	**$10^{-4} - 10^{-6}$**	High	Intermediate	Low	Trivial
Incredible	**$< 10^{-6}$**	Intermediate	Intermediate	Trivial	Trivial

Table 4.4

Mapping risk class to its integrity level
(ISO, 1998b)

Risk Class	Integrity Level
High	A
Intermediate	B
Low	C
Trivial	D

In addition, a software integrity level refers to a range of values of a software property necessary to maintain system risks within acceptable limits (ISO, 1998b).

The software integrity level represents a portion of the system integrity level, and this portion is associated with a subsystem consisting of software as a component or of software only. According to the ISO, the software integrity level for a subsystem and the overall system integrity level shall be identical (ISO, 1998b). However, there are a number of assumptions to be taken into account when determining the integrity level of the software, as follows (ISO, 1998b):

1. For the system, there exists a system integrity level assignment.
2. The architectural features of the system should be defined.
3. The inputs include:
   - the system integrity level;
   - a list of threats, and for each threat:
   - the initiating events that may lead to the threat, and
   - the expected frequency or probability of occurrence of each initiating event;
   - a system architecture definition in sufficient detail.
4. The output is the software integrity level.

4.5 IEEE Std. 1012: software verification and validation

To determine the criticality of the software, the IEEE Standard for Software Verification and Validation – IEEE Std. 1012 (IEEE, 1998) – defines four software integrity levels which vary from high integrity to low integrity.

Software products have different criticalities, and these are based on their proposed use and whether the system will be applied to critical or non-critical uses. Some software systems support critical, life-sustaining systems, while others do not. Software criticality is a description of the intended use and application of a system. Software integrity levels denote a range of software criticality values necessary to maintain risks within acceptable limits (IEEE, 1998).

In this IEEE standard, the assignment of the integrity level of any particular software product will be completely based on the error consequences and their estimated occurrence (IEEE, 1998). Table 4.5 shows the assignment of the software integrity levels using the possible error consequences and their occurrences. For example, if the error consequences for specific software are critical and their occurrence is occasional, then the integrity level for this software will be three.

Table 4.5

Assignment of software integrity levels
(IEEE, 1998)

Error consequences	Likelihood of occurrence of an operating state that contributes to the error			
	Reasonable	Probable	Occasional	Infrequent
Catastrophic	4	4	4 or 3	3
Critical	4	4 or 3	3	2 or 1
Marginal	3	3 or 2	2 or 1	1
Negligible	2	2 or 1	1	1

4.6 Summary

In this chapter, we have introduced the concepts which will be used to build the software product quality maturity model (in Chapter 9 and 10). In particular, we have introduced the following:

1. Software quality models.
2. Sigma concepts.
3. ISO 15026 on software integrity levels.
4. IEEE Std. 1012 on the software verification and validation.

CHAPTER 5

RESEARCH OBJECTIVES AND METHODOLOGY

In this chapter, we present the objectives of this research and show the different research steps, which together will formulate the research methodology to reach the objectives.

5.1 Research objectives

The following are the objectives of this research:

1. Verification of the ISO 9126 measures against the metrology concepts. These measures will be an essential part of our quality maturity model.
2. Building a maturity model to assess the quality of the software product, that is, a maturity model for each of the ISO 9126 software product quality characteristics, life-cycle stages (internal, external and in-use) and the whole software product. Such a maturity model has to produce maturity levels for one or all the three points of view quality.

5.2 Research methodology

The research methodology is divided into two phases based on the three sections of Figure 2.9 in Chapter 2 (i.e. Data Collection, Data Preparation, and Data Analysis sections). Phase-A belongs to the 'Data Collection' and 'Data Preparation' section, while phase-B belongs to the 'Data Analysis' section. These two phases are described in this section.

5.2.1 Phase-A: Verification of software measures

Step 1: Verification of the analysis framework

In this step, we will verify the usefulness of applying the Habra *et al.* (2004) software measurement analysis framework – from the metrology perspective – on the software measurement, in particular, on the Halstead's measures as an example of the measures which do not meet key design criteria of measures in engineering and the physical sciences, and as some of his measures (i.e. program vocabulary and program length) have been mentioned in Annex C of the ISO 9126 parts 2, 3 and 4. After ensuring that the metrology concepts could be applied to analyse the Halstead's measures, we will go through to apply these concepts into the ISO 9126 measures, see next phase.

Step 2: Verification of the ISO 9126 measures against the metrology concepts and ISO 15939 International Standard

In this step, we will analyse the ISO 9126 measures against the metrology concepts and ISO 15939 (the ISO VIM metrology concepts have been adopted in the ISO 15939 on software measurement process); as a case study, we will analyse the quality in-use measures of the ISO 9126 part-4. This step is composed of the following sub-steps:

1. Identifying which metrology category could be applied to the ISO 9126-4 (quality in-use) measures.
2. Analysing the quality in-use measures based on the 'quantities and units' category contents.
3. Classifying the quality in-use measures into base and derived measures.
4. Identifying the dimensionless quantities (quantities of dimension one) of the derived measures.
5. Identifying the units of measurement for both the base and derived measures.
6. Drawing up suggestions for improvements.

Step 3: Building an ISO-based Information Model to address harmonization issues in the ISO 25020 and ISO 25021 standards

This step consists of the following sub-steps:

1. Identifying the harmonization issues arising from the new ISO 25020 and 25021 standards.
2. Analysing the three new terms in ISO 25020 and ISO 25021 (i.e. quality measures, quality measure elements and quality measure element categories).
3. Identifying the terms that are already exist in the metrology concepts and ISO 15939 to replace the ones introduced in the ISO 25020 and ISO 25021, that is: base and derived measures terms.
4. Based on the outcomes of step 2 above, we will classify the ISO 9126 parts 2, 3, and 4 measures into base and derived measures to align them with the ones introduced in ISO 25020 and ISO 25021 as quality measures and quality measure elements.
5. Identify the limited coverage of ISO quality models and their corresponding measures in the ISO 25020 and ISO 25021.
6. Based on the classification of the ISO 9126 parts 2, 3, and 4 measures, we will use the base measures to build a cross-reference table to facilitate which base measures need to be collected in order to measure a specific characteristic or subcharacteristic.
7. Drawing up a set of comments on the weaknesses of the new ISO 25020 and ISO 25021.
8. Building an information model based on ISO 15939 to address the ISO 25020 and ISO 25021 weaknesses.

5.2.2 Phase-B: Building a Software Product Quality Maturity Model – SPQ^{MM}

From the outcomes of Phase-A in which we have verified the ISO 9126 measures to get: a set of measures based on sound metrological foundations, industry consensus on base

measures for software product, and industry consensus on software product quality models. In addition, using a set of measures to represent – for example – a characteristic quality is not feasible for the decision makers; instead, making a decision based on a single value will be more efficient. To make this available, there is a need to build a maturity model to assess the quality of software product using the following steps:

1. Reviewing the literature to identify the related key references (see Figure 5.1) which can assist in the building of the proposed Software Product Quality Maturity Model (SPQ^{MM}).
2. Identifying the quality model to be used.
3. Identifying the contents of the quality maturity model based on the characteristics of the quality model which has been identified.
4. Constructing a set of five maturity levels.
5. Customizing the software Product failure consequences based on what is available in the ISO 15026 and IEEE Std. 1012 standards.
6. Customizing the software integrity levels to six levels (from zero to five as in ISO 15504) instead of four (as specified in the ISO 15026 and IEEE Std. 1012) to be aligned with the identified six failure consequences for the different software product types (e.g. embedded software, real-time software, application software, etc...)
7. Review the sigma concept in order to identify how it could be used with the quality levels, and customizing the sigma values with sigma shift.
8. Mapping the sigma and sigma shift concepts to the quality levels to produce a sigma range in order to facilitate the interpretation of the measured quality level.
9. Mapping the sigma ranges to the maturity levels.
10. Draw up a detailed formulas and procedures to be followed in order to get a quality maturity level.

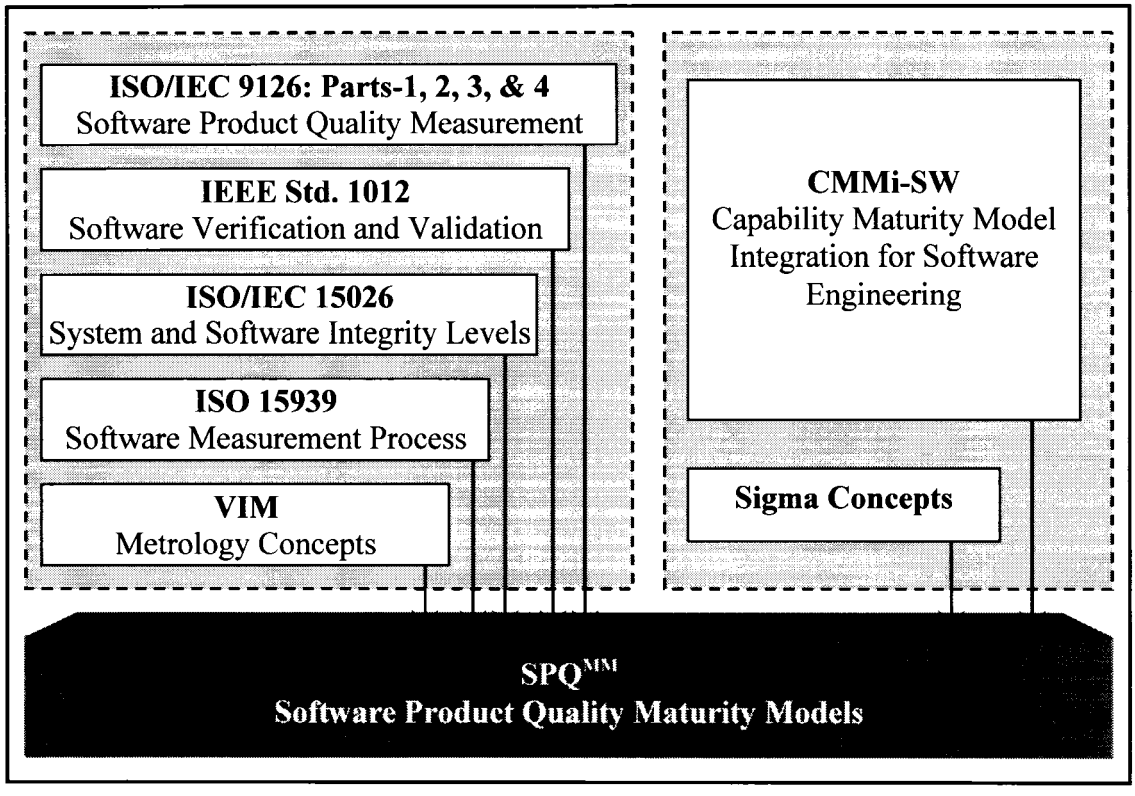

Figure 5.1 ***The related key references for building the SPQ^{MM}***

CHAPTER 6

AN ANALYSIS OF THE DESIGN AND DEFINITIONS OF HALSTEAD'S MEASURES

6.1 Introduction

A number of software measures widely used in the software industry are still not well understood (Abran, Lopez and Habra, 2004). Some of these measures were proposed over thirty years ago and, like many measures proposed later, they were defined mostly in an intuitive and heuristic manner by their designers. Moreover, authors describe their proposed measures in their own terms and structure since there is not yet a consensus on how to describe and document the design of a software measure. Of course, the lack of a common design approach has made it difficult for practitioners to assess these measures.

In 2004, Abran *et al.* (2004) revised the McCabe cyclomatic complexity number, illustrating that there is still ambiguity in its design and interpretation. In their study, Abran *et al.* (2004) used the software measurement analysis framework proposed in (Habra et al., 2004).

Halstead's measures – or what is commonly referred to collectively as 'software science' (Halstead, 1977) – are among the most widely quoted software measures. For example, researchers have used Halstead's measures to evaluate student programs (Leach, 1995) and query languages (Chuan et al., 1994), to measure software written for a real-time switching system (Bailey and Dingee, 1981), to measure functional programs (Booth and Jones, 1996), to incorporate software measurements into a compiler (Al Qutaish, 1998) and to measure open source software (Samoladas et al., 2004).

In this chapter, we investigate the usefulness of applying the software measurement analysis framework – which has been introduced in (Habra et al., 2004) – on various elements of the design and definitions of Halstead's measures.

We have selected the Habra *et al.* (2004) measurement framework among the other measurement frameworks because it has some of the metrology concepts. In contrast, the other measurement frameworks - such as Zuse (1998) and Morasca (1997) measurement frameworks - are limited to the measurement theory in its measurement aspects.

This chapter is structured as follows: section 6.2 presents a brief overview of the analysis framework used to analyze Halstead's measures. Section 6.3 presents an overview of Halstead's measures. Section 6.4 defines the context of Halstead's measures. In Section 6.5 the Halstead's measures design has been discussed. Finally, Section 6.6 contains a discussion on this analysis and a summary of our observations.

6.2 Analysis framework: an overview

Definitions of the terms that will be used in this chapter are provided first; these definitions have been adopted from ISO 15939 (ISO, 2002) and the ISO guide on international vocabulary of basic and general terms in metrology (ISO, 1993):

- Entity: is an object that is to be characterized by measuring its attributes (ISO, 2002).
- Attribute: is a property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means (ISO, 2002).
- Measurement method: is a logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale (ISO, 2002).
- Measurement procedure: is a set of operations, described specifically, used in the performance of a particular measurement according to a given method (ISO, 2002).

- Base measure: is a measure defined in terms of an attribute and the method for quantifying it (ISO, 2002).
- Derived measure: is a measure defined as a function of two or more values of base measures (ISO, 2002).
- Unit of measurement: is a scalar quantity defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitude (ISO, 1993).
- Scale: is an ordered set of values, continuous or discrete, or a set of categories to which the attribute is mapped (ISO, 2002).
- Scale type: it depends on the nature of the relationship between values on the scale. Nominal, ordinal, interval and ratio are the four types of scale defined and identified in ISO 15939 (ISO, 2002).

The analysis framework of measurement proposed by Habra *et al.* (2004) is based on work by Jacquet and Abran (1997). This analysis framework consists of four phases of the software measurement life cycle, as in Figure 6.1 (Habra et al., 2004):

1. Defining the context.
2. Designing the measurement.
3. Applying the measurement method.
4. Exploring the measurement results.

This measurement framework can be used to investigate and verify existing software measures. To analyze the design and definitions of Halstead's measures, we need to apply the first two phases of this analysis framework.

The two phases that will be used in this chapter are summarized here. The first phase is defining the context in order to state the goals of the measurement that need to be investigated in more detail. In this phase, we have to select the objectives of the

measurement in terms of the characteristics to be measured for a specific entity type (Habra et al., 2004).

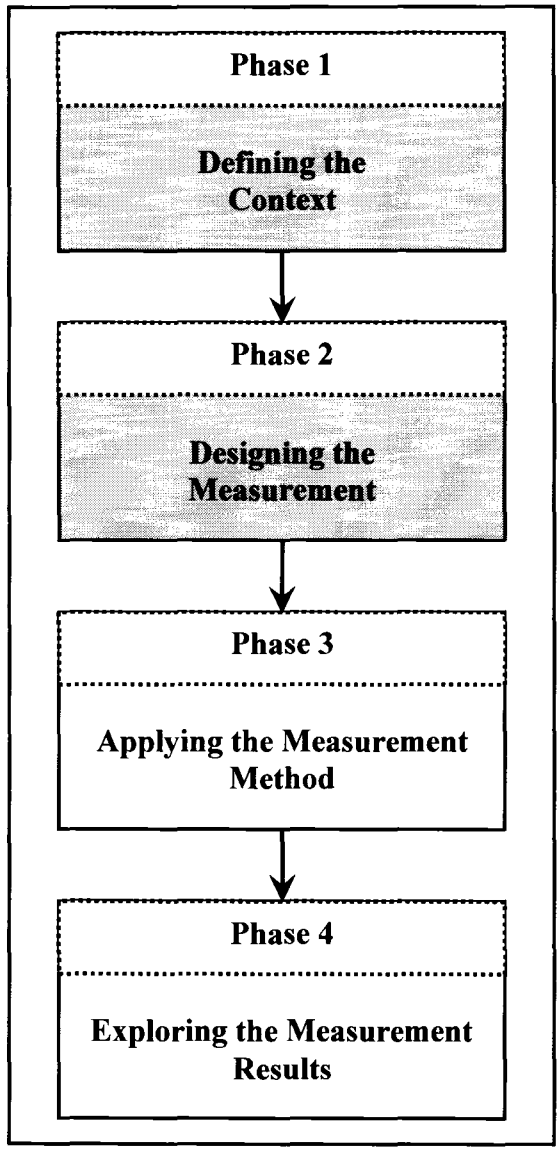

Figure 6.1 ***The four phases of the analysis framework of measurement*** *(Habra et al., 2004)*

The second phase, designing the measurement, is studied from three different points of view: activities, product and verification criteria. From the verification criteria viewpoint, this phase consists of three sub-phases (Habra et al., 2004):

1. The empirical and numerical worlds and their mapping:
   In order to define the empirical world, we need to determine the entities and their attributes to be measured. These attributes have to be ensured that they defined clearly and accurately, so that they are unambiguously characterized (Habra et al., 2004). Then – for the numerical world defined – the selected mathematical structure should conserve the properties of that empirical world. This means that the mapping between the mathematical structure and the empirical world must produce the same form (Habra et al., 2004).
2. The measurement method:
   Confirming and validating the numerical assignment rules (formulas) involve different activities, depending on the way those rules are expressed (Habra et al., 2004). These formulas will be used to produce measurement values for the attributes to be measured. In addition, the scale types of the measures and the units of measurement produced from the formulas based on the units of their operands have to be validated.
3. The measurement procedure:
   Verification of the measurement procedure to ensure that it constitutes a correct implementation of the measurement method. This verification should be achieved in accordance with the goals set out in the defining the context phase (Habra et al., 2004).

6.3 Halstead's measures: an overview

According to Halstead, a computer program is an implementation of an algorithm considered to be a collection of tokens that can be classified as either operators or operands. In other words, a program can be thought of as a sequence of operators and their associated operands. All Halstead's measures are functions of the counts of these tokens (Henry and Kafura, 1981). By counting the tokens and determining which are operators and which are operands based on a counting strategy, the following base measures can be collected (Halstead, 1977):

- *n1*: Number of distinct operators.
- *n2*: Number of distinct operands.
- *N1*: Total number of occurrences of operators.
- *N2*: Total number of occurrences of operands.

In addition to the above, Halstead (1977) defines:

- *n1**: Number of potential operators.
- *n2**: Number of potential operands.

Halstead refers to *n1** and *n2** as the minimum possible number of operators and operands for a module or a program respectively. This minimum number would occur in a programming language itself, in which the required operation already existed (for example, in C language, any program must contain at least the definition of the function *main()*), possibly as a function or as a procedure; in such a case, *n1**=2, since at least two operators must appear for any function or procedure: one for the name of the function and one to serve as an assignment or grouping symbol. Next, *n2** represents the number of parameters, without repetition, which would need to be passed on to the function or the procedure (Menzies et al., 2002).

All of the Halstead's so called 'Software Science' measures are defined based on the above collective measures (*n1*, *n2*, *N1*, *N2*, *n1** and *n2**).

Halstead defines the following measures (Halstead, 1977):

- The length (N) of a program P is:

$$N = N_1 + N_2 \tag{6.1}$$

- The vocabulary (n) of a program P is:

$$n = n_1 + n_2 \tag{6.2}$$

- Program volume (V) is defined by Halstead in his book as:
  a) "a suitable metric for the size of any implementation of any algorithm" (Halstead, 1977, p. 19);
  b) "a count of the number of mental comparisons required to generate a program" (Halstead, 1977, p. 49).

  V can be computed using the following equation:

$$V = N * \log_2 n \tag{6.3}$$

  The length, vocabulary and volume of a program are considered as reflecting different views of program size (Fenton, 1994).

- Program potential (minimal) volume (V^*), which is the volume of the minimal size (no objective evidence documented in (Halstead, 1977) that this is indeed a minimal implementation) implementation of a program P, is defined as:

$$V^* = (2 + n_2^*) \log_2 (2 + n_2^*) \tag{6.4}$$

- Program level (L) of a program P with volume V is:

$$L = \frac{V^*}{V} \tag{6.5}$$

The program level emphasizes that an increase in the volume leads to a lower level of program, and conversely. The largest value for L is one. In addition, this value is interpreted as referring to the most ideally written program and as measuring how well written a program is. Thus, programs with L values close to one are considered to be well written, in general $L<1$ (Chuan et al., 1994).

- Program difficulty (D) is defined as the inverse of program level L:

$$D = \frac{1}{L} \tag{6.6}$$

- The program level estimator ($\hat{L}$) of L is defined by Halstead as:

$$\hat{L} = \frac{2}{n_1} * \frac{n_2}{N_2} \tag{6.7}$$

and interpreted by Menzies *et al.* (2002) and by Fenton and Pfleeger (1997) as:

$$\hat{L} = \frac{1}{D} = \frac{2}{n_1} * \frac{n_2}{N_2} \tag{6.7-1}$$

- The intelligent content (I) of a program P is a measure of the information content of program P, and is defined as:

$$\boxed{I = \hat{L} * V} \tag{6.8}$$

- Programming effort (E) is a measure of the mental activity required to reduce a preconceived algorithm to a program P. The E is defined as the total number of elementary mental discriminations required to generate a program:

$$\boxed{E = \frac{V}{L} = \frac{n_1 N_2 N \log_2 n}{2 n_2}} \tag{6.9}$$

In the effort definition, the unit of measurement of E is claimed by Halstead to be an elementary mental discrimination.

- The required programming time (T) for a program P of effort E is defined as:

$$\boxed{T = \frac{E}{S} = \frac{n_1 N_2 N \log_2 n}{2 n_2 S}} \tag{6.10}$$

where S is the Stroud number (in 1967, a psychologist, John M. Stroud, suggested that the human mind is capable of making a limited number of mental discrimination per second (Stroud Number), in the range of five to 20 (Halstead, 1977)), defined as the number of elementary discriminations performed by the human brain per second. The S value for software scientists is set to 18 (Hamer and Frewin, 1982). The unit of measurement of T is the second.

All the above ten equations are based on the results of *n1*, *n2*, *N1*, *N2*, *n1** and *n2**, which themselves are based on a counting strategy to classify the program tokens as operators or operands.

Unfortunately, there is a problem in distinguishing between operators and operands. This problem occurs because Halstead has provided an example with specific illustrations of operators and operands, but without generic definitions applicable to any program context. That is, Halstead has not explicitly described the generic measurable concepts of operators and operands. He has asserted only that – in the example he provides – their description is intuitively obvious and requires no further explanation. In practice, for measurement purposes, intuition is insufficient to obtain accurate, repeatable and reproducible measurement results.

Therefore, it is important that the counting strategy should be clearly defined and consistent, since all Halstead's software science depends on counts of operators and operands (Lister, 1982). However, there is no general agreement among researchers on the most meaningful way to classify and count these tokens (Shen, Conte and Dunsmore, 1983). Hence, individual researchers (and practitioners as well) must state their own interpretation or, alternatively, use one of the available counting strategies proposed by other researchers, such as in (Abd Ghani and Hunter, 1996; Conte, Dunsmore and Shen, 1986; Salt, 1982; Szentes, 1986). Furthermore, Li *et al.* (2004) have proposed rules for identifying operators and operands in the object-oriented programming (OOP) languages.

Of course, it is to be expected that different counting strategies will produce different values of *n1*, *n2*, *N1* and *N2*, and, consequently, different values for the above ten equations.

6.4 Defining the context

The objective of Halstead's measures is to measure the following characteristics of a program: length, vocabulary, volume, level, difficulty and intelligence content. In addition, they are used to measure what is referred to as 'other characteristics' of the developer: programming effort and required programming time. All these measures are based only on the number of operators and the number of operands the given program or algorithm contains.

The last two attributes, which refer to a developer's attributes (programming effort and required programming time), seem to be identical, since 'effort to write a program' is similar to 'required programming time'.

6.5 Designing the measurement

6.5.1 The empirical and numerical worlds and their mapping

The entities that can be used to apply Halstead's measures are the source-code itself or the algorithm of that source code. However, applying Halstead's measures to these two entities will produce different values for the same base measures. For example, in Java language, the number of operators in the source code is different from the number of operators in the equivalent algorithm for that source code, since – as an example – in Java source code, each statement must be end with a semicolon (;), which is an operator.

