ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MASTER THESIS PRESENTED AT
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

AS A PARTIAL REQUIREMENT
FOR THE OBTENTION OF
MASTER OF ENGINEERING
M. ING.

BY
MARC-ALEXIS CÔTÉ

AN ANALYSIS OF QUALITY MODELS AS A FOUNDATION FOR SOFTWARE QUALITY ENGINEERING

MONTRÉAL, MARS 1, 2005

THIS THESIS HAS BEEN EVALUATED
BY A JURY COMPOSED OF :

Mister Witold Suryn, professor and research director
Faculty of Software Engineering and IT at École de technologie supérieure

Mister Pierre Bourque, professor and president of the jury
Faculty of Software Engineering and IT at École de technologie supérieure

Mister Claude Y. Laporte, professor
Faculty of Software Engineering and IT at École de technologie supérieure

THIS MEMOIR HAS BEEN DEFENDED BEFORE A JURY
ON FEBRUARY 1, 2005
AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ANALYSE DE MODÈLES DE QUALITÉ EN TANT QUE FONDATION À L'INGÉNIERIE DE LA QUALITÉ DU LOGICIEL

Marc-Alexis Côté

SOMMAIRE

L'ingénierie de la qualité du logiciel (Software Quality Engineering) est une discipline émergente dont le principal champ d'activité est l'amélioration de la qualité dans les systèmes à base de logiciels. Il est important que cette nouvelle discipline soit assise sur une base solide prenant la forme d'un modèle de qualité bien adapté à ses besoins. Afin de cerner correctement ces besoins, la signification de ce que représente la qualité est explorée lors d'une revue de la littérature scientifique sur ce sujet. Il est convenu que l'ingénierie de la qualité du logiciel nécessite un modèle de qualité possédant les caractéristiques suivantes: facilite autant la définition des exigences de qualité que l'évaluation de la qualité, exhaustif et extensible, utile tout au long du cycle de vie. Le but de cette recherche est d'identifier un modèle de qualité applicable à l'ingénierie de la qualité du logiciel. Afin d'atteindre ce but, les modèles de qualité reconnus par le milieu universitaire et l'industrie sont comparés en évaluant leur appui pour l'ingénierie de la qualité du logiciel. ISO/IEC 9126 est sélectionné comme étant le modèle le plus prometteur à ce sujet. À la suite d'une analyse approfondie du cadre du modèle et des mesures y étant attachées, il est conclu que même si le cadre appuyant le modèle ISO/IEC est conforme aux besoins, une quantité considérable de mesures ne satisfont pas les exigences. La raison de cet échec est principalement attribuable au fait qu'une majorité des mesures ne sont pas clairement utilisables pour définir des exigences de qualité. Des recommandations sont étayées afin de rectifier la situation.

AN ANALYSIS OF QUALITY MODELS AS A FOUNDATION FOR SOFTWARE QUALITY ENGINEERING

Marc-Alexis Côté

ABSTRACT

Software Quality Engineering is an emerging discipline that is concerned with improving the approach to software quality. It is important that this discipline be firmly rooted in a quality model satisfying its needs. In order to define the needs of this discipline, the meaning of quality is broadly defined by reviewing the literature on the subject. Software Quality Engineering needs a quality model that supports the specification of quality as well as its evaluation; it needs a quality model that is exhaustive and extensible; it needs a quality model that is widely applicable and usable throughout the software lifecycle. The goal of this research is to identify a quality model suitable for such a purpose. In order to attain this goal, quality models are comparatively evaluated with respect to their support of Software Quality Engineering. ISO/IEC 9126 is selected as the most promising model and further evaluated. Through a more in depth analysis of the standard and its associated measures, it is found that although the framework behind ISO/IEC 9126 is clearly supportive of the needs of Software Quality Engineering, the quality measures associated with the model largely fail to meet expectations. Measures were found to be unsatisfying because a majority of them fail to be useful in setting quality goals and requirements. Recommendations on how to improve the measures are presented in detail.

ANALYSE DE MODÈLES DE QUALITÉ EN TANT QUE FONDATION À L'INGÉNIERIE DE LA QUALITÉ DU LOGICIEL

Marc-Alexis Côté

RÉSUMÉ FRANÇAIS

Au cours de la dernière décennie, l'attention de l'industrie du logiciel s'est déplacée de l'ajout continuel de nouvelles fonctionnalités vers l'amélioration de la qualité. Avec notre dépendance toujours croissante en tant que société à l'égard des logiciels, cette tendance vers la qualité ne fera que s'accentuer avec le temps.