Halstead's measures are based on two attributes: the number of operators and the number of operands. As mentioned in section 6.3, there is no agreement on how to distinguish between operators and operands. Therefore, different counting strategies will produce different numbers of operators and operands for the same program or algorithm. The two attributes can be easily mapped to a mathematical structure by counting the

number of operators and operands in the program source code or the equivalent algorithm.

Furthermore, Kiricenko and Ormandjieva (2005) investigated the validation of the representation condition for Halstead's program length measure.

6.5.2 The measurement method

To obtain a value for each of Halstead's measures, ten equations have to be computed (see section 6.3). It is to be noted that all of these equations (equations 6.1 to 6.10) correspond to a 'derived measure', as defined by the international vocabulary of basic and general terms in metrology (VIM) and the ISO 15939.

Equation (6.3) is of a ratio scale type, while equation (6.5) is of an ordinal scale type, as noted by Fenton and Pfleeger (1997). By contrast, Zuse (1998) maintains that equation (6.1) is of the ratio scale type and equations (6.2), (6.3), (6.6) and (6.9) are of an ordinal scale type. Moreover, it can be observed that equation (6.4) is also of the ratio scale type. However, it is not clear to which scale type equations (6.7), (6.8) and (6.10) belong.

These conclusions on the scale types of Halstead's measures need to be revisited when the units of measurement in Halstead's equations are taken into consideration.

For instance, in equation (6.1), the program length (N) is calculated by the addition of the total number of occurrences of operators and the total number of occurrences of operands. However, since their units are different, operators and operands cannot be directly added together unless the concept common to them (and its related unit) is taken into consideration in the addition of these numbers, that is, 'occurrences of tokens': then,

the right-hand side of equation (6.1) gives 'occurrences of tokens' as a measurement unit on the ratio scale:

$$N^{\text{occurrences of tokens}} = N_1^{\text{occurrences of operators}} + N_2^{\text{occurrences of operands}} \quad (6.11)$$

From equation (6.2), the program vocabulary (n) can be constructed by adding the number of distinct operators and the number of distinct operands:

$$n^{\text{distinct tokens}} = n_1^{\text{distinct operators}} + n_2^{\text{distinct operands}} \quad (6.12)$$

The measurement unit here is 'distinct tokens'. This measurement unit must then also be assigned to the left-hand side of this equation, labelled 'vocabulary', and associating it to the related concepts.

It can be noted that, while the concept of 'length' is associated with a number, the concept of 'vocabulary' is not. Indeed, the program vocabulary (n) reflects a different view of program size (Fenton, 1994), and it is a measure of 'the repertoire of elements that a programmer must deal with to implement the program' (Christensen, Fitsos and Smith, 1981). Most probably, an expression such as 'size of a vocabulary' would have been more appropriate.

From equation (6.3), program volume (V) has been interpreted with two different units of measurement; 'the number of bits required to code the program' (Hamer and Frewin, 1982) and 'the number of mental comparisons needed to write the program' (Menzies et al., 2002) on the left-hand side of the equation:

$$V^{\text{bits or mental comparisons}} = N^{\text{occurrences of tokens}} * \log_2 n^{\text{distinct tokens}} \quad (6.13)$$

Thus, there is no relationship between the measurement unit on the left-hand side and those on the right-hand side of this equation. Furthermore, on the right-hand side, the true meaning of the multiplication of the 'occurrences of tokens' and the 'distinct tokens' is not clear. Such a multiplication would normally produce a number without a measurement unit; see Figure 6.2 which contains an explanation of the measurement unit produced by log_2, we got this explanation in 2005 by a contact with Mr. Richard Peterson from the Math Forum (Ask Dr. Math) at Drexel University.

In general, in engineering applications we do not take the logarithm of a dimensioned number, only of dimensionless quantities. For instance, in calculating decibels, we take the logarithm of a ratio of two quantities. A ratio of quantities with the same dimensions is itself dimensionless. We can write

$$log(a/b) = log(a) - log(b)$$

making it appear that we are taking the *logs* of dimensioned quantities *(a)* and *(b)*, but the dimensions come out in the wash: by the time we have finished (subtracting one *log* from the other), we have effectively taken the *log* of a dimensionless quantity, *(a/b)*.

We can regard units as factors in an expression, for instance:

$$8\ meters = 8 * [1\ meter]$$
$$800\ cm = 800 * [1\ cm]$$
$$= 800 * 0.01 * [1\ meter]$$

In these terms, we have:

$$(8m)*log_2(8m) = 8*[1m]*log_2(8*[1m])$$
$$= 8*[1m]*(log_2(8)+log_2[1m])$$
$$= (8*log_2(8)+8*log_2[1m])*[1m]$$

That inconvenient $8*log_2[1m]$ is an additive term that depends on the units being used. If it is part of a valid engineering calculation, this term will be cancelled out somewhere in the process. It may be, for instance, that when we take the *log* of *8 meters*, we are actually taking the *log* of a ratio of *8 meters* to a *one-meter* standard length.

Figure 6.2 ***Explanation of the measurement unit produced by log_2***

Equation (6.4) gives the definition of the program potential volume (V^*), which is a prediction of the program volume:

$$V^* = (2 \text{ potential operators} + n_2^* \text{ potential operands}) \log_2 (2 \text{ potential operators} + n_2^* \text{ potential operands}) \quad (6.14)$$

In this equation, the value two was assigned to $n1^*$, as seen in section 6.3. The measurement unit of the left-hand side is the same as in the previous equation (equation (6.3)), while there is no recognizable measurement unit for the right-hand side. As in equation (6.3), such a multiplication would also normally produce a number without a measurement unit, see Figure 6.2.

The program level (L) can be calculated using equation (6.5), in which there is no measurement unit for the left-hand side, either from Halstead himself or from other researchers. In the sense that this is the correct structure for a ratio with the same unit in both numerator and denominator; the end result is therefore a percentage:

$$L = \frac{V^* \text{ bits}}{V \text{ bits}} = \frac{V^* \text{ mental comparisons}}{V \text{ mental comparisons}} \quad (6.15)$$

For equation (6.6), the difficulty (D) is a measure of 'ease of reading' and can be seen as a measure of 'ease of writing' as well (Christensen, Fitsos and Smith, 1981). The right-hand side is also a percentage. What the right-hand side of equation (6.6) means is a riddle, as its associated label on the left-hand side.

In Equation (6.7), for the program level estimator ($\hat{L}$), there is no measurement unit for the left-hand side, while the right-hand side consists of a combination of four distinct measurement units. The exact meaning is again a riddle:

$$\hat{L} = \frac{2^{\text{potential operators}}}{n_1^{\text{distinct operators}}} * \frac{n_2^{\text{distinct operands}}}{N_2^{\text{occurrences of operands}}} \tag{6.16}$$

In equation (6.8), referred to as the intelligent content of the program (I), there is no measurement unit on the left-hand side. For the right-hand side of this equation, the measurement unit of it – which is not known since it is a combination of units – is multiplied by the measurement unit of V:

$$I = \hat{L} * V^{\text{bits}} = \hat{L} * V^{\text{mental comparisons}} \tag{6.17}$$

As for equations (6.6) and (6.7), the exact meaning of the left-hand side of equation (6.8) is a riddle if we attempt to interpret this number with measurement units.

Equation (6.9) is used by Halstead to compute the effort (E) required to generate a program:

$$E^{\text{elementary mental discriminations}} = \frac{n_1^{\text{distinct operators}} \; N_2^{\text{occurrences of operands}} \; N^{\text{occurrences of tokens}} \; \log_2 n^{\text{distinct tokens}}}{2^{\text{potential operators}} \; n_2^{\text{distinct operands}}} \tag{6.18}$$

The measurement unit of the left-hand side of this equation, referred to as 'effort', would be expected to be something such as 'hours' or 'days'. Halstead, however, referred to 'the number of elementary mental discriminations' as the unit of measurement for the

left-hand side. Next, in the sense that the 'distinct operators', the 'distinct operands' and the 'occurrences of operands' are, in a generic sense, 'tokens', then it can be concluded that the measurement unit of the right-hand side of this equation is a combination of measurement units. Therefore, there is no relationship between the units of measurement of the left-hand and the right-hand sides in equation (6.9).

Finally, equation (6.10) is used to compute the required programming time (T) for the program:

$$T \text{ seconds} = \frac{\overset{\text{distinct operators}}{n_1} \quad \overset{\text{occurrences of operands}}{N_2} \quad \overset{\text{occurrences of tokens}}{N} \quad \log_2 \overset{\text{distinct tokens}}{n}}{\underset{\text{potential operators}}{2} \quad \underset{\text{psychological moments per second}}{18} \quad \underset{\text{distinct operands}}{n_2}} \tag{6.19}$$

Again, the measurement unit of the left-hand side, that is, seconds, does not in any way imply the measurement unit of the right-hand side, that is, a combination of many different measurement units. In view of the fact that, Halstead refers to the 'moments' in this equations as "the time required by the human brain to perform the most elementary discrimination" (Halstead, 1977, p. 48).

6.6 Discussion and conclusions

In this chapter, we have investigated a well-known set of measures – Halstead's measures – by focusing on their design and, in particular, on their measurement units. The following comments can be made about Halstead's measures:

- Based on ISO 15939 (ISO, 2002) and the international vocabulary of basic and general terms in metrology (VIM) (ISO, 1993), Halstead's measures can be classified as six based measures (n_1, n_2, N_1, N_2, n_1* and n_2*) and ten derived measures (equations (6.1) to (6.10)).

- Halstead has not explicitly provided a clear and complete counting strategy to distinguish between the operators and the operands in a given program or algorithm. This has led researchers to come up with different counting strategies and, correspondingly, with different measurement results for the same measures and for the same program or algorithm.
- There are problems with the units of measurement for both the left-hand and the right-hand sides of most of Halstead's equations.

- The implementation of the measurement functions of Halstead's measures has been interpreted in different ways than the goals specified by Halstead in their designs. For example, the program length (N) has been interpreted as a measure of program complexity, which is a different characteristic of a program (Fenton, 1994).

- Equations (6.6) and (6.7-1), using basic mathematical concepts, lead to $\hat{L}$ being identical to L; this point can be clarified as follows:

$$\begin{aligned} &\hat{L} = \frac{1}{D} \quad (5.6), \quad D = \frac{1}{L} \quad (5.7\text{-}1), \\ &\hat{L} = \frac{1}{\frac{1}{L}}, \\ &\hat{L} = L. \end{aligned} \tag{6.20}$$

Therefore, using Fenton's description of $\hat{L}$, the program level estimator is identical to the program level.

- Using the previous observation (that is, $L = \hat{L}$), and from equations (6.5) and (6.8), it can be concluded that $I = V^*$. The clarification of this point is as follows:

$$
\begin{aligned}
&I = \hat{L} * V \quad (5.8), \qquad L = \frac{V^*}{V} \quad (5.5), \qquad L = \hat{L} \quad (5.11),\\
&I = L * V,\\
&I = \frac{V^*}{V} \times V,\\
&I = V^* = \text{size unit} \;\begin{cases} \nearrow \text{Bits} \\ \searrow \text{Mental comparisons} \end{cases}
\end{aligned}
\tag{6.21}
$$

Therefore, how we can use the same value to measure both 'intelligent content' (I) and 'program potential volume' (V^*), two different attributes of a program or algorithm? Also, how do we give different units of measurement to the same value?

- A number of addition issues can be raised such as the following: Equations (6.9) and (6.10), which give the programming effort (E) and the required programming time (T) in seconds, do not take into account technology evolution and characteristics: for instance, new programming languages (i.e. the 4^{th} generation programming languages) need less time for programming since most of the programming effort is expended by means of drag-and-drop processes, as in Visual Basic.

6.7 Summary

Throughout this chapter, we have verified the Habra *et al.* (2004) software measurement analysis framework, and we have found that the metrology concepts of this analysis framework are very useful to investigate the designs and definitions of software

measures. This verification was carried out on the Halstead's measures as a case study as some of his measures (i.e. program vocabulary and program length) are mentioned in Annex C of the ISO 9126 parts 2, 3 and 4. After ensuring that the metrology concepts could be applied to analyse the Halstead's measures, we will go through to apply these concepts into the ISO 9126 measures.

In the next chapter (Chapter 7), we will analyse the ISO 9126 measures (quality in-use measures as a case study) to ensure that their design and definitions are follow the concepts of metrology. This will help us in the building of our proposed software product quality maturity model (SPQ^{MM}) since we need to use a set of will-defined measures instead of using of using ill-defined ones.

CHAPTER 7

AN ANALYSIS OF ISO THE 9126-4 FROM THE METROLOGY PERSPECTIVE

7.1 Introduction

The ISO 9126 series of document on software product quality evaluation proposes a set of 197 'metrics' or measures for measuring the various characteristics and subcharacteristics of software quality. However, as typical in the software engineering literature, this ISO set of so-called 'metrics' or measures in ISO 9126 refer to multiple distinct concepts that, in metrology, would have distinct 'labels' (or naming conventions, e.g. terms) to avoid ambiguities.

To help understand and clarify the nature of each of these measures proposed in ISO TR 9126-4 (ISO, 2004f), each of them is analyzed in this chapter from a metrology perspective and is mapped into the relevant metrology concepts. Such a mapping will also contribute in identifying the measurement concepts that have not yet been tackled in the ISO 9126 series of documents. Each of these gaps represents opportunities for improvement in the design and documentation of measures proposed in ISO 9126.

The rest of this chapter is structured as follows:

- Section 7.2 presents an overview of the quality in-use measures as described in ISO 9126-4.
- Sections 7.3, 7.4, 7.5 and 7.6 present the analysis of the effectiveness, productivity, safety and satisfaction measures, respectively.
- Finally, section 7.7 concludes the chapter and provides a set of comments and suggestion for a potential improvement.

7.2 ISO 9126-4: Quality in-use measures

In ISO 9126-4 (ISO, 2004f), the quality in-use measures have been classified into four related collections of measures based on the quality in-use characteristics, that is, effectiveness measures, productivity measures, safety measures, and satisfaction measures. The names of the fifteen measures proposed by ISO for these four characteristics of quality in-use are listed in Table 7.1.

Table 7.1

ISO 9126 Quality in-use characteristics and their measures
(ISO, 2004f)

Characteristic	Measures
Effectiveness	- Task Effectiveness
	- Task Completion
	- Error Frequency
Productivity	- Task Time
	- Task Efficiency
	- Economic Productivity
	- Productive Proportion
	- Relative User Efficiency
Safety	- User Health and Safety
	- Safety of People Affected by Use of the System
	- Economic Damage
	- Software Damage
Satisfaction	- Satisfaction Scale
	- Satisfaction Questionnaire
	- Discretionary Usage

These fifteen measures are analyzed using metrology concepts structure from 'quantities and units' VIM category (ISO, 1993), based on four characteristics, that is, system of quantities, dimension of a quantity, unit of measurement and value of a quantity.

7.3 Analysis of ISO 9126-4 Effectiveness measures

In ISO 9126-4, the claim is that the three Effectiveness measures assess whether the task carried out by users reached the specific goals with accuracy and completeness in a specific context of use (ISO, 2004f). This section presents the outcomes of the mapping of the set of Quantities and Units metrology concepts to the description of Effectiveness measures in ISO 9126-4. A summary of this mapping is presented in Table 7.2.

7.3.1 System of quantities for Effectiveness

7.3.1.1 Base quantities

First, it is observed that these three Effectiveness measures are not collected directly by a measurement system, but are derived from a computation of four base quantities that are themselves collected directly, that is: task time, number of tasks, number of errors made by the user and proportional value of each missing or incorrect component.

The first three of these base measures in Table 7.2 refer to terms in common use, but this leaves much space to interpretation on what is, for example, a '*task*': it does not ensure that the measurement results are repeatable and reproducible across measurers, across groups measuring the same software and, as well, across organizations where a '*task*' might be interpreted differently and with different levels of granularity. This leeway in their interpretation makes a rather weak basis for either internal or external benchmarking.

The third base quantity 'number of errors made by the user' is defined in Annex F of the ISO TR 9126-4, as an "instance where test participants did not complete the task successfully, or had to attempt portions of the task more than once" (ISO, 2004f, p. 47). This definition diverges significantly from the one in IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990, p. 31) where error has been defined as:

"the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition; for example, a difference of 30 meters between a computed result and the correct result".

The fourth base quantity, referred to as the 'proportional value of each missing or incorrect component' in the task's output, is based in turn on another definition whereas each 'potential missing or incorrect component' is given a weighted value *Ai* based on the extent to which it detracts from the value of the output to the business or user (ISO, 2004f). These expansive embedded definitions contain a number of subjective assessment for which no repeatable procedure is provided: value of output to the business or user, the extent to which it detracts, component of a task and potential missing or incorrect component.

7.3.1.2 Derived quantities

The proposed three Effectiveness measures which are defined as a prescribed combination of the above base quantities are therefore derived quantities. The ranges of the results obtained from implementing the corresponding measurement function are introduced in the upper-part of Table 7.2 for each of these derived quantities. These derived quantities inherit the weaknesses of the base quantities from which they are composed of.

7.3.2 Dimension of a quantity for Effectiveness

Emerson (2005) states that the concept of dimension is applicable particularly to the derived quantities: two of these derived quantities, that is 'task effectiveness' and 'task completion' can have values between zero and one, and would be considered as dimensionless quantities since, as stated by Emerson (2005), a ratio of quantities with the same dimensions is itself dimensionless.

Table 7.2

The 'quantities and units' metrology concepts in the Effectiveness measures

Metrology Concepts	ISO 9126-4 (Effectiveness Measures)
System of Quantities:	
- Base Quantities:	1. Task Time. 2. Number of Tasks. 3. Number of Errors Made by the User. 4. Proportional value of each missing or incorrect component.
- Derived Quantities:	5. Task Effectiveness. $0 \leq$ Task Effectiveness ≤ 1 6. Task Completion. $0 \leq$ Task Completion ≤ 1 7. Error Frequency. Error Frequency ≥ 0
Dimension of a Quantity:	
- Quantities of Dimension One (Dimensionless Quantities):	5. Task Effectiveness. 6. Task Completion.
Units of Measurement:	
- Symbols of the Units:	- *s* (Second)
- Systems of Units:	- None.
- Coherent (Derived) Units:	- None.
- Coherent System of Units:	- None.
- International System of Units (SI):	- None.
- Base Units:	1. Second. 2. Task. 3. Error. 4. Non (*ill-defined*)
- Derived Units:	5. (1- a given weight). 6. Task/Task = % 7. Error/Task or Error/Second.
- Off-System Units:	- None.
- Multiple of a Unit:	- None.
- Submultiple of a Unit:	- None.
Value of a Quantity:	
- True Values:	- None.
- Conventional True Values:	- None.
- Numerical Values:	- Results of applying the measurement functions of the above base and derived quantities.
- Conventional Reference Scales (Reference-Value Scales):	1- Task Time.

7.3.3 Units of measurement for Effectiveness

The metrology concepts related with units of measurement are:

- Symbols of the units.
- Systems of units.
- Coherent (derived) units.
- Coherent system of units.
- International system of units.
- Base units.
- Derived units.
- Off-system units.
- Multiple of a unit.
- Submultiple of a unit.

The mappings of these metrology concepts for Effectiveness measures are presented in the Table 7.2. Two metrology concepts must be analyzed with more details: Base Units and Derived Units.

7.3.3.1 Base units

Of the four base quantities, a single one, that is, 'task time', has an internationally recognized standard base unit, that is, a "second", or a multiple of this unit. It also has a universally recognized corresponding symbol (second: '*s*'). The next two base units ('tasks' and 'errors'), do not refer to any international standard of measurement, and must be locally defined (thereby making them poorly fit for comparison purposes, when measured by different people, unless local measurement protocols have been clearly documented, and implemented rigorously in a specific organization). The fourth base unit, for the 'Proportional value of each missing or incorrect component' quantity, is

puzzling because it based on a given weighted value (number), and without any means of measurement unit for this value.

7.3.3.2 Derived units

The 'task effectiveness' derived quantity leads to a derived unit that depends on a given weight (that is (1 – 'a given weight')). Therefore, it also has an unclear derived unit of measurement.

The 'task completion' derived quantity is computed by dividing two base quantities (task/task) with the same unit of measurement.

The definition of the computation of the 'error frequency' derived quantity provides two distinct alternatives for the elements of this computation: thereby it can lead to two distinct interpretations, that is, 'Error/Task' or 'Error/Second'. Of course, this lead to two distinct derived quantities; this occurs due to implementing two different measurement functions (formulas) for the 'error frequency' derived quantity. Of course, this leaves open the possibility of misinterpretations and misuse of measurement results when combined with other units, for example, measures in centimeters and measures in inches cannot be added nor multiplied.

The lack of clarity as well as lack of references to international units of measurements could then explain why there is no attempt to integrate these proposed base and derived quantities into a system of units, including references to coherent units and a coherent system of units.

7.3.4 Value of a quantity for Effectiveness

The metrology has four types of the 'values of a quantity', that is, true value, conventional true values, numerical values and conventional reference scales.

The 'numerical values' are indeed obtained for each base quantities based on the defined data collection procedure; for each of the derived quantities the 'numerical values' are obtained from applying their respective measurement functions. For instance, the 'task effectiveness' and 'task completion' derived quantities are both percentages interpreted as the effectiveness and completion of a specific task respectively.

For the 'task effectiveness' in particular, anybody would be hard press to figure out both a true value as well as a conventional true value; for the task completion and error frequency, such true values would depend on locally defined and rigorously applied measurement procedures, but without reference to universally recognized conventional true values.

Finally, in terms of the metrological values of a quantity, only the 'task time' refers to a conventional reference scale, that is the international standard-etalon for time of which the second is derived. None of the other base quantities in these effectiveness measures refer to conventional reference scales or to locally defined ones.

7.4 Analysis of ISO 9126-4 Productivity measures

In ISO 9126-4, the claim is that the five Productivity measures assess the resources that users consume in relation to the effectiveness achieved in a specific context of use. In ISO 9126-4, the time required to complete a task is considered as the main resources to take into account (ISO, 2004f). This subsection presents the outcomes of the mapping of the set of Quantities and Units metrology concepts to the 2001 description of Productivity measures in ISO 9126-4, a summary of this mapping is presented in Table 7.3.

Table 7.3

The 'quantities and units' metrology concepts in the Productivity measures

Metrology Concepts	ISO 9126-4 (Productivity Measures)
System of Quantities:	
- Base Quantities:	1. Task Time. 2. Cost of the Task. 3. Help Time. 4. Error Time. 5. Search Time.
- Derived Quantities:	6. Task Effectiveness. $0 \leq$ Task Effectiveness ≤ 1 7. Task Efficiency. Task Efficiency ≥ 0 8. Economic Productivity. Economic Productivity ≥ 0 9. Productive Proportion. $0 \leq$ Productive Proportion ≤ 1 10. Relative User Efficiency. $0 \leq$ Relative User Efficiency ≤ 1
Dimension of a Quantity:	
- Quantities of Dimension One (Dimensionless):	6. Task Effectiveness. 7. Task Efficiency. 8. Economic Productivity. 9. Productive Proportion. 10. Relative User Efficiency.
Units of Measurement:	
- Symbols of the Units:	- *s* (Second) - Currency Symbol (for example *$*)
- Systems of Units:	- None.
- Coherent (Derived) Units:	- None.
- Coherent System of Units:	- None.
- International System of Units (SI):	- None.
- Base Units:	1. Second. 2. Currency Unit. 3. Second. 4. Second. 5. Second.
- Derived Units:	6. (1- a given weight). 7. ?/Second. 8. ?/Currency Unit. 9. Second / Second = % 10. 'Task Efficiency' Measurement Unit/'Task Efficiency' Measurement Unit = %.
- Off-System Units:	- None.
- Multiple of a Unit:	- None.
- Submultiple of a Unit:	- None.
Value of a Quantity:	
- True Values:	- None.
- Conventional True Values:	- None.
- Numerical Values:	- Results of applying the measurement functions of the above base and derived quantities.
- Conventional Reference Scales:	1. Task Time.

7.4.1 System of quantities for Productivity

One of the five proposed productivity measures in ISO 9126-4 is a base quantity (Task time) while the other four ones are derived quantities (task efficiency, economic productivity, productive portion and relative user efficiency).

In addition, the 'task efficiency' refers explicitly to another derived quantity 'task effectiveness' already analyzed in the previous section.

It is to be noted that these derived quantities are themselves based on five base quantities: Task Time, Cost of the Task, Help Time, Error Time, and Search Time.

7.4.2 Dimension of a quantity for Productivity

All of the productivity metrics are dimensionless quantities except the 'Task Time'.

7.4.3 Units of measurement for Productivity

In the lower-middle part of Table 7.3, for the base and derived quantities, there are five base units and no explicit derived units. However, it can be observed that the measurement unit for the 'task effectiveness' is not completely clear since it depends on an ill-defined 'given weight':

$$\text{'task efficiency' unit} = \frac{\text{'task effectiveness' unit}}{\text{second}} \tag{7.1}$$

$$= \frac{1 - \text{'a given weight' unit}}{\text{second}} \tag{7.2}$$

$$\boxed{= \frac{?}{\text{second}}} \tag{7.3}$$

Similarly, the measurement unit of the 'economic productivity' depends on the measurement unit of the 'task effectiveness' derived quantity which is unknown:

$$\text{'economic productivity' unit} = \frac{\text{'task effectiveness' unit}}{\text{currency unit}} \tag{7.4}$$

$$= \frac{1 - \text{'a given weight' unit}}{\text{currency unit}} \tag{7.5}$$

$$\boxed{= \frac{?}{\text{currency unit}}} \tag{7.6}$$

Whereas, there is no measurement unit for the 'productive proportion' since it has the same measurement unit in both numerator and denominator; therefore the result is a percentage:

$$\boxed{\text{'productive proportion' unit} = \frac{\text{second}}{\text{second}}} \tag{7.7}$$

Finally, for the 'relative user efficiency' there is no measurement unit since the measurement units in both the numerator and denominator are the same; that is, the 'task efficiency' measurement unit, the result of this derived quantity also is a percentage. This point can be clarified as follows:

$$\text{'relative user efficiency' unit} = \frac{\text{'task efficiency' unit}}{\text{'task efficiency' unit}} \tag{7.8}$$

$$= \frac{\dfrac{\text{'task effectiveness' unit}}{\text{second}}}{\dfrac{\text{'task effectiveness' unit}}{\text{second}}} \tag{7.9}$$

$$= \frac{\frac{1\text{-'a given weight' unit}}{\text{second}}}{\frac{1\text{-'a given weight' unit}}{\text{second}}} \tag{7.10}$$

$$\boxed{= \frac{\frac{?}{\text{second}}}{\frac{?}{\text{second}}}} \tag{7.11}$$

7.5 Analysis of ISO 9126-4 Safety measures

In ISO 9126-4, the safety measures claim to assess the level of risk of harm to people, business, software, property, or the environment in a specific context of use; it includes the health and safety of both the users and those who affected by use, as well as unintended physical or economic consequences (ISO, 2004f).

To evaluate the safety characteristic of a software product, four derived quantities must be quantified (i.e. 'user health and safety', 'software damage', 'economic damage' and 'safety of people affected by use of the system'). Each of these derived quantities is the result of a computational formula (function) which consists of a combination of pre-collected base quantities (i.e. 'number of usage situations', 'number of people', 'number of occurrences of software corruption', 'number of occurrences of economic corruption' and 'number of users'). It can be observed that the resulting values of all the derived quantities should be between zero and one. For the detailed analysis of the safety measures, see Annex I.

All the safety measures are dimensionless quantities; there are five base units and two derived units for these quantities. In addition, two of the derived quantities have no measurement units since they have the same measurement unit in numerator and denominator, that is, the 'user health and safety' and 'safety of people affected by use of the system' derived quantities; whereas none of the measurement units have a symbol.

7.6 Analysis of ISO 9126-4 Satisfaction measures

The satisfaction measures in ISO 9126-4 claim to assess the user's attitudes towards the use of the product in a specific context of use (ISO, 2004f).

All the three proposed satisfaction measures are derived quantities (i.e. 'satisfaction scale', 'satisfaction questionnaire', and 'discretionary usage') which themselves depend on four base quantities (i.e. 'population average', 'number of responses', 'number of times that specific software function/application/systems are used', and 'number of times that specific software function/application/systems are intended to be used'). Two of the proposed satisfaction measures are dimensionless quantities, that is, 'satisfaction questionnaire' and 'discretionary usage'. Annex II contains the analysis of the satisfaction metrics.

Regarding the measurement units, there are four base units and no derived units; however, the 'satisfaction scale' measurement unit is not clear since it depends on a 'questionnaire producing psychometric scales'. The clarification of this point is as follow:

$$\boxed{\text{'satisfaction scale' unit} = \frac{\text{psychometric scale unit}}{\text{people}}} \qquad (7.12)$$

7.7 Conclusion

The ISO International Vocabulary of Basic and General Terms in Metrology (VIM) represents the international consensus on a common and general terminology of metrology concepts. However, up until recently, it was not usual practice in software engineering measurement to take into account metrology concepts and criteria neither in the design of software measures, nor in their use and interpretation of measurement results.

This chapter has presented an analysis of the ISO 9126-4 Technical Report on quality in-use measures and has investigated to which extent it addresses the metrology criteria found in classic measurement. Based on the analysis in Sections 7.3, 7.4, 7.5 and 7.6, the following comments and suggestions can be made:

- Identifying and classifying the quality in-use measures into base and derived quantities makes it easy to determine which ones should be collected (base quantities) in order to be used in computing the other quantities (derived quantities).
- Based on equations 7.3, 7.6 and 7.11, some of the derived units are ambiguous since they depend on other quantities with unknown units.
- None of the quality in-use measures refer to any 'system of units', 'coherent (derived) unit', 'coherent system of units', 'international system of units (SI)', 'off-system units', 'multiple of a unit', 'submultiple of a unit', 'true values', 'conventional true values', and 'numerical values'.
- Neither the base nor the derived quantities have symbols for their measurement units, except for the 'task time'.

It is noted that the ranges of the results of many of the derived measures in ISO 9126-4 are between zero and one. Therefore, it is easy to convert them to be percentage values. However, from our point of view, it will be more understandable if these results ranked into qualitative values, for example, for the 'task completion', if the percentage result is 100% then the completion of the task is 'excellent', if the result is 80% then the completion of the task is 'very good', and so on.

In addition, we noted that each characteristic (subcharacteristic in case of the internal and external quality of software product) has a number of measures, that is, when we need to measure such a characteristic we will get a set of numbers which represent that characteristic. This will be very confusing for the decision makers since taking a decision based on a single value will be more accurate and reliable.

We will employ the above to notes along with our classification of the ISO 9126 measures (into base and derived measures as will be described in the next chapter) in the building of our proposed software product quality maturity model (SPQ^{MM}), therefore, the SPQ^{MM} will be partially based on the ISO 9126 measures taking into account our suggestions in this chapter.

Using the ISO 9126-4 Technical Report on the measurement of software 'quality in-use' as a case study, this chapter has investigated and reported on the extent to which this ISO series addresses the metrology criteria typical of classic measurement. Areas for improvement in the design and documentation of measures proposed in ISO 9126 have been identified.

CHAPTER 8

AN ISO-BASED INFORMATION MODEL TO ADDRESS THE HARMONIZATION ISSUES IN THE NEW ISO 25020 AND ISO 25021 STANDARDS

8.1 Introduction

The ISO has recognized a need for further enhancement of ISO 9126, primarily as a result of advances in the fields of information technologies and changes in environment. Therefore, the ISO is now working on the next generation of software product quality standards, which will be referred to as Software Product Quality Requirements and Evaluation (SQuaRE – ISO 25000). Once completed, this series of standards will replace the current ISO 9126 and ISO 14598 series of standards.

One of the main potential differences between (and objectives) of the SQuaRE series and the current ISO 9126 series is the coordination and harmonization of its contents with the ISO standard on software measurement process – ISO 15939 (ISO, 2007a; 2007b).

Recently, ISO has come up with two new standards on software product quality, that is, ISO 25020 and ISO 25021. However, our analysis illustrates that these new standards are not entirely conformant to the stated goal of coordinating and harmonizing their contents to the ISO standard on software measurement process – ISO 15939.

This chapter presents some of the harmonization issues we identified as arising from introducing the ISO 25020 and ISO 25021 new standards with respect to previously published measurement standards for software engineering, including ISO 15939, ISO 14143-1 and ISO 19761; in this chapter we also propose ways to address them using the measurement information model of ISO 15939 on software measurement process.

The rest of this chapter is organized as follows: Section 8.2 presents the outstanding harmonization issues in terminology and coverage in ISO FDIS 25020 and ISO 25021. Section 8.3 presents our solution for alignment of the ISO models of software product quality with the measurement information model of ISO 15939. Finally, some examples are presented in section 8.4, and a discussion and a summary are presented in section 8.5.

8.2 Outstanding harmonization issues

8.2.1 Terminology

The ISO 9126 working group has come up with the introduction of three new expressions in ISO TR 25021, namely (ISO, 2007a; 2007b):

1. 'Quality measure element categories'.
2. 'Quality measure elements'.
3. 'Quality measures'.

We have identified that the introduction of these new terms raises the following concern: either the proper mapping to the set of classic metrology terms has not yet been completed or there are concepts and related terms missing in the metrology vocabulary. The latter would be surprising, since metrology is a rather mature domain of knowledge based on centuries of expertise in the field of measurement and related international standardization. In this section, we revise the new documents ISO FDIS 25020 and ISO 25021 in order to recommend a proper mapping of concepts to the related metrology terms and to ISO 15939.

In ISO 25021, it is claimed that a 'quality measure element' is either a base measure or a derived measure (ISO, 2007a; 2007b), but then the consensual metrology terms are ignored in favour of locally defined WG6 measures, thus bypassing the ISO and SC7 harmonization requirements on measurement terminology.

The 'quality measure elements' are described as an input for the measurement of the 'software quality measures' of external quality, internal quality and quality in-use (ISO, 2007a; 2007b). Figure 8.1 shows the relationship between the 'quality measure elements' and the 'software quality measures', and between the 'software quality measures' and the quality characteristics and subcharacteristics. In metrology, these would correspond to base measures and derived measures respectively. It can be observed as well that these measures, in particular the derived measures, are defined specifically to measure the subcharacteristics of internal and external quality or the characteristics of quality in-use. None of these is directly related to the top level of 'software quality' (which is itself decomposed into three models, then into 16 characteristics and further into a large number of subcharacteristics). Therefore, the 'software quality measures' expression, which has been selected in ISO 25021, is at a level of abstraction that does not represent the proper mapping of the measures to the concept being measured.

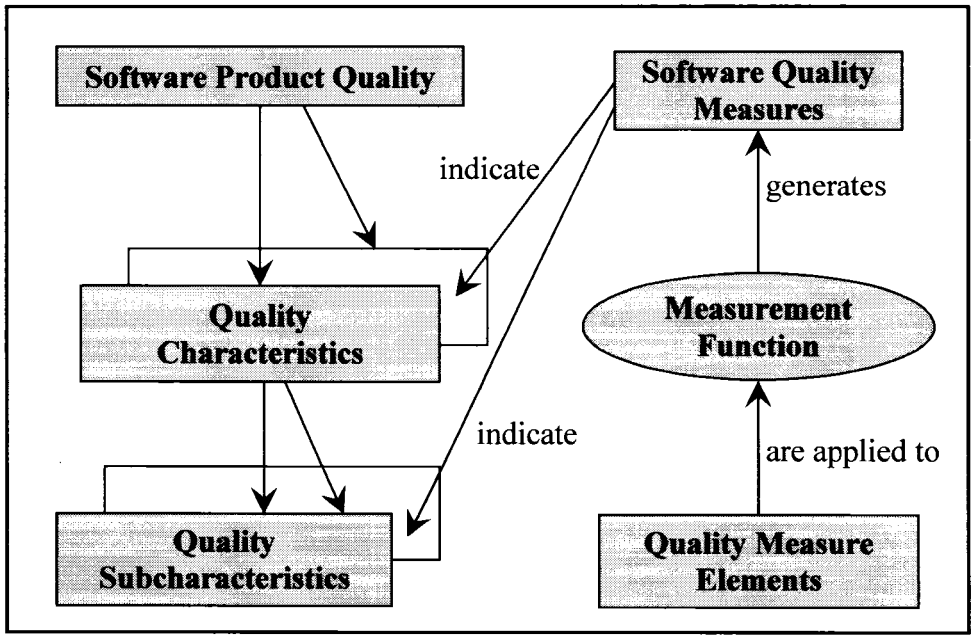

Figure 8.1 ***Quality measure elements concept in the 'Software Product Quality Measurement Reference Model'***
(ISO, 2007a; 2007b)

For the 'quality measure element categories', in ISO FDIS 25021, there are 15 categories (ISO, 2007b), see Table 8.1. It is to be noted that in ISO DIS 25021 there are no specific quality measure elements proposed within the 'number of user operations' or 'number of system operations' categories.

Table 8.1

The fifteen categories of Quality Measure Element

No.	Category
1.	Data Size
2.	Number of Data Items
3.	Number of Failures
4.	Number of Faults
5.	Number of Functions
6.	Number of I/O Events
7.	Number of Requirements
8.	Number of Restarts
9.	Number of System Operations
10.	Number of tasks
11.	Number of Test Cases
12.	Number of Trials
13.	Number of User Operations
14.	Product Size
15.	Time Duration

It can be observed that – in Table 8.1 – a number of the quantities have a label starting with 'number of'. However, these do not use a reference scale typical of measures in the sciences or in engineering, but are rather counts of entities. For any of these proposed counts, such as the 'number of functions', no specific measurement method is proposed for an identification of the number of functions in a consistent manner across measurers and organizations; for instance, the definition of the word 'function' could differ from

one individual to another within the same organization, and more so across organizations. Therefore, to say in ISO TR 25021 that such numbers are obtained by an 'objective' method is an overstatement, since they must be obtained mostly on the basis of the judgment of the person carrying out the count.

Of the 15 quality measure element categories, only 'time' comes from a classic base measure using, for instance, the international standard unit of the second (or a multiple or submultiple of it) as its reference scale. There are also measuring instruments to ensure that time measurements are indeed obtained in an objective manner.

It can also be observed that, of the 15 categories in Table 8.1, at most four are directly related to the quality of software: number of faults, number of failures, number of restarts and number of trials. None of the other 11 measures is directly or indirectly related to the quality of software. In fact, they are strictly independent of it, as they are often used for normalization purposes, for instance.

For the 'product size' category, the ISO TR 25021 lists only one way to measure the product size, that is, non-comment lines of code. There are also other ways to measure the product size, such as, function points, modules, classes and visual structures. Furthermore, there are various methods for counting lines of code and for measuring function points. Therefore, this quality measure element category could be further split into different quality measure elements (base measures). Moreover, the ISO has specified mandatory requirements for functional size measurement methods – ISO 14143-1 (ISO, 1998c), and has recognized four different functional size measurement methods as ISO standards meeting these requirements, such as COSMIC-FFP (ISO, 2003b). None of these existing ISO software engineering standards, which are referenced in ISO 90003 (ISO, 2004e), has been mentioned or referenced in ISO TR 25021. Also, the various methods available to obtain those numbers have their strengths

and weaknesses, from a measurement perspective, in terms of repeatability, reproducibility, software domains of applicability and accuracy.

In summary, from our point of view, issuing new terms such as 'quality measure element categories', 'quality measure elements' and 'quality measures' is not necessary; the terminology concepts in ISO VIM (ISO, 1993) and in ISO 15939 (ISO, 2002) are sufficient.

8.2.2 Limited coverage of the ISO quality models and corresponding measures

ISO TR 9126, parts 2 to 4, presents the ISO inventory of measures for the full coverage of the ISO software product quality models (internal quality, external quality and quality in-use) for measuring any of their quality characteristics and subcharacteristics. The full sets of base measures in these three parts of ISO 9126 are presented in Annex IV and include 82 base measures.

Of these 82 base measures, only 57 are included in ISO 25021; this means that the coverage in this new ISO document is limited, and the reasons for this are not obvious. In addition, out of the 197 measures in ISO 9126, only 51 are 'selected' in ISO 25021 as 'quality measures'. The content coverage of this subset of quality measures (derived measures) is limited and no specific criteria are provided on how they have been selected. Some generic information is provided in this standard to suggest that these measures were derived from a questionnaire-based survey; however, it does not provide the reader with information about the criteria for selection, the size and representativeness of the sample in the countries where the data were collected, or the representativeness of this sample outside these countries. Another claim, that "they represent a default kernel of quality measures, which are proven to be used in common practice" (ISO, 2007b, p. 47), is not supported by documented evidence, nor is there a discussion of its generalizability outside its data collection context.

Tables III.1, III.2 and III.3 in Annex III present a detailed analysis of the coverage of the quality measures in ISO 25021, together with the corresponding availability in ISO 9126. Table III.1 specifically illustrates that 34 measures for the 'external quality' of software product are selected in ISO 25021 out of an inventory of 112 in the corresponding ISO 9126-2, while 78 measures are excluded, again without a documented rationale.

Table III.2 provides similar information for the 'internal quality' of software product as selected in ISO 25021; out of 70 measures, only 15 measures have been selected and cover only four of the six quality characteristics of the ISO model of internal quality, and only nine of 27 subcharacteristics; again, the rationale for excluding any characteristic or subcharacteristic is not documented. Similarly in Table III.3, for the 'Quality in-Use' quality measures:

- Included: only two measures of the 15 already available in ISO 9126-4;
- Excluded: two quality in-use characteristics, that is, 'safety' and 'satisfaction';
- Does not include any quality measure elements related to the 'Number of User Operations' and 'Number of System Operations'.

8.2.3 Redundancy issues

Some additional information included in ISO 25021 has already been covered in ISO 9126 documents, and will be included in the ISO 25000 series; for instance, information about the 'scale types' is covered through rephrasing information contained in other documents, once again increasing synchronization and harmonization right away and over the long term.

Similarly for the narratives about the measures of internal software quality, external software quality and software quality in-use, as well as for the narratives about the software measurement methods; this is contrary to the ISO practice of avoiding

duplication, redundancy or the rephrasing of information across ISO documents, and increases the possibility of inconsistencies across documents; it could lead to significant effort over the long term in maintaining synchronization of documents covering similar subsets of information.

These examples point to configuration management issues over the long term which will represent additional cost to the purchasers of these ISO documents, since they will be required to pay twice for the same information which is a subset of the full inventory. This could lead to some confusion for standards users as to which of these documents is most valuable to a standard purchaser, and under what circumstances.

We have illustrated in this section how the issue of ambiguity and redundancy in ISO FDIS 25020 and ISO 25021 new terms 'quality measure elements categories', 'quality measure elements' and 'quality measures' can be avoided through the use of the corresponding metrology concepts and terms.

8.3 Mapping the quality model to the Measurement Information Model

The following two expressions come from the ISO standard on software measurement process, ISO 15939 (ISO, 2002), which is itself based on the definitions in the ISO International Vocabulary of Basic and General Terms in Metrology (VIM) (ISO, 1993):

- Base measure: a measure defined in terms of an attribute and the method for quantifying it. A base measure is functionally independent of other measures.
- Derived measure: a measure defined as a function of two or more values of base measures. A transformation of a base measure using a mathematical function can also be considered as a derived measure.

Practically, the data collection associated with a property of an object (or concept), and quantification of it, happens at the base measure level, at which time a measurement unit is assigned based on the rules of the measurement method used for the quantification.

At the derived measure level, the base measures have been already collected and are being assembled according to the combination rules (e.g. a computational formula) defined within each derived measure. A derived measured is, therefore, the product of a set of measurement units properly combined (through a measurement function). This combination is then labelled to represent an attribute (of a characteristic or subcharacteristic of the quality) of a software product.

Table 8.2 shows examples of base measures used in the definitions of the measures documented in ISO 9126-2, -3 and -4 (see Annex IV for the complete list of base measures). Table 8.2, shows the name of each base measure and the unit of measurement that is given to its value. These base measures can then be used to calculate each of the derived measures (akin to metrics) in ISO 9126-4.

Table 8.2

Examples of Base Measures in ISO 9126-4

Quality in-use Base Measures		
	Measure Name	**Unit of Measurement**
1	Task Effectiveness	(a given weight)
2	Total Number of Tasks	Task (number of)
3	Task Time	Minute
4	Cost of the Task	Dollar
5	Help Time	Second
6	Error Time	Second
7	Search Time	Second
8	Number of Users	User (number of)
9	Total Number of People Potentially Affected by the System	Person (number of)
10	Total Number of Usage Situations	Situation (number of)

Each of these base measures must be collected individually. They can be used at least once, or multiple times, for obtaining the derived measure required to quantify the software properties specified in the ISO 9126 quality model. Table 8.3 provides an example of where some base measures are used throughout ISO 9126-2. For instance, the base measure, 'number of inaccurate computations encountered by users', is used only once in 'external functionality - accuracy measures', while the base measure 'number of items requiring compliance' can be used in six subcharacteristics of external quality (ISO 9126-2). The construction of derived measures is based on a computational formula consisting of two or more base measures (see Annex V for the complete cross-reference lists of the base measures usage).

Table 8.3

Examples of the use of Base Measures in ISO 9126-2

	Measure Name	External																								
		Functionality					Reliability				Usability					Efficiency			Maintainability					Portability				
		F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5
1	Number of functions	✓									✓	✓	✓											✓			✓	
2	Operation time		✓	✓	✓		✓		✓			✓	✓			✓	✓			✓	✓	✓		✓		✓		
3	Number of inaccurate computations encountered by users		✓																									
4	Number of data formats			✓																								
5	Number of illegal operations				✓																							
6	Number of items requiring compliance					✓				✓					✓			✓					✓					✓
7	Number of interfaces requiring compliance					✓																						
8	Number of faults						✓																	✓	✓			

Such lists of base measures and of the usage cross-references are currently missing from ISO 9126 and would be helpful to those designing programs for implementing measurement of the quality of software products using ISO 9126 quality models and related measures. In particular, these lists can help in:

- Identifying, selecting and collecting a base measure (once), and then using this base measure to evaluate a number of derived measures.
- Knowledge of which base measures are required to evaluate specific software quality attributes (characteristics and subcharacteristics).

Next, we present a mapping of both the measures and of the quality models in ISO 9126 to the measurement information model described in ISO 15939. As a first step, we refer to the bottom section of Figure 8.2 by the term 'Data Collection' (e.g. the measurement methods and the base measures), the middle section by the term 'Data Preparation' using agreed upon mathematical formula and related labels (e.g. measurement functions and derived measures) and the top section by the term the 'Data Analysis' (e.g. analysis model, indicator and interpretation).