Afin de bien cerner les besoins du client, il est généralement convenu qu'une bonne pratique est d'établir de façon plus ou moins formelle ses exigences spécifiques. Traditionnellement, les exigences de ce dernier ont été classées en deux catégories distinctes, soit d'une part les exigences fonctionnelles, et d'autre part celles dites non fonctionnelles. Avec cette poussée vers des logiciels de qualité, une nouvelle catégorie d'exigences est en train de voir le jour, soit celles de qualité. Pour bien cerner les exigences de qualité du client, il est nécessaire que la qualité en tant que telle soit bien définie. Cette définition ne peut malheureusement pas prendre la forme d'un simple énoncée du genre : « La qualité du logiciel est mesurée par le niveau d'observance des exigences du client ». La qualité se doit d'être définie par un modèle complexe en décrivant tous les tenants et aboutissants. Malheureusement, cette lancée vers la qualité du logiciel n'est pas appuyée par un modèle de qualité permettant tout autant la définition des exigences de qualité que l'évaluation subséquente de leur respect dans le produit final.

Le but de ce travail est d'identifier un modèle de qualité qui pourrait servir de base à l'amélioration de la qualité des logiciels d'une façon continue, systématique, disciplinée et quantifiable (Suryn, 2003). Afin d'atteindre ce but, les étapes suivantes seront suivies :

- Une revue de la littérature pertinente au sujet de cette recherche permettra d'établir une définition largement acceptée de ce qu'est la qualité.

- En utilisant les prémisses établies lors de la revue de la littérature, quatre modèles de qualité reconnus par le milieu universitaire et l'industrie seront analysés.

- Des quatre modèles précédents, une analyse en profondeur suivie d'une évaluation seront conduites sur le modèle le plus prometteur.

- Les résultats de cette dernière étape seront analysés sous la forme d'une discussion et des recommandations seront émises.

Ce résumé survolera ces quatre étapes.

Revue de la littérature

Il fut mentionné précédemment qu'une tendance émergente dans le domaine de l'ingénierie du logiciel est d'établir des exigences de qualité. Suryn (2003), dans un recensement des principaux écrits utilisés pour enseigner l'ingénierie du logiciel, constate qu'aucun ne reconnaît l'existence de telles exigences. Il constate de plus qu'aucun de ces ouvrages de référence ne reconnaît l'implémentation de la qualité comme un effort fortement lié au cycle de vie d'un logiciel. Cela le mène à définir l'ingénierie de la qualité du logiciel, une discipline à laquelle la littérature fait souvent référence, mais dont la portée est souvent floue, comme suit:

> *"The application of a continuous, systematic, disciplined, quantifiable approach to the development and maintenance of quality of software products and systems; that is, the application of quality engineering to software."*

Cette méthode se doit d'être solidement appuyée par un modèle de qualité. Malheureusement, un recensement des écrits traitant de la qualité des logiciels nous laisse croire qu'il existe un schisme profond dans l'industrie au sujet de la définition de ce concept. Traditionnellement, l'évaluation la qualité s'est limitée à la mesure du niveau d'observance des exigences du client. Cette approche découle du milieu manufacturier où des milliers, voir des millions de pièces sont produites et doivent être identiques afin d'assurer leur interopérabilité. Il est de l'avis de plusieurs qu'une définition plus large est nécessaire pour l'ingénierie du logiciel, car diverses spécificités de la programmation sont différentes de celles d'une chaîne de montage. En effet, le problème n'est pas de produire des millions d'exemplaires identiques, mais bien de créer *un* logiciel *d'une* qualité certaine. De l'avis de Kitchenham et Pfleeger (1996), la qualité se manifeste sous 5 perspectives :

- L'aspect transcendantal reconnaît que la qualité a un élément métaphysique non quantifiable. Il s'agit d'un idéal vers lequel tous veulent se diriger, mais que nul ne peut atteindre complètement.

- L'aspect de l'utilisation reconnaît que la qualité a trait à l'applicabilité du produit par rapport à un contexte d'utilisation.

- L'aspect manufacturier reconnaît que la qualité est aussi appréciée en mesurant le niveau d'observance des exigences du client.

- L'aspect appelé « produit » reconnaît que la qualité peut être appréciée en mesurant les qualités inhérentes d'un produit.

- L'aspect de la valeur reconnaît que les perspectives énumérées ci-dessus peuvent avoir une valeur différente pour divers usagers.