Both data collection and data preparation have already been discussed, we now focus on the 'Data Analysis' section. It is in the 'Analysis Model' part of the ISO 15939 measurement information model that the ISO 9126 models of software product quality are to be put to use. Figures 2.1, 2.2, 2.3 and 2.4 in Chapter 2 present these generic models of ISO 9126 (ISO, 2001b). These generic ISO models are to be instantiated in any particular context of measuring the quality of a specific software product. This is usually performed in a four-step process, as summarized in Figure 8.2.:

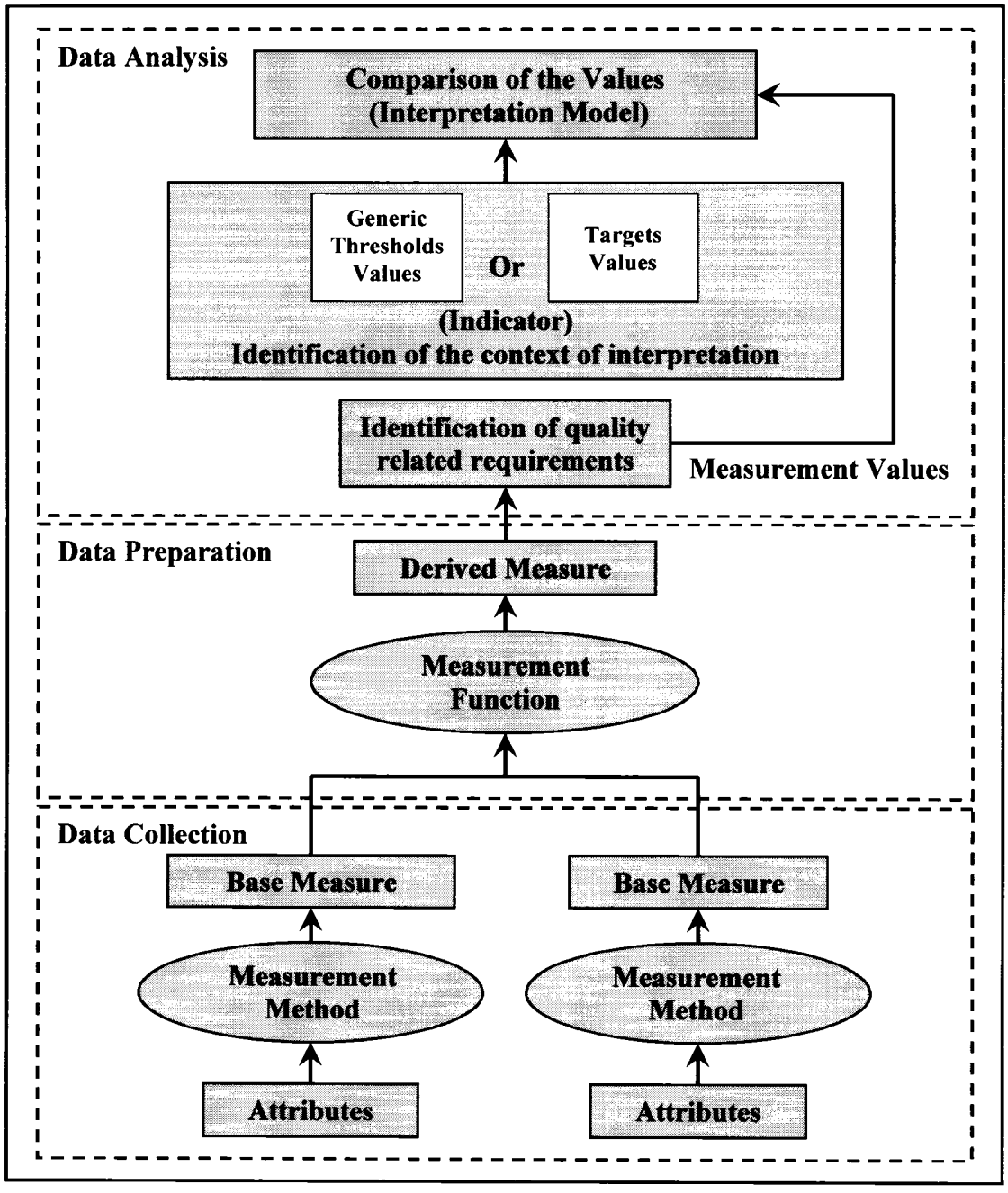

Figure 8.2 ***Mapping to the Measurement Information Model***

1. Identification of quality related requirements, that is, the selection of the parts of the ISO quality models that are relevant to a particular context of quality evaluation.

2. Identification of the context of interpretation, that is:
   - the selection of reference values, such values being either generic or specific threshold values, or
   - the determination of targets specified for a particular context.
3. Use of the derived measures from the data preparation phase to fill out the instantiated quality model determined in 1.
4. Comparison of the results of step 3 with either the set of reference values or targets determined in step 2.

8.4 Examples

Some examples are presented next to illustrate the process described in Figure 8.2. These include some of the ISO 9126 base measures and how they are combined to construct a derived measure using a computational formula (measurement function):

- **Example 1:**
  - Data Collection:
    - Base Measure 1 (B1): Number of inaccurate computations encountered by users.
    - Base Measure 2 (B2): Operation time.
  - Data Preparation:
    - Derived Measure: B1 / B2
    - Name of Derived Measure: Computational accuracy.
  - Data Analysis:
    - Quality group name: External quality measures.
    - Characteristic: Functionality.
    - Subcharacteristic: Accuracy.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).

- **Example 2:**
  - Data Collection:
    - Base Measure 1 (B1): Number of detected failures.
    - Base Measure 2 (B1): Number of performed test cases.
  - Data preparation:
    - Derived Measure: B1 / B2
    - Name of Derived Measure: Failure density against test cases.
  - Data Analysis:
    - Quality group name: External quality measures.
    - Characteristic: Reliability.
    - Subcharacteristic: Maturity.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).
- **Example 3:**
  - Data Collection:
    - Base Measure 1 (B1): Number of memory related errors.
    - Base Measure 2 (B2): Number of lines of code directly related to system calls.
  - Data Preparation:
    - Derived Measure: B1 / B2
    - Name of Derived Measure: Memory utilization message density.
  - Data Analysis:
    - Quality group name: Internal quality measures.
    - Characteristic: Efficiency.
    - Subcharacteristic: Resource utilization.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).

- **Example 4:**
  - Data Collection:
    - Base Measure 1 (B1): Task time.
    - Base Measure 2 (B2): Help time.
    - Base Measure 3 (B3): Error time.
    - Base Measure 4 (B4): Search time.
  - Data Preparation:
    - Derived Measure: (B1-B2-B3-B4) / B1
    - Name of Derived Measure: Productive proportion.
  - Data Analysis:
    - Quality group name: Quality in-use measures.
    - Characteristic: Productivity.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).
- **Example 5:**
  - Data Collection:
    - Base Measure 1 (B1): Number of errors made by user.
    - Base Measure 2 (B2): Number of tasks.
  - Data Preparation:
    - Derived Measure: B1 / B2
    - Name of Derived Measure: Error frequency.
  - Data Analysis:
    - Quality group name: Quality in-use measures.
    - Characteristic: Effectiveness.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).

- **Example 6:**
  - Data Collection:
    - Base Measure 1 (B1): Task effectiveness.
    - Base Measure 2 (B2): Task time.
  - Data Preparation:
    - Derived Measure: B1 / B2
    - Name of Derived Measure: Task efficiency.
  - Data Analysis:
    - Quality group name: Quality in-use measures.
    - Characteristic: Effectiveness.
    - Comparison of values obtained with the indicators (generic thresholds and/or targets).

8.5 Summary and discussion

Within the ISO's mandate to upgrade its set of technical reports on the measurement of the quality of software products (ISO 9126), ISO has come up with a new structure for upgrading the current series of ISO 9126 documents for the measurement of the quality of software products. This new structure is referred to as 'Software Product Quality Requirements and Evaluation – SQuaRE'. In this chapter, we presented an alignment of the ISO models of software product quality with the measurement information model of ISO 15939 and how to use them for data collection, data preparation and data analysis; some examples have also been illustrated.

In addition, in this chapter, some issues have been raised concerning three new concepts proposed in ISO 25020 and ISO 25021, that is: 'quality measure element categories', 'quality measure elements' and 'quality measures'. The following is a summary of the harmonization issues identified:

1. Terminology in ISO 25021:
    - What is referred to as a 'quality measure element' corresponds to the classic concept of 'base measure' in ISO 15939;
    - What is referred to as 'software quality measure':
        - Corresponds to the classic concept of 'derived measure' in ISO 15939;
        - It is not at the proper level of abstraction for the concept being measured when mapped to the hierarchy of concepts for software product quality adopted by the ISO.
    - In both ISO FDIS 25020 (ISO, 2007a, p. 4) and ISO 25021 (ISO, 2007b, p. 11), the 'measurement method' is defined as "a logical sequence of operations, described generically, used in quantifying an attribute with respect to a specific scale". But, in ISO 25021, it is used in the 'set of quality measure elements' to represent the type of the measurement method (objective or subjective). In contrast, a new data field called 'detail' is used to represent the measurement method.
2. Harmonization with the Information Model of ISO 15939:
    - Unless the terminology is harmonized with the ISO International Vocabulary of Basic and General Terms in Metrology, then it is a challenge to align the older versions of the ISO 9126 and ISO 14598, and it will be even more challenging with the updates in ISO 25000.
    - Should the harmonization of terminology proposed in this thesis chapter be accepted at the ISO level, it would be then easier to map each of these ISO 9126 and ISO 14598 series into the Information Model of ISO 15939.
3. Description harmonization:
    - A large number of the base measures proposed in ISO 25021 are counts of entities rather than measures per se with required metrological characteristics such as: unit, scale, dimension, measurement method, measurement procedures, etc.

- In ISO 25021, in some instances, like 'product size' for example, there is no reference to other existing ISO standards for software size, such as ISO 19761, etc.
- There are a number of claims that the proposed base measures are 'objective', while they are obviously derived from a manual process without precisely documented measurement procedures, thereby leaving much space to the measurer's judgment.

4. Coverage harmonization in ISO 25021:
   - The set of base measures documented represents only a limited subset of the base measures within ISO 9126, parts 2 to 4; the rationale for inclusion or exclusion is not documented.
   - The set of base measures does not allow coverage of the full spectrum of quality characteristics and subcharacteristics in ISO 9126, parts 2 to 4; again, the rationale for inclusion or exclusion is not documented.

These concerns can be summarized as follows:

- Quality measure elements categories and quality measure elements: non alignment with the classic terminology on measurement is puzzling.
- Quality measures: some inconsistencies in the terminology used, and some ambiguity about which level of the ISO 9126 multi-level standard is being applied.

From the above analysis, the following recommendations are put forward to the ISO working group dedicated to the improvement of ISO documents on software product quality:

- Ensure that the terminology on software product quality measurement is fully aligned with the classic measurement terminology in the sciences and in engineering;
- Provide full coverage of the base measures for all three ISO models of software quality;

- Provide improved documentation of the base measure using the criteria from metrology;
- Provide clear mapping and traceability of the new ISO 25000 documents to the ISO 15939 Information Model.

We have analyzed some of the new terms weaknesses and have proposed ways to address them by using the ISO 15939 measurement information model on software measurement process. Briefly, using predefined terms such as 'base measure' and 'derived measure', as well as the proper mapping to the Measurement Information Model in well-developed standards like ISO 15939 and the International Vocabulary of Basic and General Terms in Metrology (ISO VIM) is more utile than the introduced weakly defined terms.

This analysis has been done because the Quality Measurement Division standards of the SQuaRE series of standards (i.e. ISO 25020, ISO 25021, ISO 25022, ISO 25023, and ISO 25024) will soon replace the current ISO 9126 standard which will be partially used as an input (both its quality models and quality measures) to our software product quality maturity model SPQ^{MM}.

CHAPTER 9

THE STRUCTURE OF THE QUALITY MATURITY MODEL

9.1 Introduction

A key difficulty in choosing a quality model is the diversity of such models described in the literature, the variety of views on quality and the detailed techniques they embody; for example, McCall's quality model (McCall, Richards and Walters, 1977), Boehm's quality model (Boehm et al., 1978; Boehm, Brown and Lipow, 1976), Dromey's quality model (Dromey, 1995; 1996) and the FURPS quality model (Grady, 1992), see Chapter 4 for the details of these quality models. In addition, hundreds of software measures – or what are commonly called 'software metrics' – have been proposed to assess and evaluate software quality, but, unfortunately, without wide international consensus on the use and the interpretations of these quality models and their software measures.

The development of a consensus in software engineering standards organizations (ISO and IEEE) is now leading to some agreement on both the quality model contents and corresponding measures. Our strategy is to build a maturity model based not on our own – or any individual – views of software quality, but on the emerging consensus in the software engineering standards community. Therefore, the proposed maturity model is based on the following ISO and IEEE standards:

- ISO 9126 – Part 1: Quality Model (ISO, 2001b).
- ISO 9126 – Part 2: Internal Measures (ISO, 2003c).
- ISO 9126 – Part 3: External Measures (ISO, 2003d).
- ISO 9126 – Part 4: Quality in-Use Measures (ISO, 2004f).
- ISO 15026: System and Software Integrity Levels (ISO, 1998b).
- IEEE Std. 1012: Standard for Software Verification and Validation (IEEE, 1998).

In addition to the above-mentioned standards, the so-called sigma concepts are used to align the mapping between the quality levels and the maturity levels of the quality of the software product.

A Software Product Quality Maturity Models (SPQ^{MM}) could be used to determine the maturity of the quality of a specific software product. More specifically, it could be used to:

- Certify a quality maturity level for a new software product, which could help promote it on the market;
- Benchmark existing software products to assist in deciding which of them to select based on their quality maturity levels;
- Assess the quality of the software product during the development life-cycle to investigate the relationships between the development stages and to find the weaknesses of the software product in order to make improvements to it.

This chapter describes the proposed Software Product Quality Maturity Model (SPQ^{MM}), which consists of three quality maturity submodels (viewpoints) that can be used not only once the software product has been delivered, but also throughout the life-cycle:

- Software Product Internal Quality Maturity Model – $SPIQ^{MM}$.
- Software Product External Quality Maturity Model – $SPEQ^{MM}$.
- Software Product Quality-in-Use Maturity Model – $SPQiU^{MM}$.

The rest of this chapter is structured as follows: Section 9.2 presents an architectural view of the quality maturity model and Section 9.3 describes the contents of the software product quality maturity model.

9.2 Quality maturity model: an architectural view

The architecture of the Software Product Quality Maturity Model (SPQ^{MM}) proposed in this Chapter is based on two sets of concepts which exist in industry in general, which are:

- The levels of quality of a product, and
- A quantitative approach to product quality.

For a quantitative approach to product quality, the six-sigma approach to software product quality (Akingbehin, 2005; Head, 1994; Sauro and Kindlund, 2005a) has been selected to build the quality maturity model presented in this chapter.

9.2.1 Quality maturity levels

For the levels of quality of a product, the following five quality maturity levels have been identified from the observation of general industry practices outside the software domain:

- Guaranteed
- Certified
- Neutral
- Dissatisfied
- Completely Dissatisfied

To determine the maturity levels, concepts from the six-sigma approach are used. Sigma is used in statistics to denote the standard deviation, a statistical measurement of variation, that is, the exceptions to expected outcomes; the standard deviation can be considered as a comparison between expected results (or outcomes) in a group of operations, versus those that fail. Thus, the measurement of standard deviation shows that the rates of defects, or exceptions, are measurable. The six-sigma is the definition of

outcomes as close as possible to perfection; for example, with six sigma (with 1.5 sigma shift – see Figure 9.1), we achieve 3.4 defects per million opportunities, or 99.9997% of quality level (Thomsett, 2005).

Sigma Shift

In the six-sigma approach, the value 3.4 DPMO (Defect Per Million of Opportunities) is followed with a footnote or an asterisk, and the related fine print is typically ignored by readers. In six-sigma, this 3.4 DPMO is presumed to be the long-term process performance after the occurrence of a *sigma shift*. The *sigma shift* is a 1.5 sigma difference from 6 to 4.5 sigma performance. This underlying assumption is that short-term performance (of say 6 sigma) is really 4.5 sigma in the long-term as entropy sets in (Wheeler, 2004).

It is to be noted that in six-sigma the parts-per-million values are dependent upon the assumption that there is a 1.5 standard deviation shift.

Figure 9.1 ***Sigma shift***

As a result of applying the proposed quality maturity model, we will have one of five quality maturity levels based on the sigma values. Figure 9.2 shows the five maturity levels selected to rank the quality of the software product. This maturity scale can be applied in turn to the three different viewpoints; that is, not only for the quality of the whole software product, but also for the life cycle stage quality (i.e. internal quality, external quality and quality in-use of the software product) and for the software product quality characteristics.

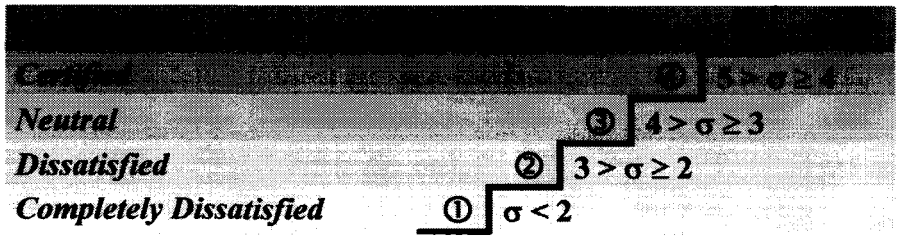

Figure 9.2 ***Quality maturity levels***

9.2.2 A Quantitative approach to product quality

To make communication with both customers and managers easier, it is more convenient to refer to a quality characteristic by a single value rather than a number of values (since each characteristic is represented by a number of related measures). To produce this single value, an organization needs to assign different weights to the individual quality views (characteristics, subcharacteristics or measures) on a software product. A number of techniques exist for combining multiple values. For instance, Sauro and Kindlund (2005b) have introduced a method to integrate three usability measures (effectiveness, efficiency and satisfaction) into a single value that measures and communicates usability as a quality characteristic of a software product; to do so, the principal component analysis (PCA) technique is used to find the weight of each of the individual usability measures using historical data.

In our proposed model, in order to produce single values for the life-cycle-stage, characteristic or subcharacteristic quality levels, we assume initially that all measures have an equal weight (contribution) in the computation of the subcharacteristic quality level (and characteristic quality level, in the case of quality in-use), and each subcharacteristic has an equal weight in the calculation of the characteristic quality level.

This single value could be easily converted to a sigma value using the NORMSINV function, which is a Microsoft Excel function that delivers the inverse of the cumulative standardized normal distribution (Sauro and Kindlund, 2005a).

Using a single sigma without shifting is not sufficient for our model since individual integrity levels do not require the same sigma value for the corresponding quality level within different integrity levels.

The sigma shift is used as a technique to increase or decrease the corresponding quality levels. In other words, the sigma shift is the value which needs to be added to the sigma value to make a gap between the quality levels; for instance a higher sigma shift will produce a higher gap between the quality levels. For example, without sigma shift, the corresponding quality level for the one sigma is 84.14%, but when the sigma is shifted by 1.5 the quality level becomes 99.38%.

In Table 9.1, the sigma values have been calculated and shifted based on the quality levels and on the software integrity levels respectively. Again, the NORMSINV function is used to produce the sigma values for all the quality levels, and the integrity levels have been used to determine the value of the sigma shift.

Table 9.1

The sigma ranges based on the Quality Level and the Software Integrity Level

Software Integrity Levels and Risk Classes					
5	**4**	**3**	**2**	**1**	**0**
Very High	**High**	**Intermediate**	**Low**	**Trivial**	**None**

Quality Levels (QL) for each Sigma Shift						**Assigned Sigma Ranges**
Zero Sigma Shift	**1.5 Sigma Shift**	**2.0 Sigma Shift**	**2.5 Sigma Shift**	**3.0 Sigma Shift**	**3.5 Sigma Shift**	
QL≥99.99997%	QL≥99.976%	QL≥99.865%	QL≥99.379%	QL≥97.724%	QL≥93.319%	**σ ≥ 5**
QL<99.99997% and QL≥99.996%	QL<99.976% and QL≥99.379%	QL<99.865 and QL≥97.724%	QL<99.379% and QL≥93.319%	QL<97.724% and QL≥84.134%	QL<93.319% and QL≥69.146%	**5 > σ ≥ 4**
QL<99.996% and QL≥99.865%	QL<99.379% and QL≥93.319%	QL<97.724% and QL≥84.134%	QL<93.319% and QL≥69.146%	QL<84.134% and QL≥50%	QL<69.146% and QL≥30.853%	**4 > σ ≥ 3**
QL<99.865% and QL≥97.724%	QL< 93.319% and QL≥69.146%	QL<84.134% and QL≥50%	QL<69.146% and QL≥30.853%	QL<50% and QL≥15.865%	QL<30.853% and QL≥6.680%	**3 > σ ≥ 2**
QL<97.724%	QL<69.146%	QL<50%	QL<30.853%	QL<15.865%	QL<6.680%	**σ < 2**

Table 9.1 includes three parts:

- The upper part: the software integrity levels and risk classes.
- The lower part: the quality levels (QL) for each sigma shift.
- The right-hand part: the assigned sigma ranges.

Using the first part and the second part of Table 9.1, we can get the assigned sigma range value from the third part.

In Table 9.1 we identified six values for the sigma shift (i.e. 0, 1.5, 2.0, 2.5, 3.0 and 3.5) to determine the various gaps between the quality levels values; the higher sigma shift will produce a higher gap between the quality levels. For example, we used zero sigma shift with integrity level five because the software products with this integrity level are very sensitive to the quality; therefore, the use of this sigma shift (zero sigma shift) produced ranges of quality levels with small gaps, that is, from 97.724% to 99.99997% (see Table 9.1 – zero sigma shift column). While the 1.5 sigma shift is used with the integrity level four because the software products with this integrity level are less sensitive to the quality than the software products with integrity level five, and the produced ranges of quality levels have larger gaps, that is, from 69.146% to 99.976% (see Table 9.1 – 1.5 sigma shift column). Table 9.1 will be used as reference to assess the product quality maturity level (see later Chapter 10).

Table 9.2 illustrates next an example of a software product with a quality level of 99% (using the NORMSINV Excel function, the original sigma (without sigma shift) value for 99% is 2.36σ). However, for evaluating its quality maturity level, we have six cases based on its integrity level:

Table 9.2

Example of a Quality Level with its different Maturity Levels

Quality Level	Original Sigma Value (OSV)	Integrity Level	Sigma Shift Value (SSV)	Shifted Sigma Value (OSV+SSV)	Corresponding Maturity Level
99%	2.32σ	5	0.0	(2.32+0.0) = 2.32σ	2
		4	1.5	(2.32+1.5) = 3.82σ	3
		3	2.0	(2.32+2.0) = 4.32σ	4
		2	2.5	(2.32+2.5) = 4.82σ	4
		1	3.0	(2.32+3.0) = 5.32σ	5
		0	3.5	(2.32+3.5) = 5.82σ	5

1. If the integrity level = 5
   - Using Table 9.1, the corresponding sigma shift = 0.0 sigma shift.
   - The shifted sigma value = 2.32σ + 0.0σ = 2.32σ.
   - Using Figure 9.2, the corresponding maturity level = 2.

2. If the integrity level = 4
   - Using Table 9.1, the corresponding sigma shift = 1.5 sigma shift.
   - The shifted sigma value = 2.32σ + 1.5σ = 3.82σ.
   - Using Figure 9.2, the corresponding maturity level = 3.

3. If the integrity level = 3
   - Using Table 9.1, the corresponding sigma shift = 2.0 sigma shift.
   - The shifted sigma value = 2.32σ + 2.0σ = 4.32σ.
   - Using Figure 9.2, the corresponding maturity level = 4.

4. If the integrity level = 2
   - Using Table 9.1, the corresponding sigma shift = 2.5 sigma shift.
   - The shifted sigma value = 2.32σ + 2.5σ = 4.82σ.
   - Using Figure 9.2, the corresponding maturity level = 4.

5. If the integrity level = 1
    - Using Table 9.1, the corresponding sigma shift = 3.0 sigma shift.
    - The shifted sigma value = $2.32\sigma + 3.0\sigma = 5.32\sigma$.
    - Using Figure 9.2, the corresponding maturity level = 5.

6. If the integrity level = 0
    - Using Table 9.1, the corresponding sigma shift = 3.5 sigma shift.
    - The shifted sigma value = $2.32\sigma + 3.5\sigma = 5.82\sigma$.
    - Using Figure 9.2, the corresponding maturity level = 5.

In other words, Table 9.1 can be directly used to get the assigned sigma range for a quality level when the integrity level is known for the software product benign assessed. Therefore, the same quality level for a specific software product could lead to different maturity levels of that software product based on its integrity level.

9.3 The Software Product Quality Maturity Model – SPQ^{MM}

The quality maturity model of software products is built to be used from three different viewpoints:

1. The Whole Software Product Quality Maturity Model.
2. The Software Product Life-Cycle-Stage Quality Maturity Models.
3. The Software Product Quality Characteristic Maturity Model.

Figure 9.3 illustrates the relationships between the above-mentioned three viewpoints of the quality maturity model.

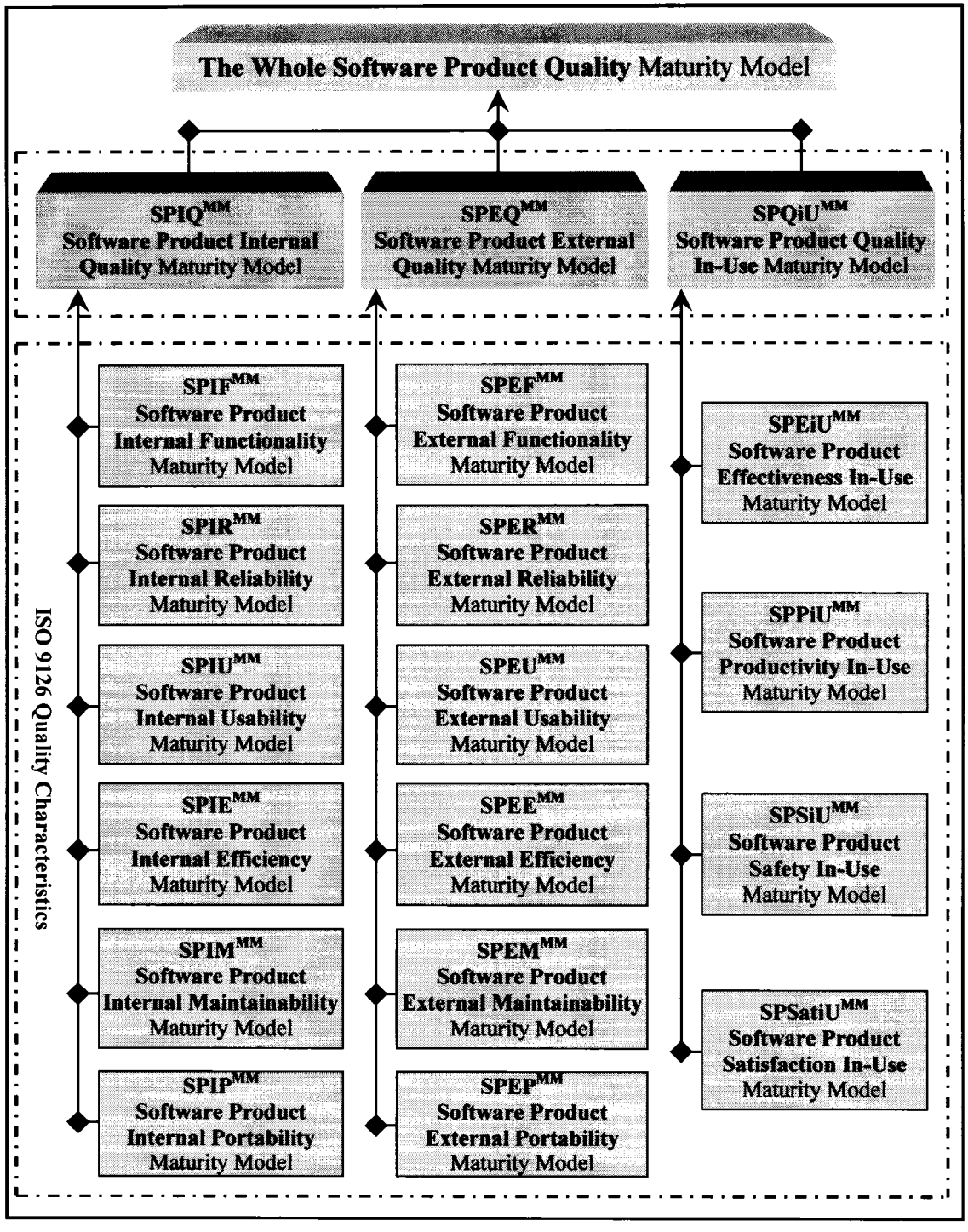

Figure 9.3 ***The contents of the Software Product Quality Maturity Model***

9.3.1 The whole software product quality maturity model

The whole software product view of our model (Software Product Quality Maturity Model – SPQ^{MM}) is illustrated in Figure 9.4, and shows the components that should be computed to achieve the quality maturity level of the whole software product. The resulting maturity level is derived from the quality levels of the software product stages (internal, external and in-use).

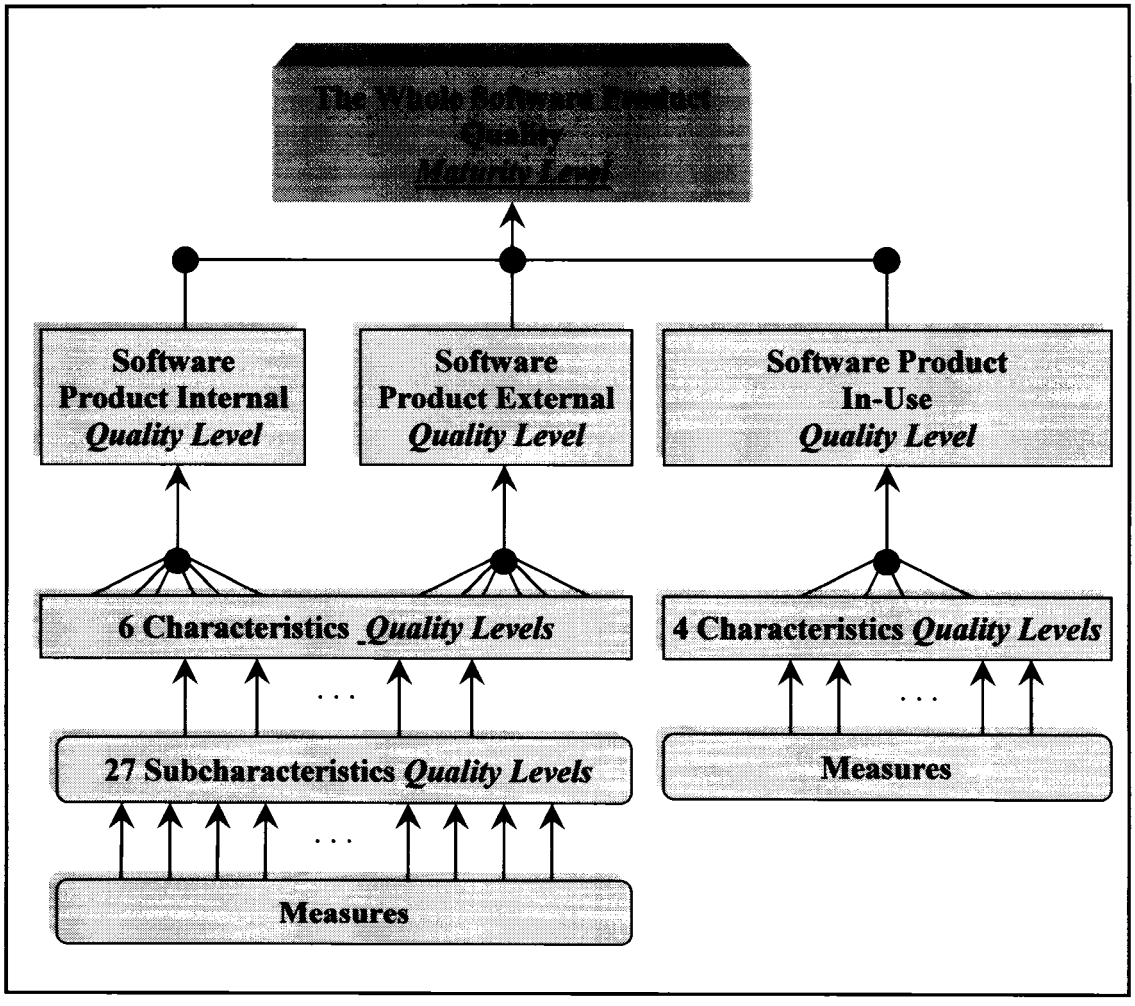

Figure 9.4 *The components of the Quality Maturity Level for the whole software product*

9.3.2 The life-cycle stages quality maturity model

The following are the three types of Quality Maturity Models from the life-cycle-stage viewpoint:

1. Software Product Internal Quality Maturity Model – $SPIQ^{MM}$.
2. Software Product External Quality Maturity Model – $SPEQ^{MM}$.
3. Software Product Quality-in-Use Maturity Model – $SPQiU^{MM}$.

Each of the above life-cycle stage-view maturity models is based on the selected ISO 9126 software product quality characteristics. The software product quality characteristics and subcharacteristics should be selected based on the type of software (e.g. embedded software, real-time software, application software, system software, etc.) to be evaluated. Figure 9.5 shows the components of the software product life-cycle-stage quality maturity model.

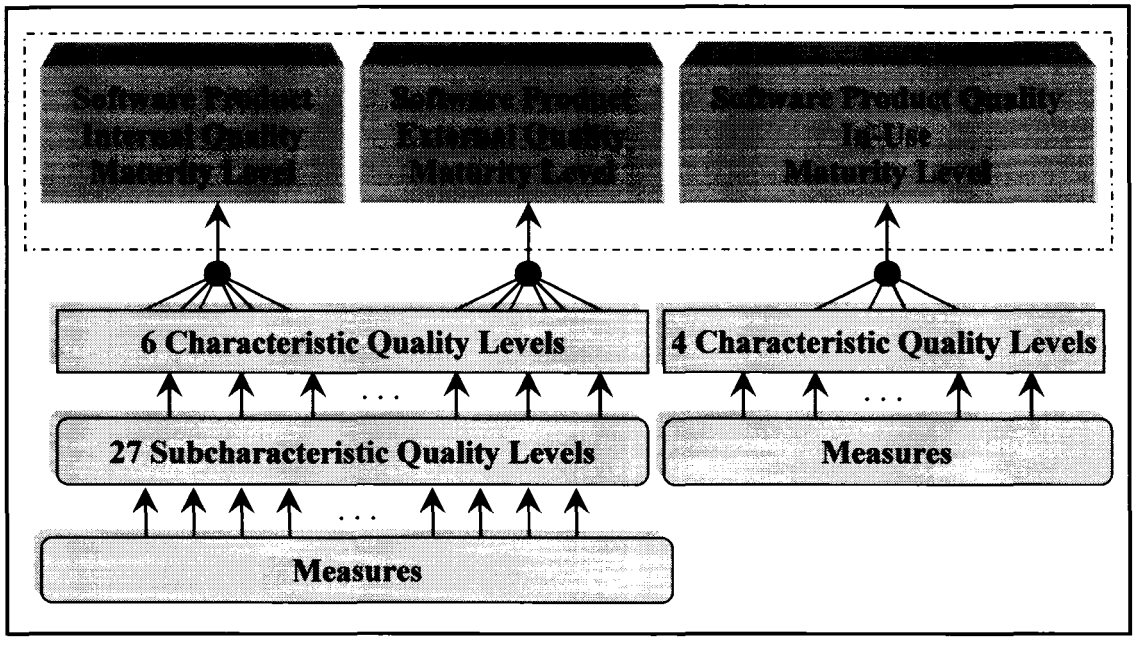

Figure 9.5 ***Components of the Quality Maturity Levels from the life-cycle-stages viewpoint***

9.3.3 The characteristics quality maturity models

The characteristics view of our quality maturity model (e.g. Software Product Internal Functionality Maturity Level) is partially based on the ISO 9126 quality model; see Figures 2.2 and 2.3 in Chapter 2 for the complete lists of the ISO 9126 quality characteristics. Figure 9.6 illustrates the structure of the characteristics-view maturity models.

Only the quality maturity levels of the required characteristic will be evaluated using these models, based on the quality level of the selected subcharacteristics in the case of external and internal software products, and based directly on selected measures in the case of an in-use software product.

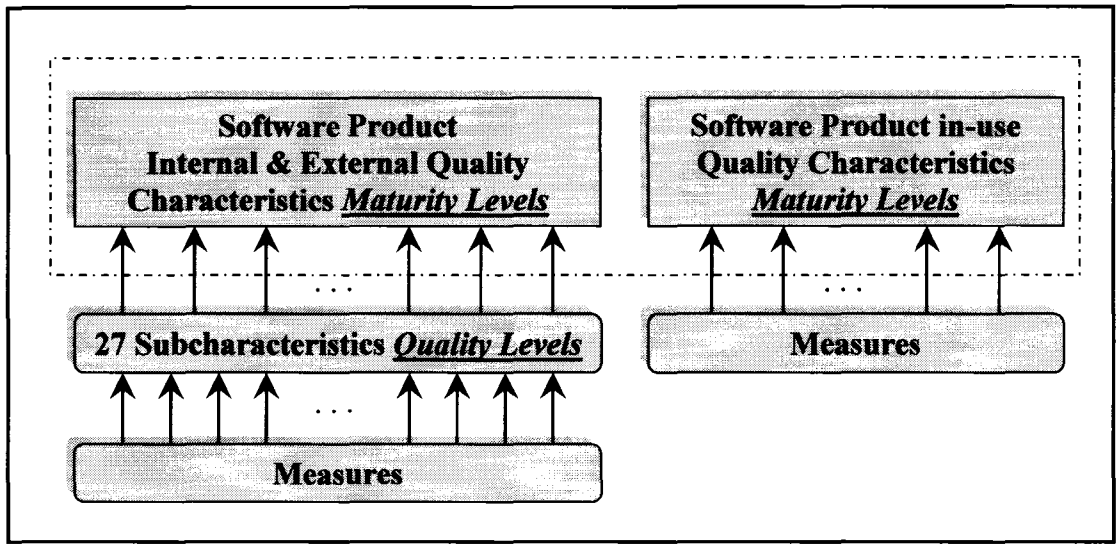

Figure 9.6 ***Components of the Quality Maturity Levels from the internal, external and in-use characteristics viewpoint of the software product***

9.4 Summary

In this chapter, we have presented the design of the proposed software product quality maturity model which can be used from three different points of view:

1. Software Product Internal Quality Maturity Model – $SPIQ^{MM}$.
2. Software Product External Quality Maturity Model – $SPEQ^{MM}$.
3. Software Product Quality-in-Use Maturity Model – $SPQiU^{MM}$.

In addition, we show the detailed mapping of the related concepts such as the ISO 9126 quality model, the software integrity levels, and the sigma concepts to produce such a quality maturity model.

In the next chapter, we will draw up the different steps need to be followed in order to compute a quality maturity level for any software product.

CHAPTER 10

DETERMINING THE QUALITY MATURITY LEVELS USING THE SOFTWARE PRODUCT QUALITY MATURITY MODEL – SPQ^{MM}

10.1 Introduction

We introduced the structure of our software product quality maturity model in Chapter 9. In this chapter, we describe the detailed steps which should be followed to get the maturity level of a specific software product. These steps can be summarized as follows:

1. Determine the software integrity level.
2. Select the required characteristics, subcharacteristics, and derived measures.
3. Identify the required base measures for each of the selected subcharacteristics and characteristics (in the case of in-use quality).
4. Compute the selected derived measures.
5. Compute the quality level of the selected software product quality characteristics.
6. Identify the sigma value and the maturity level.

10.2 Software Integrity Level determination

The software integrity level is used in our models to classify the software products. Therefore, the software products with high integrity levels are very sensitive to quality. In order to identify the integrity level of a software product, we first need to know the consequences of any failure of that software product.

Table 10.1 illustrates the definitions of the consequences that have been expanded into six consequences instead of the four in the IEEE standard for software verification and validation (IEEE, 1998).

Table 10.1

Definitions of expanded consequences

Consequences	Definitions
Catastrophic	Loss of human life, complete mission failure, loss of system security and safety, or extensive financial or social loss
Critical	Major and permanent injury, partial loss of mission, major system damage, or major financial or social loss
Severe	Severe injury or illness, degradation of secondary mission, or some financial or social loss
Marginal	Minor injury or illness
Minor	Minor impact on system performance or operator inconvenience
None	No impact

In addition, the occurrence of those consequences is very important for determining the integrity level. The frequency of their occurrence can be estimated using the indicative frequency (IFreq) of previous and similar software products (see Table 10.2).

Table 10.2

Indicative frequency for each occurrence
(ISO, 1998b)

Occurrence	Indicative Frequency (IFreq) (per year)
Frequent	$IFreq > 1$
Probable	$0.1 < IFreq \leq 1$
Occasional	$0.01 < IFreq \leq 0.1$
Remote	$0.0001 < IFreq \leq 0.01$
Improbable	$0.000001 < IFreq \leq 0.0001$
Incredible	$IFreq < 0.000001$

Based on the consequences and their occurrence, we have classified the software integrity levels into an ordered scale of ranging from zero to five – see Table 10.3. For example, if the consequences and their occurrence are 'severe' and 'occasional' respectively, then the integrity level of that software is three.

Table 10.3

Determination of the Software Integrity Level using the consequences and their occurrence

Consequence	Occurrence					
	Frequent	**Probable**	**Occasional**	**Remote**	**Improbable**	**Incredible**
Catastrophic	5	5	5	4	3	2
Critical	5	5	4	3	2	1
Severe	4	4	3	2	1	1
Marginal	3	3	2	2	0	0
Minor	2	2	1	1	0	0
None	0	0	0	0	0	0

10.3 Selection of the required characteristics, subcharacteristics and measures

In order to evaluate the maturity level of a software product, it is necessary to identify which characteristics are most closely related to this software product. Therefore, the characteristics that must be taken into account should be selected based on the type of software product (e.g. embedded software, real-time software, etc.). In addition, the subcharacteristics (in the case of internal and external software products) and the measures also need to be identified.

10.4 Identification of the required Base Measures for each of the selected characteristics

We have already classified the ISO 9126 measures into base or derived measures based on the ISO 15939 and ISO VIM, and have provided a list of proposed base measures in ISO 9126, parts 2, 3 and 4 (see Annex IV). This classification helps in determining which measures should be collected before starting the measurement process, taking into account that most of the ISO 9126 measures are derived and could not be computed without first identifying and collecting the base measures. In addition, a cross-reference table of base measure usage provided in Annex V, identifies for each subcharacteristic (or characteristic, in the case of quality in-use), which base measures should be collected to compute the related derived measures for that subcharacteristic or characteristic. In particular, these lists can help in:

- Identifying, selecting and collecting a base measure (once), and then reusing this base measure to evaluate a number of derived measures;
- Learning which base measures are required to evaluate specific software quality attributes (characteristics and subcharacteristics).

Now the derived measures could be computed easily since we have all the required base measures.

10.5 Computing the Quality Levels of the selected software product quality characteristics

At this point, the characteristics, subcharacteristics and measures have been selected and the base measures identified and collected. In order to calculate the quality maturity level of the software product, the quality levels of the selected characteristics must be computed.

Figure 10.1 shows the steps to be followed to compute the quality level of any of the selected internal or external characteristics, and, in Figure 10.2, of the quality-in-use characteristics.

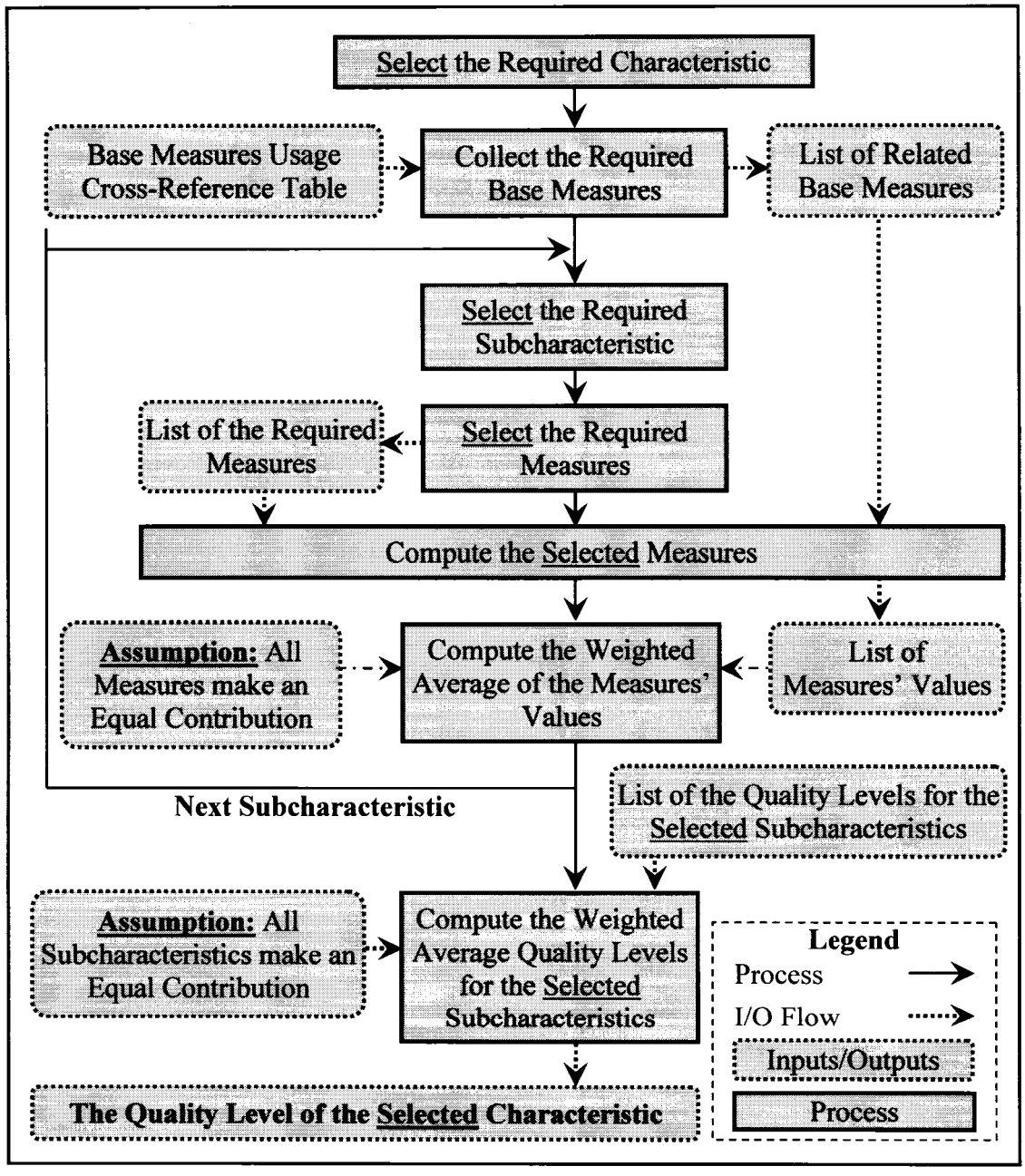

Figure 10.1 ***Computation of the Quality Level for each of the ISO 9126 internal and external characteristics***

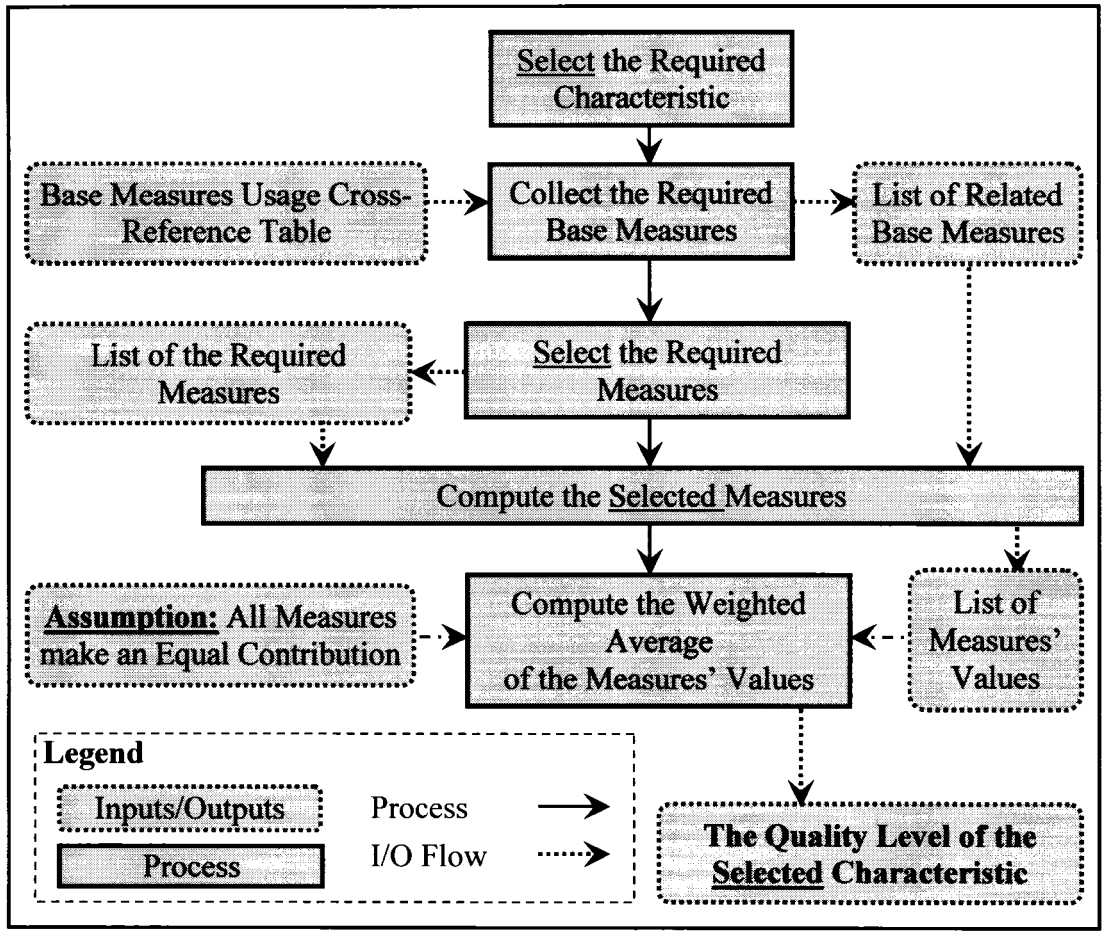

Figure 10.2 ***Computation of the Quality Level for each of the software product quality in-use characteristics***

The following equations can be used to calculate the quality levels of the selected subcharacteristics, characteristics and internal / external software product (internal / external life cycle phase):

$$SQL = \frac{\left(\sum_{i=1}^{n} qm_i\right)}{n} \tag{10.1}$$

$$CQL = \frac{\left(\sum_{i=1}^{m} SQL_i\right)}{m} \tag{10.2}$$

$$QL = \frac{\left(\sum_{i=1}^{L} CQL_i\right)}{L} \tag{10.3}$$

where:

- *SQL* is the Subcharacteristic Quality Level,
- *CQL* is the Characteristic Quality Level,
- *QL* is the Internal/External/in-Use Quality Level,
- *qm* is the selected Quality Measure,
- *n* is the number of selected measures of that subcharacteristic,
- *m* is the number of selected subcharacteristics of that characteristic, and
- *L* is the number of selected internal/external/in-use characteristics of that software product.