Traditionnellement, les différents efforts pour atteindre la qualité ont eu tendance à se concentrer sur un ou l'autre de ces aspects. Par exemple, les modèles comme le Capability Maturity Model (CMM) tendent à mettre une emphase sur l'aspect manufacturier. D'autres croient qu'en infusant des caractéristiques de qualité à un produit, le résultat manifestera des valeurs probantes de qualité. Il s'agit là d'une emphase sur l'aspect produit. La première prémisse sur laquelle s'appuie ce travail stipule qu'un modèle de qualité visant à appuyer l'ingénierie de la qualité du logiciel doit reconnaître l'importance de chacun de ces 5 aspects. Une seconde prémisse précise qu'il ne suffit pas que le modèle permette l'évaluation de la qualité; il doit également appuyer la découverte et la définition d'exigences ayant trait à cette dernière.

Sélection d'un modèle de qualité

L'aspect le plus important d'un modèle devant appuyer l'ingénierie de la qualité du logiciel est qu'il permette autant l'évaluation de la qualité que la définition d'exigences ayant trait à celle-ci. L'IEEE (IEEE, 1998) définit ces deux éléments comme étant respectivement l'approche du bas vers le haut (bottom to top) et l'approche du haut vers le bas (top to bottom). L'évaluation de la qualité, soit l'approche du bas vers le haut, est inhérente à tout modèle de qualité. En nous inspirant du standard IEEE 1061, nous avons formulé trois questions nous permettant d'évaluer le support d'un modèle de qualité pour la définition des exigences de qualité :

- Est-ce que le modèle peut être utilisé tôt dans le cycle de vie pour définir des facteurs de qualité importants?

- Est-ce que les exigences de qualité établies à l'aide du modèle peuvent être communiquées de façon efficace vers le personnel technique chargé de les implémenter?

- Est-il possible d'identifier des mesures qui permettront de vérifier l'implémentation de ces exigences?

Une évaluation des modèles de qualité de McCall (1977), Boehm (1978), Dromey (1995) et ISO/IEC 9126 (2001a), à l'aide de ces questions, nous permet de conclure que seul ISO/IEC 9126 *peut* appuyer l'ingénierie de qualité. De plus, ce modèle, séparé en différentes parties, est le seul qui reconnaît les différentes perspectives de la qualité mentionnées ci-dessus. Une étude plus approfondie est nécessaire afin de vérifier si ce support se manifeste dans tous les aspects du modèle.

Analyse et évaluation de ISO/IEC 9126

L'analyse préliminaire conduite précédemment nous a permis de conclure qu'à première vue, le modèle proposé par le standard ISO/IEC 9126 semble être une fondation crédible pour l'ingénierie de qualité. Avant d'en arriver à une conclusion définitive, une étude approfondie se doit d'être entreprise.

Une analyse détaillée du standard ISO/IEC 9126-1 nous permet de relever les extraits de la norme suivants:

1. *The model must be usable in "defining quality requirements." (page iv, paragraphe 3)*
2. *The model must be "applicable to every kind of software." (page 1, paragraphe 3)*
3. *The model must "provide consistent terminology." (page 1, paragraphe 3)*
4. *The quality model must be usable for setting quality goals for software products and intermediate products." (page 6, paragraphe 8)*
5. *The model should be "hierarchically decomposed into a quality model composed of characteristics and subcharacteristics." (page 6, paragraphe 8)*
6. *The model must be predictive. This means that Internal Quality should be predictive of External Quality. Likewise, External Quality must be predictive of quality in use. (page 3, figure 2 et page 4, figure 3)*
7. *Conformance to the model shall be judged either by the usage of the characteristics and subcharacteristics or by a mapping to those characteristics and subcharacteristics. (page 2, clause 2)*

Il s'agit là de promesses faites aux utilisateurs quant aux caractéristiques que possède le modèle dans son ensemble. Les points 1, 4 et 6 demandent une vérification plus poussée.

Il se doit d'être souligné que la norme ISO/IEC 9126 est séparée en quatre parties interdépendantes. La première partie décrit le modèle de qualité dans son ensemble, alors que les trois autres définissent les aspects de qualité interne, externe et d'utilisation ainsi que les mesures y étant rattachées. Les trois aspects sont reliés ensemble par un modèle prédictif qui a pour point d'entrée la qualité d'utilisation (voir la Figure 12, page 40). En principe, la définition des exigences quant à la qualité d'utilisation peut permettre de découvrir une quantité considérable d'exigences ayant trait à la qualité externe. Il en va de même pour la qualité externe et la qualité interne. Lors de l'implémentation du logiciel, la qualité interne peut être utilisée pour prédire la qualité externe tandis quela qualité externe, quant à elle, peut servir à prédire la qualité d'utilisation. La qualité d'utilisation est donc le point d'entrée et le point de sortie de l'ingénierie de qualité. Elle représente en soi le résultat que les utilisateurs exigent et l'objet qu'ils mesurent. L'analyse qui suit porte donc sur cet aspect particulier du standard.