In the case of the in-use software product, there are no subcharacteristics for each characteristic, but rather a number of related measures. Equation (10.4) can be used to calculate the quality levels of the characteristics, whereas the quality level of the in-use software product (in-use stage) can be calculated using equation (10.3):

$$CQL = \frac{\left(\sum_{i=1}^{m} qm_i\right)}{m} \tag{10.4}$$

where *m* is the number of selected measures of that characteristic.

For the whole software product, equation (10.5) can be used to achieve a single value of the quality level of the whole software product:

$$WQL = \frac{(IQL + EQL + iUQL)}{3} \tag{10.5}$$

where:

- *WQL* is the quality level of the whole software product, including the quality levels of all three stages of the software product,
- *IQL* is the internal quality level,
- *EQL* is the external quality level, and
- *iUQL* is the in-use quality level of the software product.

For equations 10.1 to 10.5, we have made an assumption that all measures, subcharacteristics quality levels, characteristics quality levels and stage quality levels make the same contribution (have the same weight) in the calculation of the corresponding subcharacteristic quality level, characteristic quality level, life-cycle stage (internal, external or in-use) quality level and whole software product quality level respectively. Of course, if an organization wishes to assign different weights, relevant techniques must be used to integrate them.

10.6 Identifying the sigma value and the maturity level

In the previous sections, instructions have been provided for calculating the quality level for any of the three viewpoints (characteristic, stage, whole software product), and how to determine the software integrity level.

It is easy to achieve the corresponding sigma value for the quality level of any of the three viewpoints. For example, if the following information about a specific software product is available,

- the Software Integrity Level = 2, and
- the Quality Level of the External Software Product = 80%.

Then, using Table 9.1, the sigma value will be $4 > \sigma \geq 3$. Moreover, using Figure 9.2, which illustrates the quality maturity levels, the Software Product External Quality Maturity level is three (neutral).

10.7 Discussion

Evaluation of the quality of any software product is very important, since poor quality in a software product (particularly in sensitive systems, such as real-time systems, control systems, etc.) may lead to loss of human life, permanent injury, mission failure or financial loss.

Several capability and maturity models have been designed for the software process. For example, the CMMi, the Software Maintenance Maturity Model (S^{3M}) and the Testing Maturity Model (TMM). Up to now, there had been no corresponding product maturity model for assessing the quality of software products.

The design of a software product maturity model to assess the quality of a software product therefore represented a challenge. In Chapter 9 and 10, we presented a product quality assessment model based on ISO 9126 and ISO 15026. Specifically, we discussed the structure of the quality maturity model from the following three distinct points of view:

- The whole software product,
- The software product life-cycle stage (internal, external and in-use), and
- The software product quality.

In the literature, there are many quality models and hundreds of measures that deal with the software product. Selecting which to use for evaluating the software product quality is a challenge. To address this diversity of alternatives, the ISO has come up with a consensus on a quality model and an inventory of measures to evaluate the quality of a

software product (ISO 9126). However, using individual ISO 9126 measures to evaluate the quality of a software product is also a challenge, since we will obtain a set of numbers that reflects the quality level of each measure for each quality characteristic. Moreover, it is difficult on the one hand to interpret these numbers, and, on the other hand, to integrate them into a decision-making model. Therefore, a single number which will reflect the quality of a characteristic is sorely needed.

Also, we have combined the set of quality measures into a single value for each quality characteristic by assuming that all the measures for a single quality characteristic have an equal weight in the computation of a single value for that quality characteristic (they all make an equal contribution), yielding a quality level for that quality characteristic. The resulting quality level is then transformed into a sigma value positioned within a quality maturity level.

Our Software Product Quality Maturity Model can be used to determine the maturity of the quality of a software product. Specifically, it can be used to:

- Certify a quality maturity level for a new software product, which could help promote it on the market;
- Benchmark existing software products to assist in making a selection based on their quality maturity level;
- Assess the quality of the software product during the development life-cycle (i.e. internally, externally and in-use) to investigate the relationships between the three stages and to find any weaknesses in order to improve the software product;
- Assess the maturity of the internal quality of a software product to be reused in other software products.
- Compare the maturity levels of the life-cycle-stage quality (i.e. internal, external and in-use).

The software product quality maturity model discussed in Chapters 9 and 10 are limited to the ISO 9126 quality model and its set of measures. Another limitation is that the results yielded by the quality maturity model discussed are initially based on the assumption of the equal weight of all measures, all characteristics and all subcharacteristics. To avoid making this assumption, an organization can apply the PCA (Principal Component Analysis) statistical technique to a large set of historical data to find a corresponding weight for each measure, characteristic or subcharacteristic. Alternatively, an organization can assigned its own weights using for instance the analytical hierarchy process (AHP) (Koscianski and Costa, 1999) technique and then use relevant techniques to combined them into aggregated values at higher levels, such as done for example in the QEST multi-dimensional models for quality and performance (Buglione and Abran, 1999; 2002).

CONCLUSION AND FUTURE WORK

This thesis had two main research objectives: The first objective was the building of an understanding of the designs and definition of the current proposed measures for software product quality to determine their strengths and weaknesses. The second research objective was to build a maturity model based on software engineering standards rather than individual models of software product quality.

Research objective 1: Key research contributions and limitations

To meet the first research objective, we used the Habra *et al.* (2004) software measurement analysis framework (which contains some of the metrology concepts) for the analysis of, some measures quoted in ISO 9126, including the Halstead's measures.

The key findings of the analysis of the design of Halstead's measures are:

- Halstead has not explicitly provided a clear and complete counting strategy to distinguish between the operators and the operands in a given program or algorithm. This has led researchers to come up with different counting strategies and, correspondingly, with different measurement results for the same measures and for the same program or algorithm.
- There are problems with the units of measurement for both the left-hand and the right-hand sides of most of Halstead's equations.
- The implementation of the measurement functions of Halstead's measures has been interpreted in different ways than the goals specified by Halstead in their designs. For example, the program length (N) has been interpreted as a measure of program complexity, which is a different characteristic of a program.
- Based on ISO 15939 (ISO, 2002) and the international vocabulary of basic and general terms in metrology (VIM) (ISO, 1993), Halstead's measures can be

classified as six based measures (n_1, n_2, N_1, N_2, n_1* and n_2*) and ten derived measures (equations (6.1) to (6.10)).

- Halstead has not explicitly provided a clear and complete counting strategy to distinguish between the operators and the operands in a given program or algorithm. This has led researchers to come up with different counting strategies and, correspondingly, with different measurement results for the same measures and for the same program or algorithm.

This analysis of the Halstead's measures with the Habra *et al.* (2004) framework has demonstrated the usefulness and contributions of this framework.

Next, the ISO 9126-4 measures were analyzed to understand their designs and definitions since they were to be used in our software product quality maturity model SPQ^{MM}. In addition, since the current ISO 9126 will be soon replaced by the ISO 2502n series of standards, we have analysed two of these upcoming standards, that is, ISO 25020 and ISO 25021. This analysis demonstrated that some of the ISO 9126 measures are not well defined. From the analysis of the quality in-use measures (ISO 9126), the summary of the key findings are:

- The ISO 9126 measures need to be classified based on the metrology concepts into base and derived measures. This classification will make it easy to determine which measures should be collected (base measures) in order to be used in the computing of derived measures.
- The measurement units of some derived measures are ambiguous since they depend on other measures with unknown units.
- Based on the used metrology concepts, none of the 'quality in-use' metrics in ISO 9126-4 refers to any 'system of units', 'coherent (derived) unit', 'coherent system of units', 'international system of units (SI)', 'off-system units', 'multiple of a unit', 'submultiple of a unit', 'true values', 'conventional true values', and

'numerical values'. Furthermore, neither of the base nor the derived quantities have symbols for their measurement units, except for the 'task time'.

- It can be also noticed that the ranges of the results of many of the derived 'metrics' in ISO 9126-4 are between zero and one. Therefore, it is easy to convert them into percentage values. However, from our point of view, it will be more understandable if these results were ranked into ordinal categories. For example, for the 'task completion', if the percentage result is 100% then the completion of the task could be categorized as 'excellent', if the result is 80% then the completion of the task could be categorized as 'very good', and so on.

From the analyses of the ISO 25020 and ISO 25021, some additional issues have been raised concerning three new concepts proposed in ISO 25020 and ISO 25021; that is, 'quality measure element categories', 'quality measure elements' and 'quality measures'. The following is a summary of the harmonization issues identified:

- What is referred to as a 'quality measure element' corresponds to the classic concept of 'base measure' in ISO 15939 and what is referred to as 'software quality measure' corresponds to the classic concept of 'derived measure' in ISO 15939.
- A large number of the base measures proposed in ISO 25021 are counts of entities rather than measures per se with required metrological characteristics such as: unit, scale, dimension, measurement method, measurement procedures, etc.
- In ISO 25021, in some instances, like 'product size' for example, there is no reference to other existing ISO standards for software size, such as ISO 19761, etc.
- There are a number of claims that the proposed base measures are 'objective', while they are obviously derived from a manual process without precisely documented measurement procedures, thereby leaving much to the measurer's judgment.
- The set of quality measures documented in ISO 25021 represents only a limited subset of the base measures within ISO 9126, parts 2 to 4; the rationale for inclusion or exclusion is not documented, and this selection of quality measures

does not allow coverage of the full spectrum of quality characteristics and subcharacteristics in ISO 9126, parts 2 to 4; again, the rationale for inclusion or exclusion is not documented in ISO 25021.

To tackle these shortcomings noted above, the ISO group working on the next version of the ISO 9126 series should:

- Ensure that the terminology on software product quality measurement is fully aligned with the classic measurement terminology in the sciences and in engineering;
- Provide full coverage of the base measures for all three ISO models of software quality;
- Provide improved documentation of the base measures using criteria from metrology;
- Provide clear mapping and traceability of the new ISO 25000 documents to the ISO 15939 Information Model.

The other research contributions which come from meeting the first research objective are:

- Verification of the usefulness of the measurement analysis framework which has been introduced by Habra *et al.* (2004); this has been illustrated by applying it to the Halstead's measures, and the findings of various weaknesses in their designs from a metrology perspective.
- Verification of the ISO 9126-4 measures against the metrology concepts.
- Identification of some of the harmonization issues arising with the addition of new documents like the ISO 25020 and ISO 25021, in particular, with respect to previously published measurement standards for software engineering, such as ISO 9126, ISO 15939, ISO 14143-1 and ISO 19761.
- Identification of the list of base measures which could be used to evaluate the ISO 9126 part-2, part-3, and part-4 derived measures. For each of these base measures,

the measurement unit has been identified. In addition, a cross-reference list between the base measure and the related characteristics/subcharacteristics has been built.

In summary, with respect to the first research goal, the analysis in this thesis has identified weaknesses in some of the new terms proposed by ISO, and we have proposed ways to address them by using the ISO 15939 measurement information model on software measurement process: the use of predefined terms such as 'base measure' and 'derived measure', as well as the proper mapping to the Measurement Information Model in well-developed standards like ISO 15939, and the International Vocabulary of Basic and General Terms in Metrology (ISO VIM) is more useful than the weakly defined terms introduced in the proposed new version of the ISO 25000 series.

Research objective 2: Key research contributions and limitations

The second research objective was to build a maturity model based on software engineering standards rather than individual models of software product quality. Thus, the design of a software product maturity model to assess the quality of a software product, therefore, represented a new challenge in software engineering. In this thesis, we presented a product quality maturity model based on a set of ISO/IEEE standards, metrology, and sigma concepts. Specifically, we built the structure of the quality maturity model from the following three distinct points of view:

- The whole software product.
- The software product life cycle stage (internal, external and in-use).
- The software product characteristic.

This Software Product Quality Maturity Model can be used to determine the maturity of the quality of a software product. Specifically, it can be used to:

- Certify a quality maturity level for a new software product, which could help promote it on the market;

- Benchmark two existing software products to assist in making a selection based on their quality maturity level;
- Assess the quality of the software product during the development life cycle (i.e. internally, externally and in-use) to investigate the relationships between the three stages and to find any weaknesses in order to improve the software product;
- Assess the maturity of the internal quality of a software product to be reused in other software products.
- Compare the maturity levels of the life cycle stage quality (i.e. internal, external and in-use).

The following are the main research contributions which come from meeting the second research objective, that is, the proposed software product quality maturity model:

- The design of a Software Product Quality Maturity Model (SPQ^{MM}) that can be used from three different viewpoints, that is: the quality characteristic view, the life cycle stage view, and the whole software product view.
- For this quality maturity model, the sigma concepts have been incorporated into the software integrity levels to obtain the appropriate values of the quality maturity levels of different types of software products (real-time software, application software, etc.).
- By applying the sigma concepts to the measured quality levels of the software product, by mapping the quality level to the corresponding sigma value; this sigma value can then be used to derive the maturity level. The following table shows a high level comparison between our proposed product quality maturity model and the other available product maturity models.

A Comparison between the software product maturity models

Characteristic	OSMM	Nastro Model	SPQ^{MM}
Quantitative Model		✓	✓
Based on a Quality Model			✓
Based on Software Engineering Standards			✓
Implemented during the Development Life-Cycle			✓
Ranked Maturity levels			✓
For any Software Product		✓	✓
Based on the Software Integrity levels			✓

Validation

For the first objective of this thesis, which is about the verification of the software measures, the findings from the following topics have all been validated by papers accepted and published at conferences, in journals and more recently by requests for reprints from the Institute of Chartered Financial Analysts of India (ICFAI) press:

- The measurement analysis framework - see Chapter 6.
- Halstead's measures referenced in ISO 9126 - see Chapter 6.
- The ISO 9126-4 measures - see Chapter 7.
- The new ISO 25020 and ISO 25021 standards since they will replace ISO 9126 - see Chapter 8.

For the second objective of this research, the validation of software maturity model is very time consuming and techniques to do so are not yet well developed and not yet in general use (Coallier, 2006). The proposed quality maturity model (SPQ^{MM}) itself needs to be validated over the future years since its validation is time consuming and will require much further research. For instance, for a partial addition, it would be useful to apply the SPQ^{MM} to two software products with known qualities to check if the result of the SPQ^{MM} model is aligned with the pre-known qualities or not.

Further limitations and future work

The 'data collection' and 'data preparation' for the proposed software product quality maturity model discussed in this thesis are currently limited to the ISO 9126 and SQuaRE quality models and their set of measures. However, since our analysis of these ISO 9126 measures from a metrology perspective has shown some weaknesses in these measures, further work will be required to enhance these inputs into our proposed quality maturity model.

Another current limitation is that the results yielded by the quality maturity models discussed are initially based on the assumption of the equal weight of all measures, all characteristics and all subcharacteristics. To avoid making this assumption, an organization can apply various techniques to taken into account non equal weights, such as the PCA (Principal Component Analysis) statistical technique to a large set of historical data to find a corresponding weight for each measure, characteristic or subcharacteristic.

In addition, the SPQ^{MM} is built to assess the quality maturity level of the software product itself, but does not include other related elements such as its documentation, supporting, integration and training.

The quality of a software product depends on many elements. However, sometimes the quality of the software product itself is high, but the low quality of the other elements (such as documentation, integration, supporting, or training) may lead to a poor quality of the whole software product from the customers point of view. Therefore, based on these elements, the following quality maturity models need to be developed:

- Software Product Documentation Quality Maturity Model – $SPDQ^{MM}$.
- Software Product Integration Quality Maturity Model – $SPIntegQ^{MM}$.

- Software Product Supporting Quality Maturity Model – $SPSQ^{MM}$.
- Software Product Training Quality Maturity Model – $SPTQ^{MM}$.

The quality maturity levels to be designed for the above maturity models, should be used with the ones which we described in this thesis to give a quality maturity model for the whole software product (which consists of the software itself, its documentation, its integration with the in-use environment such as the OS, its training, and its support) to be used with the ones which have been described in this thesis to assess the quality of the software product from the different elements.

In addition, the software product quality maturity model – which has been developed in this thesis – along with the above suggested quality maturity models could be automated, at least partially, by means of web-based tools to make the assessment procedure of the quality maturity levels faster and easier. Furthermore, the users of these web-based tools could choose for which elements they need to assess the quality maturity level or even assess the whole software product quality maturity level.

ANNEX I

'QUANTITIES AND UNITS' METROLOGY CONCEPTS IN THE SAFETY MEASURES

Metrology Concepts	ISO 9126-4 (Safety Measures)
System of Quantities:	
- Base Quantities:	1. Number of Usage Situations. 2. Number of People. 3. Number of Occurrences of Software Corruption. 4. Number of Occurrences of Economic Corruption 5. Number of Users.
- Derived Quantities:	6. User Health and Safety. $0 \leq$ User Health and Safety ≤ 1 7. Software Damage. $0 \leq$ Software Damage ≤ 1 8. Economic Damage. $0 \leq$ Economic Damage ≤ 1 9. Safety of People Affected by Use of the System. $0 \leq$ Safety of People Affected by Use of the System ≤ 1
Dimension of a Quantity:	
- Quantities of Dimension One (Dimensionless Quantities):	6. User Health and Safety. 7. Software Damage. 8. Economic Damage. 9. Safety of People Affected by Use of the System.
Units of Measurement:	
- Symbols of the Units:	- None.
- Systems of Units:	- None.
- Coherent (Derived) Units:	- None.
- Coherent System of Units:	- None.
- International System of Units (SI):	- None.
- Base Units:	1. Usage Situation. 2. People. 3. Software Corruption. 4. Economic Corruption. 5. User.
- Derived Units:	6. (1-User/User) = % 7. (1-Software Corruption/Usage Situation). 8. (1-Economic Damage/Usage Situation). 9. (1- People/People) = %.
- Off-System Units:	- None.
- Multiple of a Unit:	- None.
- Submultiple of a Unit:	- None.
Value of a Quantity:	
- True Values:	- None.
- Conventional True Values:	- None.
- Numerical Values:	- Results of applying the measurement functions of the above base and derived quantities.
- Conventional Reference Scales:	- None.

ANNEX II

'QUANTITIES AND UNITS' METROLOGY CONCEPTS IN THE SATISFACTION MEASURES

Metrology Concepts	ISO 9126-4 (Satisfaction Measures)
System of Quantities:	
- Base Quantities:	1. Population Average. 2. Number of Responses. 3. Number of Times that Specific Software function / application / systems are used. 4. Number of Times that Specific Software function / application / systems are intended to be used.
- Derived Quantities:	5. Satisfaction Scale. Satisfaction Scale > 0 6. Satisfaction Questionnaire. 7. Discretionary Usage. $0 \leq$ Discretionary Usage ≤ 1
Dimension of a Quantity:	
- Quantities of Dimension One (Dimensionless Quantities):	6. Satisfaction Questionnaire. 7. Discretionary Usage.
Units of Measurement:	
- Symbols of the Units:	- None.
- Systems of Units:	- None.
- Coherent (Derived) Units:	- None.
- Coherent System of Units:	- None.
- International System of Units (SI):	- None.
- Base Units:	1. People. 2. Response. 3. Occurrence. 4. Occurrence.
- Derived Units:	5. Psychometric Scale (produced by questionnaire)/People. 6. Response/Response = %. 7. Occurrence/Occurrence = %.
- Off-System Units:	- None.
- Multiple of a Unit:	- None.
- Submultiple of a Unit:	- None.
Value of a Quantity:	
- True Values:	- None.
- Conventional True Values:	- None.
- Numerical Values:	- Results of applying the measurement functions of the above base and derived quantities.
- Conventional Reference Scales (Reference-Value Scales):	- None.

ANNEX III

INTERNAL QUALITY, EXTERNAL QUALITY, AND QUALITY IN-USE MEASURES IN THE ISO THE 9126 AND ISO 25021

Table III.1

External Quality Measures in the ISO 9126-2 and ISO 25021

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-2
Functionality	Accuracy	1	Computational accuracy	√	√
		2	Precision	√	√
		3	Accuracy relative to expectations		√
	Interoperability	4	Data exchangeability (Data format-based)	√	√
		5	Data exchangeability (User's success, attempt-based)		√
	Security	6	Access controllability	√	√
		7	Access auditability		√
		8	Data corruption prevention		√
	Suitability	9	Funcional implementation completeness	√	√
		10	Functional adequacy	√	√
		11	Functional implementation coverage	√	√
		12	Functional specification stability (volatility)		√
	Functionality Compliance	13	Functional compliance		√
		14	Interface standard compliance		√
Reliability	Maturity	15	Failure density against test cases	√	√
		16	Failure resolution	√	√
		17	Fault removal	√	√
		18	Mean time between failures (MTBF)	√	√
		19	Test maturity	√	√
		20	Estimated latent fault density	√	√
		21	Fault density	√	√
		22	Test coverage (Specified operation scenario testing coverage)		√
	Recoverability	23	Restartability	√	√
		24	Availability		√

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-2
		25	Mean down time		√
		26	Mean recovery time		√
		27	Restorability		√
		28	Restore effectiveness		√
	Fault Tolerance	29	Breakdown avoidance		√
		30	Failure avoidance		√
		31	Incorrect operation avoidance		√
	Reliability Compliance	32	Reliability compliance		√
Usability	Learnability	33	Effectiveness of the user documentation and/or help system	√	√
		34	Help accessibility	√	√
		35	Ease of function learning		√
		36	Ease of learning to perform a task in use		√
		37	Effectiveness of user documentation and/or help system in use		√
		38	Help frequency		√
	Operability	39	Physical accessibility	√	√
		40	Operational consistency in use		√
		41	Error correction		√
		42	Error correction in use		√
		43	Default value availability in use		√
		44	Message understandability in use		√
		45	Self-explanatory error messages		√
		46	Operational error recoverability in use		√
		47	Time between human error operations in use		√
		48	Undoability (User error correction)		√
		49	Customizability		√
		50	Operation procedure reduction		√
	Understandability	51	Completeness of description	√	√
		52	Function understandability	√	√
		53	Understandable input and output	√	√
		54	Demonstration accessibility		√
		55	Demonstration accessibility in use		√

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-2
		56	Demonstration effectiveness		√
		57	Evident functions		√
	Attractiveness	58	Attractive interaction		√
		59	Interface appearance customizability		√
	Usability Compliance	60	Usability compliance		√
Efficiency	Resource Utilization	61	I/O loading limits	√	√
		62	Maximum memory utilization	√	√
		63	Maximum transmission utilization	√	√
		64	Mean occurrence of transmission error	√	√
		65	I/O device utilization		√
		66	I/O-related errors		√
		67	Mean I/O fulfillment ratio		√
		68	User waiting time of I/O device utilization		√
		69	Mean occurrence of memory errors		√
		70	Ratio of memory error/time		√
		71	Media device utilization balancing		√
		72	Mean transmission error per time		√
		73	Transmission capacity utilization		√
	Time Behavior	74	Response time (Mean time to respond)	√	√
		75	Throughput (Mean amount of throughput)	√	√
		76	Turnaround time (Mean time for turnaround)	√	√
		77	Response time		√
		78	Response time (Worst case response time ratio)		√
		79	Throughput		√
		80	Throughput (Worst case throughput time ratio)		√
		81	Turnaround time		√
		82	Turnaround time (Worst case turnaround time ratio)		√
		83	Waiting time		√
	Efficiency Compliance	84	Efficiency compliance		√
Maintainability	Analyzability	85	Audit trail capability	√	√
		86	Diagnostic function support		√

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-2
		87	Failure analysis capability		√
		88	Failure analysis efficiency		√
		89	Status monitoring capability		√
	Changeability	90	Software change control capability	√	√
		91	Change cycle efficiency		√
		92	Change implementation elapsed time		√
		93	Modification complexity		√
		94	Parameterized modifiability		√
	Stability	95	Change success ratio		√
		96	Modification impact localization (Emerging failure after change)		√
	Testability	97	Availability of built-in test function		√
		98	Retest efficiency		√
		99	Test restartability		√
	Maintainability Compliance	100	Maintainability compliance		√
Portability	Adaptability	101	Adaptability of data structures	√	√
		102	Hardware environmental adaptability (adaptability to hardware devices and network facilities)	√	√
		103	System software environmental adaptability (adaptability to OS, network software and cooperated application software)	√	√
		104	Organizational environment adaptability (Organization adaptability to infrastructure of organization)		√
		105	Porting user-friendliness		√
	Installability	106	Ease of installation	√	√
		107	Ease of setup retry		√
	Coexistance Replaceability	108	Availability coexistence		√
	Replaceability	109	Continued use of data		√
		110	Function inclusiveness		√
		111	User support functional consistency		√
	Portability Compliance	112	Portability Compliance		√

Table III.2

Internal Quality Measures in ISO the 9126-3 and ISO 25021

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-3
Functionality	Accuracy	1	Computational accuracy	√	√
		2	Precision	√	√
	Interoperability	3	Data exchangeability (Data format-based)	√	√
		4	Interface consistency (protocol)		√
	Security	5	Access controllability	√	√
		6	Access auditability		√
		7	Data corruption prevention		√
		8	Data encryption		√
	Suitability	9	Funcional implementation completeness	√	√
		10	Functional adequacy	√	√
		11	Functional implementation coverage	√	√
		12	Functional specification stability (volatility)		√
	Functionality Compliance	13	Functional compliance		√
		14	Intersystem standard compliance		√
Reliability	Maturity	15	Fault removal	√	√
		16	Fault detection		√
		17	Test adequacy		√
	Recoverability	18	Restorability		√
		19	Restoration effictiveness		√
	Fault Tolerance	20	Failure avoidance		√
		21	Incorret operation avoidance		√
	Reliability Compliance	22	Reliability Compliance		√
Usability	Learnability	23	Completeness of user documentation and/or help facility	√	√
	Operability	24	Physical accessibility	√	√
		25	Input validity checking		√
		26	User operation cancellability		√
		27	User operation undoability		√
		28	Customizability		√
		29	Operation status monitoring capability		√
		30	Operational consistency		√
		31	Message clarity		√
		32	Interface element clarity		√

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-3
		33	Operational error recoverability		√
	Understandability	34	Completeness of description	√	√
		35	Function understandability	√	√
		36	Demonstration capability		√
		37	Evident functions		√
	Attractiveness	38	Attractive interaction		√
		39	User interface appearance customizability		√
	Usability Compliance	40	Usability Compliance		√
Efficiency	Resource Utilization	41	I/O utilization		√
		42	I/O utilization message density		√
		43	Memory utilization		√
		44	Memory utilization message density		√
		45	Transmission utilization		√
	Time Behavior	46	Response time		√
		47	Throughput time		√
		48	Turnaround time		√
	Efficiency Compliance	49	Efficiency compliance		√
Maintainability	Analyzability	50	Activity recording		√
		51	Readiness of diagnostic function		√
	Changeability	52	Change recordability		√
	Stability	53	Change impact		√
		54	Modification impact localization		√
	Testability	55	Completeness of built-in test function		√
		56	Autonomy of testability		√
		57	Test progress observability		√
	Maintainability Compliance	58	Maintainability compliance		√
Portability	Adaptability	59	Adaptability of data structures	√	√
		60	Hardware environmental adaptability (adaptability to hardware devices and network facilities)	√	√
		61	System software environmental adaptability (adaptability to OS, network software and cooperated application software)	√	√
		62	Organizational environment adaptability		√

Quality Characteristics	Quality Subcharacteristics		Measure Names	ISO DTR 25021	ISO 9126-3
		63	Porting user-friendliness		√
	Installability	64	Ease of setup retry		√
		65	Installation effort		√
		66	Installation flexibility		√
	Co-existence	67	Availability of coexistence		√
	Replaceability	68	Continued use of data		√
		69	Functional inclusiveness		√
	Portability Compliance	70	Portability compliance		√

Table III.3

Quality in-Use Measures in the ISO TR 9126-4 and ISO TR 25021

Quality Characteristics		Measure Names	ISO DTR 25021	ISO 9126-4
Effectiveness	1	Task effectiveness		√
	2	Task completion	√	√
	3	Error frequency		√
Productivity	4	Task time	√	√
	5	Task efficiency		√
	6	Economic productivity		√
	7	Productive proportion		√
	8	Relative user efficiency		√
Safety	9	User health and safety		√
	10	Safety of people affected by use of the system		√
	11	Economic damage		√
	12	Software damage		√
Satisfaction	13	Satisfaction scale		√
	14	Satisfaction questionnaire		√
	15	Discretionary usage		√

ANNEX IV

LISTS OF THE ISO 9126 BASE MEASURES

Table IV.1

The Base Measures in ISO 9126-2

External Base Measures		
	Measure Name	**Unit of Measurement**
1	Number of Functions	Function (number of)
2	Operation Time	Minute
3	Number of Inaccurate Computations Encountered by Users	Case (number of)
4	Total Number of Data Formats	Format (number of)
5	Number of Illegal Operations	Operation (number of)
6	Number of Items Requiring Compliance	Item (number of)
7	Number of Interfaces Requiring Compliance	Interface (number of)
8	Number of Faults	Fault (number of)
9	Number of Failures	Failure (number of)
10	Product Size	Byte
11	Number of Test Cases	Case (number of)
12	Number of Breakdowns	Breakdown (number of)
13	Time to Repair	Minute
14	Down Time	Minute
15	Number of Restarts	Restart (number of)
16	Number of Restoration Required	Restoration (number of)
17	Number of Tutorials	Tutorial (number of)
18	Number of I/O Data Items	Item (number of)
19	Ease of Function Learning	Minute
20	Number of Tasks	Task (number of)
21	Help Frequency	Access (number of)
22	Error Correction	Minute
23	Number of Screens or Forms	Screens (number of)

External Base Measures		
	Measure Name	**Unit of Measurement**
24	Number of User Errors or Changes	Error (number of)
25	Number of Attempts to Customize	Attempt (number of)
26	Total Number of Usability Compliance Items Specified	Item (number of)
27	Response Time	Second or Millisecond
28	Number of Evaluations	Evaluation (number of)
29	Turnaround Time	Second or Millisecond
30	Task Time	Minute
31	Number of I/O Related Errors	Error (number of)
32	User Waiting Time of I/O Device Utilization	Second or Millisecond
33	Number of Memory Related Errors	Error (number of)
34	Number of Transmission Related Errors	Error (number of)
35	Transmission Capacity	Byte
36	Number of Revised Versions	Version (number of)
37	Number of Resolved Failures	Failure (number of)
38	Porting User Friendliness	Minute

Table IV.2

The Base Measures in ISO 9126-3

Internal Base Measures		
	Measure Name	**Unit of Measurement**
1	Number of Functions	Function (number of)
2	Number of Data Items	Item (number of)
3	Number of Data Formats	Formats (number of)
4	Number of Interface Protocols	Protocol (number of)
5	Number of Access Types	Access-Type (number of)
6	Number of Access Controllability Requirements	Requirement (number of)
7	Number of Instances of Data Corruption	Instance (number of)
8	Number of Compliance Items	Item (number of)
9	Number of Interface Requiring Compliance	Interface (number of)
10	Number of Faults	Fault (number of)
11	Number of Test Cases	Test-Case (number of)
12	Number of Restoration	Requirement (number of)
13	Number of Input Items Which Could Check for Valid Data	Item (number of)
14	Number of Operations	Operation (number of)
15	Number of Messages Implemented	Message (number of)
16	Number of Interface Elements	Element (number of)
17	Response Time	Second or Millisecond
18	Turnaround Time	Second or Millisecond
19	I/O Utilization (Number of Buffers)	Buffer (number of)
20	Memory Utilization	Byte
21	Number of Lines of Code Directly Related to System Calls	Line (number of)
22	Number of I/O Related Errors	Error (number of)
23	Number of Memory Related Errors	Error (number of)
24	Number of Items Required to be Logged	Item (number of)
25	Number of Modifications Made	Modification (number of)
26	Number of Variables	Variable (number of)
27	Number of Diagnostic Functions Required	Function (number of)
28	Number of Entities	Entity (number of)

Internal Base Measures		
	Measure Name	**Unit of Measurement**
29	Number of Built-in Test Function Required	Function (number of)
30	Number of Test Dependencies on Other System	Dependency (number of)
31	Number of Diagnostic Checkpoints	Checkpoint (number of)
32	Number of Data Structures	Data-Structure (number of)
33	Total Number of Setup Operations	Operation (number of)
34	Number of Installation Steps	Step (number of)

Table IV.3

The Base Measures in ISO 9126-4

Quality in-use Base Measures		
	Measure Name	**Unit of Measurement**
1	Task Effectiveness	(a given weight)
2	Total Number of Tasks	Task (number of)
3	Task Time	Minute
4	Cost of the Task	Dollar
5	Help Time	Second
6	Error Time	Second
7	Search Time	Second
8	Number of Users	User (number of)
9	Total Number of People Potentially Affected by the	Person (number of)
10	Total Number of Usage Situations	Situation (number of)

ANNEX V

CROSS-REFERENCE TABLE OF THE BASE MEASURE USAGES

	Measure Name	External																										Internal																						Quality in use			
		Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Q1	Q2	Q3	Q4
		F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5	F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5				
1	Number of Functions	✓									✓	✓	✓											✓			✓		✓	✓					✓			✓	✓	✓												✓		✓						
2	Operation Time		✓	✓	✓		✓		✓			✓	✓			✓	✓			✓	✓	✓		✓		✓																																				
3	Number of Inaccurate Computations Encountered by Users		✓																																																									
4	Number of Data Formats			✓																										✓																														
5	Number of Illegal Operations				✓																													✓																										
6	Number of Items Requiring Compliance					✓				✓					✓		✓						✓					✓				✓				✓					✓					✓					✓					✓				
7	Number of Interfaces Requiring Compliance					✓																										✓																												
8	Number of Faults						✓																	✓	✓																																						
9	Number of Failures						✓	✓									✓			✓		✓					✓																																						
10	Product Size						✓																																																							
11	Number of Test Cases				✓		✓	✓														✓												✓																												
12	Number of Breakdowns							✓	✓																																																					
13	Time to Repair								✓																																																				
14	Down Time								✓																																																				
15	Number of Restarts								✓																																																				
16	Number of Restoration								✓																												✓																										
17	Number of Tutorials										✓																																																		
18	Number of I/O Data Items										✓																																																		
19	Ease of Function Learning											✓																																																	
20	Number of Tasks											✓				✓																																												✓			
21	Help Frequency											✓																																																	

	Measure Name	External																						Internal																				Quality in use			
		Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Functionality					Reliability				Usability				Efficiency			Maintainability					Portability					Q1	Q2	Q3	Q4
		F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5	F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5				
22	Error Correction												✓																																											
23	Number of Screens/Forms												✓																																											
24	Number of User Errors/Changes												✓																																										✓			
25	Number of Attempts to Customize												✓																																											
26	Number of Corrected Faults																															✓																								
27	Response Time															✓																															✓												
28	Number of Evaluations															✓	✓																																									
29	Turnaround Time															✓																															✓												
30	Task Time															✓																																									
31	Number of I/O Related Errors												✓				✓																																✓												
32	User Waiting Time of I/O Device Utilization																✓																																									
33	Number of Memory Related Errors																✓																																									
34	Number of Transmission Related Error																✓																																									
35	Transmission Capacity																✓																																									
36	Number of Revised Versions																			✓																																						
37	Number of Resolved Failures						✓													✓	✓	✓																																				
38	Porting User Friendliness																							✓																																		
39	Number of Data Items																		✓					✓		✓			✓		✓																								✓			
40	Number of Interface Protocols																															✓																								
41	Number of Access Types																															✓																								
42	Number of Controllability Requirements																															✓																								

	Measure Name	External																									Internal																					Quality in use			
		Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Functionality				Reliability				Usability					Efficiency			Maintainability					Portability					Q1	Q2	Q3	Q4
		F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5	F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5				
22	Error Correction												✓																																											
23	Number of Screens/Forms												✓																																											
24	Number of User Errors/Changes												✓																																										✓			
25	Number of Attempts to Customize												✓																																											
26	Number of Corrected Faults																															✓																								
27	Response Time															✓																													✓														
28	Number of Evaluations															✓	✓																																								
29	Turnaround Time																✓																													✓														
30	Task Time															✓																																								✓			
31	Number of I/O Related Errors												✓				✓																														✓														
32	User Waiting Time of I/O Device Utilization																✓																																								
33	Number of Memory Related Errors																✓																														✓														
34	Number of Transmission Related Error																✓																																								
35	Transmission Capacity																✓																																								
36	Number of Revised Versions																			✓																																						
37	Number of Resolved Failures						✓													✓	✓	✓																																				
38	Porting User Friendliness																							✓																																	
39	Number of Data Items																		✓					✓			✓			✓		✓																							✓			
40	Number of Interface Protocols																														✓																								
41	Number of Access Types																															✓																								
42	Number of Controllability Requirements																															✓																								

	Measure Name	External																							Internal																			Quality in use			
		Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Functionality					Reliability				Usability					Efficiency			Maintainability					Portability					Q1	Q2	Q3	Q4
		F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5	F1	F2	F3	F4	F5	R1	R2	R3	R4	U1	U2	U3	U4	U5	E1	E2	E3	M1	M2	M3	M4	M5	P1	P2	P3	P4	P5				
58	Number of data Structures																							✓																																	
59	Number of Setup Operations																						✓																												✓									
60	Number of Installation Steps																								✓																																	
61	Task Effectiveness																																																							✓	✓		
62	Cost of the Task																																																							✓		
63	Help Time																																																							✓		
64	Error Time																		✓	✓																																					✓		
65	Search Time																																																							✓		
66	Number of Users																																																								✓	
67	Number of People Potentially Affected by the System																																																								✓	✓
68	Number of Usage Situations																																																							✓		

Legend of the Quality in Use characteristics

Q1	Effectiveness
Q2	Productivity
Q3	Safety
Q4	Satisfaction

Legend of the External and Internal sub-characteristics

F1	Suitability	E1	Time Behavior
F2	Accuracy	E2	Resource Utilization
F3	Interoperability	E3	Efficiency Compliance
F4	Security	M1	Analyzability
F5	Functionality Compliance	M2	Changeability
R1	Maturity (Hardware/Software/Data)	M3	Stability
R2	Fault Tolerance	M4	Testability
R3	Recoverability (Data/Process/Technology)	M5	Maintainability Compliance
R4	Reliability Compliance	P1	Adaptability
U1	Understandability	P2	Instability
U2	Learnability	P3	Co-existence
U3	Operability	P4	Replaceability
U4	Attractiveness	P5	Portability Compliance
U5	Usability Compliance		

BIBLIOGRAPHY

Abd Ghani, Abdul Azim, and Robin Hunter. 1996. "An Attribute Grammar Approach to specifying Halstead's Metrics". *Malaysian Journal of Computer Science,* vol. 9, n° 1, p. 56-67.

Abran, Alain. 1996. "Teaching Software Engineering Using ISO Standards". *StandardView,* vol. 4, n° 3, p. 139-145.

Abran, Alain. 1998. "Software Metrics Need to Mature into Software Metrology (Recommendations)". In *the NIST Workshop on Advancing Measurements and Testing for Information Technology.* (Gaithersburg (MD), USA, 26-27 Oct. 1998). <http://www.lrgl.uqam.ca/publications/pdf/374.pdf>. Accessed on 28 March 2007.

Abran, Alain, Miguel Lopez and Naji Habra. 2004. "An Analysis of the McCabe Cyclomatic Complexity Number". In *Proceedings of the 14th International Workshop on Software Measurement - IWSM'2004.* (Berlin (Germany), 2-5 Nov. 2004), p. 391-405. Aachen (Germany): Shaker Verlag.

Abran, Alain, and Asma Sellami. 2002a. "Initial Modeling of the Measurement Concepts in the ISO Vocabulary of Terms in Metrology". In *Proceedings of the 12th International Workshop on Software Measurement - IWSM'2002.* (Magdeburg (Germany), 7-9 Oct. 2002). Aachen (Germany): Shaker Verlag. <http://www.lrgl.uqam.ca/publications/pdf/756.pdf>. Accessed on 28 March 2007.

Abran, Alain, and Asma Sellami. 2002b. "Measurement and Metrology Requirements for Empirical Studies in Software Engineering". In *Proceedings of the 10th International Workshop on Software Technology and Engineering Practice - STEP'2002.* (Montreal (Que), Canada, 6-8 Oct. 2002), p. 185-192. Los Alamitos (CA), USA: IEEE Computer Society Press.

Abran, Alain, and Asma Sellami. 2004. "Analysis of Software Measures Using Metrology Concepts - ISO 19761 Case Study". In *the 6th International Conference on Enterprise Information Systems - ICEIS'2004: the International Workshop on Software Audits and Metrics - SAM'2004.* (Porto (Portugal), 14-17 Apr. 2004). <http://www.lrgl.uqam.ca/publications/pdf/801.pdf>. Accessed on 28 March 2007.

Abran, Alain, Asma Sellami and Witold Suryn. 2003. "Metrology, Measurement and Metrics in Software Engineering". In *Proceedings of the 9th International*

Software Metrics Symposium - METRICS'2003. (Sydney (Australia), 22-23 Sept. 2003), p. 2-11. Los Alamitos (CA), USA: IEEE Computer Society Press.

Akingbehin, Kiumi. 2005. "A Quantitative Supplement to the Definition of Software Quality". In *Proceedings of the 3rd ACIS International Conference on Software Engineering Research Management and Applications - SERA'2005*. (Mount Pleasant (MI), USA, 11-13 Aug. 2005), p. 348- 352. Los Alamitos (CA), USA: IEEE Computer Society Press.

Al Qutaish, R. E. 1998. "Incorporating Software Measurements into a Compiler". MSc thesis, Serdang, Malaysia, Department of Computer Science, Putra University, 186 p.

Albrecht, Allan J. 1979. "Measuring Application Development Productivity". In *Proceedings of the Joint SHARE, GUIDE and IBM Application Development Symposium* (Monetary (CA), USA, Oct. 1979), p. 83-92. Armonk (NY), USA: IBM Corp.

Andersson, Thorbjörn. 1990. "A Survey on Software Quality Metrics". On Line. <http://citeseer.ist.psu.edu/cache/papers/cs/1381/ftp:zSzzSzftp.abo.fizSzpubzSzcszSzpaperszSztaanderszSza120.pdf/andersson90survey.pdf>. Accessed on 26 March 2007.

April, Alain, Alain Abran and Reiner R. Dumke. 2004. "Assessment of Software Maintenance Capability: A Model and its Architecture". In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering - CSMR'2004*. (Tampere (Finland), 24-26 Mar. 2004), p. 243-248. Los Alamitos (CA), USA: IEEE Computer Society Press.

April, Alain, Jane Huffman Hayes, Alain Abran and Reiner R. Dumke. 2005. "Software Maintenance Maturity Model (SMmm): the Software Maintenance Process Model". *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 17, n° 3, p. 197-223.

Azuma, M. 2001. "SQuaRE: The next Generation of ISO/IEC 9126 and 14598 International Standards Series on Software Product Quality". In *Proceedings of the European Software Control and Metrics Conference - ESCOM'2001*. (London (England), UK, 2-4 Apr. 2001), p. 337-346. Aachen (Germany): Shaker Verlag.

Bailey, C. T., and W. L. Dingee. 1981. "A Software Study Using Halstead Metrics". In *Proceedings of the 1981 ACM Workshop / Symposium on Measurement and Evaluation of Software Quality*. (College Park (MD), USA, 25-27 Mar. 1981), p. 189-197. New York (NY), USA: ACM Press.

Biehl, Richard E. 2004. "Six Sigma for Software". *IEEE Software,* vol. 21, n° 2, p. 68-70.

Binder, Robert V. 1997. "Can a Manufacturing Quality Model Work for Software?" *IEEE Software,* vol. 14, n° 5, p. 101-102.

Binder, Robert V. 2001. "Six Sigma: Hardware Si, Software No!" On Line. <http://www.rbsc.com/pages/sixsig.html>. Accessed on 5 April 2007.

Boehm, B. W., J. R. Brown, H. Kaspar, M. Lipow, G. McLeod and M. Merritt. 1978. *Characteristics of Software Quality.* Amsterdam (The Netherlands): North Holland Publishing, 169 p.

Boehm, Barry W., J. R. Brown and M. Lipow. 1976. "Quantitative evaluation of software quality". In *Proceedings of the 2nd international conference on Software engineering.* (San Francisco (CA), USA, 13-15 Oct. 1976), p. 592-605. Los Alamitos (CA), USA: IEEE Computer Society.

Booth, Simon P., and Simon B. Jones. 1996. "Are Ours Really Smaller Than Theirs?" In *the Glasgow Workshop on Functional Programming.* (Ullapool (Scotland), UK, Jul. 1996). <http://www.dcs.gla.ac.uk/fp/workshops/fpw96/Booth.pdf>. Accessed on 28 March 2007.

Bourque, Pierre, Sibylle Wolff, Robert Dupuis, Asma Sellami and Alain Abran. 2004. "Lack of Consensus on Measurement in Software Engineering: Investigation of Related Issues". In *Proceedings of the 14th International Workshop on Software Measurement - IWSM'2004.* (Berlin (Germany), 2-4 Nov. 2004), p. 321-333. Aachen (Germany): Shaker Verlag.

Breyfogle, Forrest W. 2003. *Implementing Six Sigma: Smarter Solution Using Statistical Methods.* Hoboken (NJ), USA: John Wiley & Sons Inc.

Brue, Greg. 2005. *Six Sigma for Managers.* New York (NY), USA: McGraw-Hill, 64 p.

Buglione, Luigi, and Alain Abran. 1999. "Geometrical and statistical foundations of a three-dimensional model of software performance". *Advances in Engineering Software,* vol. 30, n° 12, p. 913-919.

Buglione, Luigi, and Alain Abran. 2002. "QEST nD: n-dimensional extension and generalisation of a software performance measurement model". *Advances in Engineering Software,* vol. 33, n° 1, p. 1-7.

Burnstein, Ilene, Taratip Suwanassart and C. R. Carlson. 1996a. "Developing a Testing Maturity Model: Part I". *CrossTalk: the Journal of Defense Software Engineering,* vol. 9, n° 8, p. 21-24.

Burnstein, Ilene, Taratip Suwanassart and C. R. Carlson. 1996b. "Developing a Testing Maturity Model: Part II". *CrossTalk: the Journal of Defense Software Engineering,* vol. 9, n° 9, p. 19-26.

Carnahan, Lisa, Gary Carver, Martha Gray, Michael Hogan, Theodore Hopp, Jeffrey Horlick, Gordon Lyon and Elena Messina. 1997. "Metrology for Information Technology". *StandardView,* vol. 5, n° 3, p. 103-109.