Afin de vérifier si les promesses d'ISO/IEC 9126 sont bel et bien réalisées dans l'aspect de la qualité d'utilisation, les mesures associées à ce dernier sont étudiées selon quatre angles d'analyse. Ces quatre angles, inspirés du standard IEEE 1061 (IEEE, 1998), sont les suivants :

- L'impact de la mesure est évalué. La mesure de l'impact permet d'apprécier la capacité de la mesure à discriminer la qualité.

- Le coût relatif de la mesure est rapidement estimé. Un coût trop élevé pourrait représenter un frein à l'utilisation.

- La pertinence de la mesure comme exigence de qualité est analysée.

- La place de la mesure dans le modèle prédictif préconisé par ISO/IEC 9126 est évaluée.

L'annexe 1 présente en détail les résultats de cette analyse. Le tableau XVIII (voir page 49), qui résume cette annexe, permet de constater que plus de la moitié des mesures ne sont pas clairement utilisables tôt dans le cycle de vie pour définir des exigences de qualité. Par contre, toutes les mesures hormis une semblent avoir leur place dans le modèle prédictif. Afin de pallier à cette situation, les annexes 2 et 3 présentent des améliorations possibles aux mesures jugées plus faibles.

Discussion, Conclusion et Recommandations

À la suite de l'analyse détaillée conduite précédemment, il ne fait aucun doute que le standard ISO/IEC 9126, dans son texte et son intention, appuie les idées et les besoins de l'ingénierie de qualité du logiciel. Cependant, les mesures, qui sont en fait l'implémentation de cette intention, ne répondent pas à ces besoins. De plus, d'autres études font la démonstration qu'il n'est pas clair que les mesures et les caractéristiques proposées forment un ensemble exhaustif. Finalement, il n'est pas possible de prouver formellement les liens entre la qualité d'utilisation et la qualité externe. À partir de ces constatations, trois recommandations visant à améliorer l'applicabilité de ISO/IEC 9126 à l'ingénierie de la qualité émergent:

- L'applicabilité des mesures pour la définition des exigences tôt dans le cycle de vie doit être améliorée.

- Vérifier l'exhaustivité de chacune des parties du standard avec des modèles spécifiques à chacun des aspects.

- Découvrir au sein d'études de cas concises les liens réels entre la qualité d'utilisation et la qualité externe en s'inspirant des liens proposés dans ce texte.

Cette recherche s'est montrée très critique à l'égard des mesures associées à l'aspect de la qualité d'utilisation du modèle ISO/IEC 9126. La question suivante est alors pertinente : est-ce que ce jugement sévère face aux mesures de la qualité d'utilisation fait que le modèle ISO/IEC 9126 n'est pas applicable à l'ingénierie de qualité du logiciel? La réponse à cette question est négative. Les mesures forment une partie dite « informative » du standard. En d'autres mots, leur utilisation n'est pas obligatoire. À la suite de cette recherche, il est possible de conclure que le modèle de qualité exposé dans ISO/IEC 9126-1 est applicable à l'ingénierie de qualité du logiciel selon les critères énumérés précédemment. Une amélioration des mesures ne fera qu'améliorer ce verdict et faciliter l'union du standard à ce nouveau courant de pensée.

AVANT-PROPOS

Le domaine de l'ingénierie du logiciel étant de façon prédominante anglophone, la langue anglaise fut choisie pour la rédaction de ce mémoire afin de l'offrir dans sa totalité à un public plus étendu. Un tel choix nous permettra également de publier plus rapidement les résultats présentés dans ce mémoire.

Nos recherches de maîtrise ont tout d'abord porté sur la contribution à la qualité du logiciel des méthodes dites “Agiles” par rapport aux méthodes traditionnelles d'ingénierie du logiciel. Comme il est souvent le cas dans le domaine de la recherche, nos investigations nous ont amenés à nous questionner sur l'influence des différents modèles de qualité sur le choix d'une méthode d'ingénierie. Il nous apparaissait alors possible que le clivage actuel entre les méthodes “Agiles” et les méthodes traditionnelles soit causé par une vision différente de la qualité. Les protagonistes des méthodes “Agiles” disent souvent que ce qui est important pour un client se résume à trois dimensions: le *temps* nécessaire à la construction du logiciel, le *coût* du logiciel, et la *qualité* du produit final. Il s'agit là d'une base de comparaison intéressante entre les différentes méthodes d'ingénierie. Bien que les deux premières variables soient quantifiables, une évaluation claire et précise des modèles de qualité permettant autant la définition que l'évaluation des exigences de ce paramètre n'a pas été trouvée dans la littérature. Ce travail de recherche vise à combler cette lacune.