Christensen, K., G. P. Fitsos and C. P. Smith. 1981. "A perspective on software science". *IBM Systems Journal,* vol. 20, n° 4, p. 372-387.

Chuan, Chan Hock, Lim Lin, Lim Liuh Ping and Loh Vee Lian. 1994. "Evaluation of Query Languages with Software Science Metrics". In *Proceedings of the IEEE Region 10's 9th Annual International Conference on Frontiers of Computer Technology - TENCON'1994.* (Singapore (Singapore), 22-26 Aug. 1994), p. 516-520. Los Alamitos (CA), USA: IEEE Computer Society Press.

Coallier, François. 2006. "Les Pratiques en Informatique d'entreprise un Modèle Intégrateur". PhD thesis, Montréal, Québec, Canada, Département de Génie Informatique, École Polytechnique de Montréal, 316p.

Conte, Samuel D., H. E. Dunsmore and V. Y. Shen. 1986. *Software Engineering Metrics and Models.* Menlo Park (CA), USA: Benjamin Cummings, 396 p.

Curtis, Bill. 1980. "Measurement and Experimentation in Software Engineering". *Proceedings of the IEEE,* vol. 68, n° 9, p. 1144-1157.

DeMarco, T. 1986. *Controlling Software Projects: Management, measures and estimation.* New York (NY), USA: Prentice Hall PTR, 296 p.

Dromey, R. G. 1995. "A model for software product quality". *IEEE Transactions on Software Engineering,* vol. 21, n° 2, p. 146-162.

Dromey, R. G. 1996. "Concerning the Chimera [software quality]". *IEEE Software,* vol. 13, n° 1, p. 33-43.

Dumke, Reiner R., and Erik Foltin. 1998. "Metrics-Based Evaluation of Object-Oriented Software Development Methods". In *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering - CSMR'1998.*

(Florence (Italy), 8-11 Mar. 1998), p. 163-196. Los Alamitos (CA), USA: IEEE Computer Society.

Emerson, W. H. 2005. "Short Communication on the Concept of Dimension". *Metrologia,* vol. 42, n° 2, p. 21–22.

Fehlmann, Thomas M. 2004. "Six Sigma for Software". In *Proceedings of the 1st Software Measurement European Forum - SMEF'2004.* (Rome (Italy), 28-30 Jan. 2004). <http://www.swisma.ch/Meeting18/SixSigmaforSoftware.pdf>. Accessed on 5 April 2007.

Fenton, N. 1994. "Software Measurement: A Necessary Scientific Basis". *IEEE Transaction on Software Engineering,* vol. 20, n° 3, p. 199-206.

Fenton, Norman E., and Shari Lawrence Pfleeger. 1997. *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston (MA), USA: PWS Publishing Company, 656 p.

Golden, Bernard. 2004. *Succeeding with Open Source*. Boston (MA), USA: Addison-Wesley Professional, 272 p.

Grady, Robert B. 1992. *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs (NJ), USA: Prentice Hall, 282 p.

Grady, Robert B., and Deborah L. Caswell. 1987. *Software Metrics: Establishing a Company-wide Program*. Upper Saddle River (NJ), USA: Prentice Hall PTR, 275 p.

Gray, Martha M. 1999. "Applicability of Metrology to Information Technology". *Journal of Research of the National Institute of Standards and Technology,* vol. 104, n° 6, p. 567-578.

Habra, Naji, Alain Abran, Miguel Lopez and Valérie Paulus. 2004. *Toward a Framework for Measurement Lifecycle*. Technical Report TR37/04, Namur (Belgium): University of Namur, 15 p.

Halstead, Maurice. H. 1972. "Natural Laws Controlling Algorithm Structure". *ACM SIGPLAN Notices,* vol. 7, n° 2, p. 19-26.

Halstead, Maurice. H. 1977. *Elements of Software Science*. New York, NY, USA: Elsevier North-Holland, 127 p.

Hamer, Peter G., and Gillian D. Frewin. 1982. "M. H. Halstead's Software Science - A Critical Examination". In *Proceedings of the 6th International Conference on*

Software Engineering. (Tokyo (Japan), 13-16 Sept. 1982), p. 197-206. Los Alamitos (CA), USA: IEEE Computer Society Press.

Hayes, Bruce. 2003. "Introduction to Six Sigma for Software: the Third Wave". In *the Six Sigma for Software Development Conference*. (Boston (MA), USA, 16-17 Oct. 2003). <http://www.sei.cmu.edu/sema/pdf/sdc/hayes.pdf>. Accessed on 5 April 2007.

Head, G. E. 1994. "Six-Sigma Software Using Cleanroom Software Engineering Techniques". *Hewlett-Packard Journal,* vol. 45, n° 3, p. 40-50.

Hefner, Rick, and Michael Sturgeon. 2002. "Optimize Your Solution: Integrating Six Sigma and CMM/CMMI-Based Process Improvement". In *the 14th Annual Software Technology Conference*. (Salt Lake City (UT) USA, 29 Apr.- 2 May 2002). <http://www.sstc-online.org/proceedings/2002/SpkrPDFS/TuesTrac/p566.pdf>. Accessed on 5 April 2007.

Heinz, Lauren. 2004. "Using Six Sigma in Software Development". *News@SEI*, n° 1 (November), p. 1-3. On Line. <http://www.sei.cmu.edu/news-at-sei/features/2004/1/feature-3.htm>. Accessed on 5 April 2007.

Henry, S., and D. Kafura. 1981. "Software Structure Metrics Based on Information Follow". *IEEE Transaction on Software Engineering,* vol. 7, n° 5, p. 510-518.

Hong, G. Y., and T. N. Goh. 2003. "Six Sigma in Software Quality". *the TQM Magazine,* vol. 15, n° 6, p. 364-373.

IEEE. 1990. *Standard Glossary of Software Engineering Terminology*. IEEE Std. 610.12-1990, New York (NY), USA: the Institute of Electrical and Electronics Engineers, 83 p.

IEEE. 1998. *Standard for Software Verification and Validation*. IEEE Std. 1012-1998, New York (NY), USA: The Institute of Electrical and Electronics Engineers, 71 p.

Ince, D. C., and M. J. Shepperd. 1989. "An Empirical and Theoretical Analysis of An Information-Flow-Based System Design Metrics". In *Proceedings of the 2nd European Software Engineering Conference*. (Warwick (Coventry), UK, 11-15 Sept. 1989). <http://portal.acm.org/citation.cfm?id=168026.168067&coll=GUIDE&dl=GUIDE&CFID=13880735&CFTOKEN=16648325>. Accessed on 11 December 2006.

Ince, Darrel S., Helen Sharp and Mark Woodman. 1993. *Introduction to Software Project Management and Quality Assurance*. New York (NY), USA: McGraw-Hill, 224 p.

ISO. 1984. *International Vocabulary of Basic and General Terms in Metrology (VIM)*. Geneva (Switzerland): International Organization for Standardization.

ISO. 1991. *Software Product Evaluation - Quality Characteristics and Guidelines for their Use*. ISO/IEC IS 9126, Geneva (Switzerland): International Organization for Standardization.

ISO. 1993. *International Vocabulary of Basic and General Terms in Metrology (VIM)*. Geneva (Switzerland): International Organization for Standardization.

ISO. 1994. *Information technology - Software packages - Quality Requirements and Testing*. ISO/IEC 12119, Geneva (Switzerland): International Organization for Standardization.

ISO. 1998a. *Information technology - Software product evaluation - Part 5: Process for evaluators*. ISO/IEC 14598-5, Geneva (Switzerland): International Organization for Standardization, 35 p.

ISO. 1998b. *Information Technology - System and Software Integrity Levels*. ISO/IEC 15026, Geneva (Switzerland): International Organization for Standardization, 12 p.

ISO. 1998c. *Software Measurement - Functional Size Measurement - Part 1: Definition of Concepts*. ISO/IEC 14143-1, Geneva (Switzerland): International Organization for Standardization.

ISO. 1999a. *Information technology - Software product evaluation - Part 1: General overview*. ISO/IEC 14598-1, Geneva (Switzerland): International Organization for Standardization, 19 p.

ISO. 1999b. *Software engineering - Product evaluation - Part 4: Process for acquirers*. ISO/IEC 14598-4, Geneva (Switzerland): International Organization for Standardization, 34 p.

ISO. 2000a. *Software engineering - Product evaluation - Part 2: Planning and management*. ISO/IEC 14598-2, Geneva (Switzerland): International Organization for Standardization, 12 p.

ISO. 2000b. *Software engineering - Product evaluation - Part 3: Process for developers.* ISO/IEC 14598-3, Geneva (Switzerland): International Organization for Standardization, 16 p.

ISO. 2001a. *Software engineering - Product evaluation - Part 6: Documentation of evaluation modules.* ISO/IEC 14598-6, Geneva (Switzerland): International Organization for Standardization, 31 p.

ISO. 2001b. *Software Engineering - Product Quality - Part 1: Quality Model.* ISO/IEC 9126-1, Geneva (Switzerland): International Organization for Standardization, 25 p.

ISO. 2002. *Software Engineering - Software Measurement Process.* ISO/IEC 15939, Geneva (Switzerland): International Organization for Standardization, 37 p.

ISO. 2003a. *Information Technology - Process assessment - Part 2: Performing an Assessment.* ISO/IEC 15504-2, Geneva (Switzerland): International Organization for Standardization, 16 p.

ISO. 2003b. *ISO/IEC 19761: Software Engineering - COSMIC-FFP - A Functional Size Measurement Method.* Geneva (Switzerland): International Organization for Standardization, 17 p.

ISO. 2003c. *Software Engineering - Product Quality - Part 2: External Metrics.* ISO/IEC TR 9126-2, Geneva (Switzerland): International Organization for Standardization, 86 p.

ISO. 2003d. *Software Engineering - Product Quality - Part 3: Internal Metrics.* ISO/IEC TR 9126-3, Geneva (Switzerland): International Organization for Standardization, 62 p.

ISO. 2004a. *Information Technology - Process Assessment - Part 1: Concepts and Vocabulary.* ISO/IEC 15504-1, Geneva (Switzerland): International Organization for Standardization, 19 p.

ISO. 2004b. *Information Technology - Process Assessment - Part 3: Guidance on Performing an Assessment.* ISO/IEC 15504-3, Geneva (Switzerland): International Organization for Standardization, 54 p.

ISO. 2004c. *Information Technology - Process Assessment - Part 4: Guidance on Use for Process Improvement and Process Capability Determination.* ISO/IEC 15504-4, Geneva (Switzerland): International Organization for Standardization, 33 p.

ISO. 2004d. *Software and System Engineering - Guidelines for the Design and Preparation of User Documentation for Application Software.* ISO/IEC 18019, Geneva (Switzerland): International Organization for Standardization, 146 p.

ISO. 2004e. *Software Engineering - Guidelines for the Application of ISO 9001:2000 to Computer Software.* ISO/IEC 90003, Geneva (Switzerland): International Organization for Standardization, 54 p.

ISO. 2004f. *Software Engineering - Product Quality - Part 4: Quality in Use Metrics.* ISO/IEC TR 9126-4, Geneva (Switzerland): International Organization for Standardization, 59 p.

ISO. 2005. *Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE.* ISO/IEC 25000, Geneva (Switzerland): International Organization for Standardization, 41 p.

ISO. 2006a. *Information Technology - Process Assessment - Part 5: An Exemplar Process Assessment Model, Document Number: N3302 Dated on 14 September 2005.* ISO/IEC 15504-5, Geneva (Switzerland): International Organization for Standardization, 162 p.

ISO. 2006b. *Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Requirements for Quality of Commercial Off-The-Shelf (COTS) Software Product and Instructions for Testing.* ISO/IEC 25051, Geneva (Switzerland): International Organization for Standardization, 27 p.

ISO. 2007a. *Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Measurement Reference Model and Guide.* ISO/IEC FDIS 25020, Geneva (Switzerland): International Organization for Standardization, 15 p.

ISO. 2007b. *Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Quality Measure Elements.* ISO/IEC DTR 25021, Geneva (Switzerland): International Organization for Standardization, 64 p.

Jacobson, I., G. Booch and J. Rumbaugh. 1999. *The Unified Software Development Process.* Boston (MA), USA: Addison-Wesley Professional. 463 p.

Jacowski, Tony. 2006. "Six Sigma in the Software Industry". On Line. <http://ezinearticles.com/?Six-Sigma-In-The-Software-Industry&id=198268>. Accessed on 5 April 2007.

Jacquet, Jean-Philppe, and Alain Abran. 1997. "From Software Metrics to Software Measurement Methods: A Process Model". In *Proceedings of the 3rd*

International Symposium and Forum on Software Engineering Standards - ISESS'1997. (Walnut Creek (CA), USA, 1-6 Jun. 1997), p. 128-135. Los Alamitos (CA), USA: IEEE Computer Society Press.

Joyce, Edward. 1989. "Is Error-Free Software Possible?" *Datamation,* (February 18).

Kafura, Dennis, and James Canning. 1985. "A Validation of Software Metrics Using Many Metrics and two Resources". In *Proceedings of the 8th International Conference on Software Engineering.* (London (England), UK, 28-30 Aug. 1985), p. 378-385. Los Alamitos (CA), USA: IEEE Computer Society Press.

Kiricenko, Victoria, and Olga Ormandjieva. 2005. "Measurement of OOP Size Based on Halstead's Software Science". In *Proceedings of the 2nd Software Measurement European Forum - SMEF'2005.* (Rome (Italy), 2-4 Mar. 2005), p. 253-259.

Kitchenham, B., S. L. Pfleeger and N. Fenton. 1995. "Towards a Framework for Software Measurement Validation". *IEEE Transaction on Software Engineering,* vol. 21, n° 12, p. 929-944.

Kitchenham, Barbara, and Shari Lawrence Pfleeger. 1996. "Software Quality: the Elusive Target". *IEEE Software,* vol. 13, n° 1, p. 12-21.

Koscianski, André, and João Candido Bracarense Costa. 1999. "Combining Analytical Hierarchical Analysis with ISO/IEC 9126 for a Complete Quality Evaluation Framework". In *Proceedings of the 4th IEEE International Symposium and Forum on Software Engineering Standards - ISESS'1999.* (Curitiba (Brazil), 17-22 May 1999), p. 218-226. Los Alamitos (CA), USA: IEEE Computer Society Press.

Kruchten, P. 2000. *The Rational Unified Process: An Introduction,* 2nd Ed. Boston (MA), USA: Addison-Wesley Professional. 320 p.

Kumar, Garikapati Pavan. 2005. "Software Process Improvement - TRIZ and Six Sigma (Using Contradiction Matrix and 40 Principles)". *TRIZ Journal,* (April). On Line. < http://www.triz-journal.com/archives/2005/04/07.pdf>. Accessed on 5 April 2007.

Leach, R. J. 1995. "Using Metrics to Evaluate Student Programs". *ACM SIGCSE Bulletin,* vol. 27, n° 2, p. 41-48.

Li, Da Yu, Victoria Kiricenko and Olga Ormandjieva. 2004. "Halstead's Software Science in Today's Object Oriented World". *Metrics News,* vol. 9, n° 2, p. 33-40.

Lister, A. M. 1982. "Software Science - The Emperor's New Clothes". *the Australian Computer Journal,* vol. 14, n° 2, p. 66-70.

Mazur, Glenn H. 2003. "QFD in support of design for six sigma". In *Proceedings of 8th International Conference on ISO and TQM.* (Montreal (Que.), Canada, Apr. 2003). <http://jobfunctions.bnet.com/whitepaper.aspx?docid=157440>. Accessed on 5 April 2007.

McBride, Tom, Brian Henderson-Sellers and Didar Zowghi. 2004. "Project Management Capability Levels: An Empirical Study". In *Proceedings of the 11th Asia-Pacific Software Engineering Conference - APSEC'2004.* (Busan (Korea), 30 Nov. - 3 Dec. 2004), p. 56-63. Los Alamitos (CA), USA: IEEE Computer Society Press.

McCabe, T. J., and C. W. Butler. 1989. "Design Complexity Measurement and Testing". *Communication of the ACM,* vol. 32, n° 12, p. 1415-1425.

McCabe, Thomas J. 1976. "A Complexity Measure". *IEEE Transaction on Software Engineering,* vol. 2, n° 4, p. 308-320.

McCall, J. A., P. K. Richards and G. F. Walters. 1977. *Factors in Software Quality, Volumes I, II, and III.* USA: US Rome Air Development Center Reports, US Department of Commerce.

Menzies, Tim, Justin S. Di Stefano, Mike Chapman and Ken McGill. 2002. "Metrics That Matter". In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop.* (Greenbelt (MD), USA, 5-6 Dec. 2002), p. 51-57. Los Alamitos (CA), USA: IEEE Computer Society Press.

Microsoft. 2006. "Six Sigma: High Quality Can Lower Costs and Raise Customer Satisfaction". On Line. <http://download.microsoft.com/download/a/1/b/a1b38fdf-efff-4fc1-8813-1ffde5ce0d11/sixsigma.pdf>. Accessed on 5 April 2007.

Morasca, Sandro and Lionel C. Briand. 1997. "Towards a Theoretical Framework for Measuring Software Attributes". In *Proceedings of 4th International Software Metrics Symposium.* (Albuquerque (NM), USA, 5-7 Nov. 1997), p. 119-126. Los Alamitos (CA), USA: IEEE Computer Society Press.

Motorola. 2004. "Six Reasons Why Leaders Love Six-Sigma". On Line. <https://mu.motorola.com/six_sigma_lessons/contemplate/assembler.asp?page=leaders_title>. Accessed on 5 April 2007.

Motorola. 2007. "Six Sigma Dictionary". On line. <http://www.motorola.com/content.jsp?globalObjectId=3074-5804>. Accessed on 5 April 2007.

Nastro, John. 1997. "A Software Product Maturity Model". *CrossTalk: the Journal of Defense Software Engineering,* vol. 10, n° 8.

Petrasch, Roland. 1999. "The Definition of 'Software Quality': A Practical Approach". In *Proceedings of the 10th International Symposium on Software Reliability Engineering - ISSRE'1999.* (Boca Ralon (FL), USA). <http://www.chillarege.com/fastabstracts/issre99/99124.pdf>. Accessed on 28 March 2007.

Pickerill, Jay. 2005. "Implementing the CMMI in a Six Sigma World". In *the 17th Software Engineering Process Group Conference - SEPG'2005.* (Seattle, (WA), USA, 7-10 Mar. 2005). <http://www.sei.cmu.edu/cmmi/adoption/pdf/pickerill.pdf>. Accessed on 5 April 2007.

Pressman, Roger S. 2004. *Software Engineering: A Practitioner's Approach*, 6th ed. New York (NY), USA: McGraw-Hill, 880 p.

Roberts, F. S. 1979. *Measurement Theory, with Applications to Decision Making, Utility, and the Social Sciences.* Boston (MA), USA: Addison Wesley.

Salt, N. F. 1982. "Defining Software Science Counting Strategies". *ACM SIGPLAN Notices,* vol. 17, n° 3, p. 58-67.

Samoladas, I., I. Stamelos, L. Angelis and A. Oikonomou. 2004. "Open Source Software Development Should Strive for Even Greater Code Maintainability". *Communication of ACM,* vol. 47, n° 10, p. 83-87.

Sauro, Jeff, and Erika Kindlund. 2005a. "Making Sense of Usability Metrics: Usability and Six Sigma". In *Proceedings of the 14th Annual Conference of the Usability Professionals Association - UPA'2005.* (Montreal (Que.), Canada). <http://www.measuringusability.com/sauro-kindlund_paper.pdf>. Accessed on 23 March 2007.

Sauro, Jeff, and Erika Kindlund. 2005b. "A Method to Standardize Usability Metrics into a Single Score". In *Proceedings of the Conference in Human Factors in Computing Systems - CHI'2005.* (Portland (OR), USA, 2-7 Apr. 2005), p. 401-409. New York (NY), USA: ACM Press.

Schmidt, Michael E. C. 2000. *Implementing the IEEE Software Engineering Standards.* Indianapolis, IN, USA: Sams Publishing, 256 p.

Schofield, Joe. 2006. "When Did Six Sigma Stop Being a Statistical Measure?" *CrossTalk: the Journal of defense Software Engineering,* vol. 19, n° 6, p. 28-29.

SEI. 1993. *Capability Maturity Model for Software Engineering (Version 1.1).* CMU/SEI -93-TR024, Pittsburgh (PA), USA: Software Engineering Institute, Carnegie Mellon University.

SEI. 2002a. *Capability Maturity Model Integration for Software Engineering (CMMI-SW) - Continuous Representation, Version 1.1.* CMU/SEI-2002-TR-028. Pittsburgh (PA), USA: Software Engineering Institute, Carnegie Mellon University, 630 p. <http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr028.pdf>. Accessed on 29 March 2007.

SEI. 2002b. *Capability Maturity Model Integration for Software Engineering (CMMI-SW) - Staged Representation, Version 1.1.* CMU/SEI-2002-TR-029. Pittsburgh (PA), USA: Software Engineering Institute, Carnegie Mellon University, 624 p. <http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr029.pdf>. Accessed on 29 March 2007.

Sellami, Asma, and Alain Abran. 2003. "the Contribution of Metrology Concepts to Understanding and Clarifying a Proposed Framework for Software Measurement Validation". In *Proceedings of the 13th International Workshop on Software Measurement - IWSM'2003.* (Montreal (Que.), Canada, 23-25 Sept. 2003), p. 18-40. Aachen (Germany): Shaker Verlag.

Sharp, Alex. 1993. *Software Quality and Productivity.* New York (NY), USA: Nost Rand Reinhold, 382 p.

Shelley, C. C. 2003. "Six Sigma and its Application to Software Development". In *Oxford Software Engineering.* On Line. <http://www.osel.co.uk/sixsigmapaper.pdf>. Accessed on 5 April 2007.

Shen, V. Y., S. D. Conte and H. E. Dunsmore. 1983. "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support". *IEEE Transaction on Software Engineering,* vol. 9, n° 2, p. 155-165.

Shepperd, M. J. 1990. "Design Metrics: An Empirical Analysis". *Software Engineering Journal,* vol. 5, n° 1, p. 3-10.

Simpson, John A. 1981. "Foundations of Metrology". *Journal of Research of the National Bureau of Standards,* vol. 86, n° 3, p. 36-42.

Siviy, Jeannine M., and Eileen C. Forrester. 2004. "Using Six Sigma to Accelerate the Adoption of CMMI for Optimal Results". In *the Six Sigma for Software Development Conference.* (Boston (MA), USA, 16-17 Oct. 2003). <http://www.sei.cmu.edu/sema/presentations/optimal.pdf>. Accessed on 5 April 2007.

Suryn, Witold, Alain Abran and Alain April. 2003. "ISO/IEC SQuaRE: The Second Generation of Standards for Software Product Quality". In *Proceedings of the 7th*

IASTED International Conference on Software Engineering and Applications. (Marina del Rey (CA), USA, 3-5 Nov. 2003). Calgary (Alb.), Canada: ACTA Press.

Szentes, S. 1986. *QUALIGRAPH User Guide*. Budapest (Hungary): Research and Innovation Center.

Thomsett, Michael C. 2005. *Getting Started in Six Sigma*. Hoboken (NJ), USA: John Wiley & Sons, 213 p.

VanHilst, Michael, Pankaj K. Garg and Christopher Lo. 2005. "Repository Mining and Six Sigma for Process Improvement". *ACM SIGSOFT Software Engineering Notes,* vol. 30, n° 4 (July), p. 1-4.

Veenendaal, Erik Van, and Julie McMullan. 1997. *Achieving Software Product Quality*. Den Bosch (The Netherlands): UTN Publishers, 208 p.

Wheeler, Donald J. 2004. *The Six Sigma Practitioner's Guide to Data Analysis*. Knoxville (TN), USA: SPC Press, 410 p.

Yin, B. H., and J. W. Winchester. 1978. "The Establishment and Use of Measures to Evaluate the Quality of Software Design". In *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues* (San-Diego (CA), USA), p. 45-52. New York (NY), USA: ACM Press.

Zuse, Horst. 1998. *A Framework of Software Measurement*. Berlin (Germany): Walter de Gruyter, 755 p.