REMERCIEMENTS

Je tiens à remercier le Conseil de recherches en sciences naturelles et en génie du Canada et l'Institut National d'Optique pour leur précieux soutien financier.

Je tiens également à remercier Monsieur Witold Suryn, professeur à l'École de Technologie Supérieure, pour avoir dirigé ce mémoire et mis à notre disposition les résultats de ses recherches sur l'ingénierie de la qualité de logiciels.

FOREWORD

The first research subject for this thesis was the influence of software development methodologies on the resulting software quality. The goal was to find a differentiating factor between the emerging “Agile” methodologies and the more traditional development methodologies. As is often the case in research, such investigations led to questions about the influence of quality models on the choice of an engineering methodology. It then seemed probable that the rift between Agile methodologies and more traditional methodologies could be caused by a different vision of what constitutes quality. The proponents of Agile methodologies often assert that there are three dimensions that are of importance for a client: the *time* it takes to produce software, the amount of money it *costs* to produce software, and finally the *quality* of the resulting product. These three dimensions seem to form a reasonable foundation on which to compare software engineering methodologies. While the first two dimensions can be quantified, a comprehensive evaluation of the different quality models that allow for both the definition and evaluation of the requirements related to the quality dimension has not been found in the literature. This research aims to close this gap.

TABLE OF CONTENTS

LISTE OF TABLES

Page

LIST OF FIGURES

Page

INTRODUCTION

Over the last decade, the general focus of the software industry has shifted from providing ever more *functionalities* to improving what has been coined the *user experience.* The user experience refers to characteristics such as ease of use, security, stability, etc. Improvements in such areas lead to an improved *quality* as perceived by the end users. Some software products, most notably Microsoft's next iteration of their Windows operating system, have been delayed by as much as two years in order to improve their quality. There is no doubt that software quality is becoming an increasingly important subject in software engineering.

Traditionally, software requirements have been classified either as *functional* or *non-functional* with eventual notions of quality hidden in the latter. As the industry focus is shifting from *functionality* to *improving quality*, a new category of requirements are emerging.

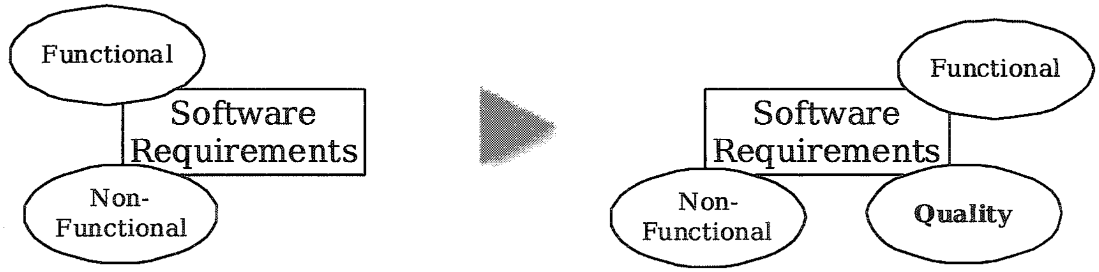


Figure 1 Focus is moving towards quality requirements

In order to define these new quality requirements, quality itself must be defined. The role of the definition of quality is filled by what is called a quality model. Unfortunately, the push towards software quality that can be observed in the industry today is lacking a solid foundation in the form of an agreed upon quality model that can be used not only to evaluate software quality, but also to specify it.

The primary objective of this research is to identify a quality model that can serve as a basis for the improvement of software quality in a continuous, systematic, disciplined and quantifiable way (Suryn, 2003). In order to attain this objective, the following process will be followed:

- A review of the literature will allow for the observation of the state of the art in the industry and the research community with respect to software quality. This part of the thesis will identify possible causes for lacking software quality in the industry and further stress the need for a solid foundation to the engineering of quality.

- Using the premises established in the review of the literature, four quality models recognized today will be described and evaluated with respect to their suitability for the improvement of software quality. One model will be selected for further analysis.

- An in-depth analysis and evaluation of the model that seems the most suited for the improvement of quality will be conducted.

- The result of the analysis will be presented in form of a discussion and recommendations.

Limitations of the research

The following limitations apply to this research:

- The goal of the preliminary analysis is to identify the best possible quality model for Software Quality Engineering. As there are many quality models, the review of all the quality models is beyond the scope of this project. Rather, four quality models were selected for this preliminary analysis. McCall's (1977) and Boehm's (1978) models were selected for their historical importance and because they are at the root of some corporate quality models. Dromey's (1995) model was selected because it presented a novel approach to software quality. Finally, the ISO/IEC 9126 (2001a) quality model was selected because of its importance as an international standard.

- The in-depth analysis could be subject to further limitations, depending on the model that will be selected in the preliminary analysis. Should such limitations be necessary, they will be detailed at the beginning of the in-depth analysis.

CHAPTER 1

LITERATURE REVIEW

The software engineering industry has long been diagnosed with a "*quality* problem" (Glass, 1997; NIST, 2002; SEI, 2002). This quality problem can take different incarnations: from monumental disasters related to software (Glass, 1997) to disastrous economic losses. For example, a NIST report clearly blames lacking software quality for losses of up to 60 billion US dollars in 2002 in the United States alone (NIST, 2002). Discussion on how to resolve the quality problem in software engineering leads to heated and interesting debates because what exactly constitutes the *quality* of a product is often the subject of hot debate. The reason the word *quality* is so controversial is that people fail to agree on what it means. For some it is "[the] degree to which a set of inherent characteristics fulfills requirements" (ISO/IEC 1999b) while for others it can be synonymous with "customer value" (Highsmith, 2002), or even "defect levels" (Highsmith, 2002). A possible explanation as to why any of these definitions fail to garner a consensus is that they generally fail to recognize the different perspectives of *quality*. Kitchenham and Pfleeger (1996), by reporting the teachings of David Garvin, report on the 5 different perspectives of quality:

- The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality, it is "something toward which we strive as an ideal, but may never implement completely." (Kitchenham & Pfleeger, 1996);

- The user perspective is concerned with the appropriateness of the product for a given context of use. Kitchenham and Pfleeger further note that "whereas the transcendental view is ethereal, the user view is more concrete, grounded in the product characteristics that meet user's needs.";

- The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO 9001, which defines quality as "[the] degree to which a set of inherent characteristics fulfills requirements" (ISO/IEC 1999b). Other models, like the Capability Maturity Model (CMM) state that the quality of a product is directly related to the quality of the engineering process, thus emphasizing the need for a manufacturing-like process;

- The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product. Such an approach often leads to a bottom-up approach to software quality: by measuring some attributes of the different components composing a software product, a conclusion can be drawn as to the quality of the end product;
- The final perspective of quality is value-based. This perspective recognizes that the different perspectives of quality may have a different importance, or value, to various stakeholders.

One could argue that in a world where conformance to ISO and IEEE standards is increasingly present in contractual agreements and used as a marketing tool (Adey & Hill, 2000), all the perspectives of quality are subordinate to the manufacturing view. This importance of the manufacturing perspective has increased throughout the years through works like *Quality is Free* (Crosby, 1979) and the popularity of movements like *Six-Sigma* (Biehl, 2001). The predominance of the manufacturing view in Software Engineering can be traced back to the 1960s, when the US Department of Defense and IBM gave birth to Software Quality Assurance (Voas, 2003). This has led to the belief that adherence to a development process, as in manufacturing, will lead to a quality product. The corollary to this belief is that process improvement will lead to improved product quality. According to many renowned researchers, this belief is false, or at least flawed. Geoff Dromey states:

> *"The flaw in this approach [that you need a quality process to produce a quality product] is that the emphasis on process usually comes at the expense of constructing, refining, and using adequate product quality models." (Dromey, 1996)*

Kitchenham and Pfleeger reinforce this opinion by stating:

> *"There is little evidence that conformance to process standards guarantees good products. In fact, the critics of this view suggest that process*

> *standards guarantee only uniformity of output [...]" (Kitchenham & Pfleeger, 1996)*

Furthermore, data available from so-called Agile (Highsmith, 2002) projects show that high quality is attainable without following a manufacturing-like approach.

However, recent studies conducted at Motorola (Eickelman, 2003; Diaz & Sligo, 1997) and Raytheon (Haley, 1996) show that there is indeed a correlation between the maturity level of an organization as measured by the Capability Maturity Model and the *quality* of the resulting product. These studies provide data on how a higher maturity level (as measured by the CMM) can lead to:

- Improved error/defect density (i.e. the error/defect density lowers as maturity improves)
- Lower error rate
- Lower cycle time (time to complete parts of the lifecycle)
- Better estimation capability

From these results, one could conclude that the "quality problem" is non-existent, that it can easily be solved by following a mature process. However, these measured improvements are directly related to the manufacturing perspective of quality. Therefore, such quality improvement efforts fail to address the other perspectives of quality. This might be one of the reasons that some observers of the software development scene perceive the "quality problem" as one of the main failings of the software engineering industry. Furthermore, studies show that improvement efforts grounded in the manufacturing perspective of quality are difficult to scale down to smaller projects and/or smaller teams (Laitinen, 2000; Boddie, 2000). Indeed, rather than being scaled down in smaller projects, these practices are simply not performed.

Over the recent years, researchers have proposed new models that try to encompass more perspectives of quality than just the manufacturing view. One such model was proposed by Geoff Dromey (1995; 1996). Dromey's view of the quality of the end product is that it is directly related to the quality of the artifacts that are a by-product of the process being followed. Therefore, he developed different models that can be used to evaluate the quality of the requirements model, the design model and the resulting software. The reasoning is that if quality artifacts are conceived and produced throughout the lifecycle, then the end product will manifest attributes of quality. This approach can clearly be linked to the product perspective of quality with elements from the manufacturing view. This is certainly a step forward from the manufacturing-only approach described above, but it fails to view the engineering of quality as a process that covers all the perspectives of quality. Pfleeger (2001) warns against approaches that focus only on the product perspective of quality:

> *"This view [the product view] is the one often advocated by software metrics experts; they assume that good Internal Quality indicators will lead to good external ones, such as reliability and maintainability. However, more research is needed to verify these assumptions and to determine which aspects of quality affect the actual product's use."*

The above observations illustrate the disagreements that exist in both the research community and the industry on the subject of software quality. One thing is certain however: it is difficult to measure something that has not been thoroughly defined. Furthermore, in this day and age of rationalization, it is doubtful that something that is not specifically required will be implemented. This reasoning leads to the first premiss that is the basis of this work:

A possible contribution to a complete solution to the quality problem in software engineering is to establish quality requirements.

Premiss I

To the extent of the knowledge of the author, to date no methodology for establishing quality requirements exists. In fact, quality requirements are not even a recognized body of knowledge. A survey of the most popular books on software engineering, books that are used to teach software engineering, shows that they fail to acknowledge quality requirements.

Table I

Survey of books on software engineering
(Adapted from Suryn (2003))

Author	*Book*	*Year*
Van Vliet	Software Engineering (2nd edition)	2003
Pfleeger	Software Engineering (3rd edition)	2002
Leffingwell/Widrig	Managing Software Requirements, 1st and 2nd edition	1993, 2003
Lauesen	Software Requirements	2002
Budgen	Software Design	2003
Humphrey	A Discipline for Software Engineering	2002
Ghezzi	Fundamentals of Software Engineering	2002
Kendall	Systems Analysis and Design	2002
Donaldson	Successful Software Development	2000
Jarvis	Inroads to Software Quality	1997
Kan	Metrics and Models in Software Quality Engineering	2003
N/A	SWEBOK	2003

Suryn (2003) in his analysis observes:

- None recognizes the implementation of quality as an effort that closely follows the life cycle;
- Only one recognizes that the implementation of quality is part of the engineering process;
- None recommends that quality requirements be modeled at the same time as functional requirements;
- None teaches how to implement quality in the product;
- None offers advice, tool support or methodology to quality engineers;

In fact, most of these books view quality from the manufacturing and product perspectives. The "quality problem" is thus easily solved by following a mature process and performing reviews, tests and inspections. Leffingwell and Widrig (1999), in their book on requirements engineering, only mention quality as an attribute of the use case model, thus approaching Dromey's view of quality (i.e. something that should be considered for all the products of the software engineering process). There is not a single word on the quality attributes that should be a part of the end product. Sadly, this may be considered not only as the state of the practice, but also as the state of the art.

Consequently, one could question why while most people would agree on the importance of software quality, very few tie its implementation to the software life cycle. The unification of the software life cycle with the engineering of quality would manifest itself in part by the establishment of quality requirements. This leads to the second premiss that justifies the research presented in this thesis:

> **A possible reason for the absence of quality requirements is that no quality model has yet been identified to serve as a foundation for their definition.**
>
> *Premiss II*

This premiss does not state that there exists no model suitable for establishing quality requirements, but simply that no such model has been clearly identified in the literature. Such a model could serve as a basis for the definition of quality requirements.

The goal of a quality model is in essence to define quality. While specific definitions have been established for given contexts, there is no consensus as to what constitutes quality in the general sense in software engineering. In such a situation, a model that encompasses as many perspectives of quality would prove useful. The following premiss defines the first requirement that a suitable model for the identification of quality requirements should respect.

> **A quality model that is to be used as the foundation for the definition of quality requirements should acknowledge all the perspectives of quality, namely transcendental, user, manufacturing, product and value-based.**
>
> *Premiss III*

Another requirement that a quality model should respect is to have the ability to support both the definition of quality requirements and their subsequent evaluation. This can be explained by referring to the manufacturing perspective of quality, which states that quality is conformance to requirements. Conformance to requirements implies that something has to be defined and measured. The following premiss states this concisely.

> **A quality model that is to be used as the foundation for the definition of quality requirements should help in both the specification of quality requirements and the evaluation of software quality. In other words, it should be usable from the top of the development process to the bottom, and from the bottom to the top.**
>
> *Premiss IV*

Should such a model be identified, it could lead to a new approach to quality in software engineering. Using such a model to identify and specify quality requirements at the beginning of the lifecycle would mark a transition from a reactive approach to quality towards a methodology that is proactive in the engineering of quality into software. Such a methodology is taking shape in the teachings of Suryn (2003). He has coined such a methodology *Software Quality Engineering* and defined it as follows:

> *"The application of a continuous, systematic, disciplined, quantifiable approach to the development and maintenance of quality of software products and systems; that is, the application of quality engineering to software."*

This can be contrasted with traditional software engineering which is comprised of the following activities[1] (Pfleeger, 2001; Pressman, 2001; Leffingwell & Widrig, 1999):

[1] This representation of the activities included in the development process is not meant to indicate that the Waterfall model should be used. The observations made in this section are applicable to most, if not every, process model.

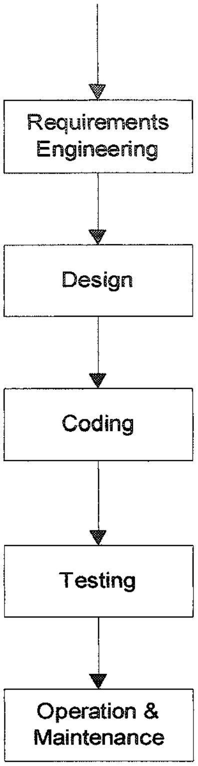


Figure 2 Traditional activities in the software development lifecycle (ISO/IEC 2000)

In most cases, these activities will produce artifacts at their completion. For example, the accomplishment of the Requirements Engineering process will result in the production of a Software Requirements Specification document (SRS). The Design phase will end with the release of a design document. The approach to achieving quality until now has been to measure the quality of these individual artifacts with respect to a certain evaluation model and use the results as a prediction of the end product's quality.

As was previously mentioned, several researchers have expressed doubts about the validity of such an approach. An approach as the one illustrated in Figure 3 can be defined as bottom-up; by building quality components, it is assumed that the whole will be of quality. On the other end, software quality engineering also approaches the problem from the top to the bottom, as is illustrated in Figures 4 and 5.

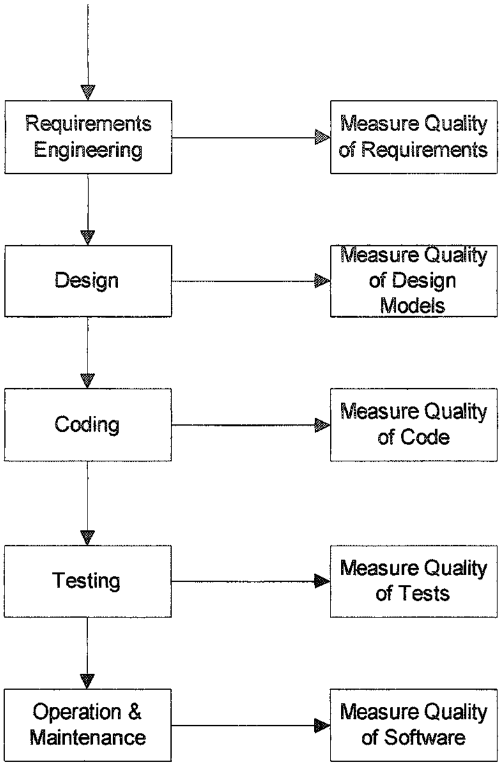


Figure 3 Traditional approach to quality prediction
Adapted from Dromey (1996)

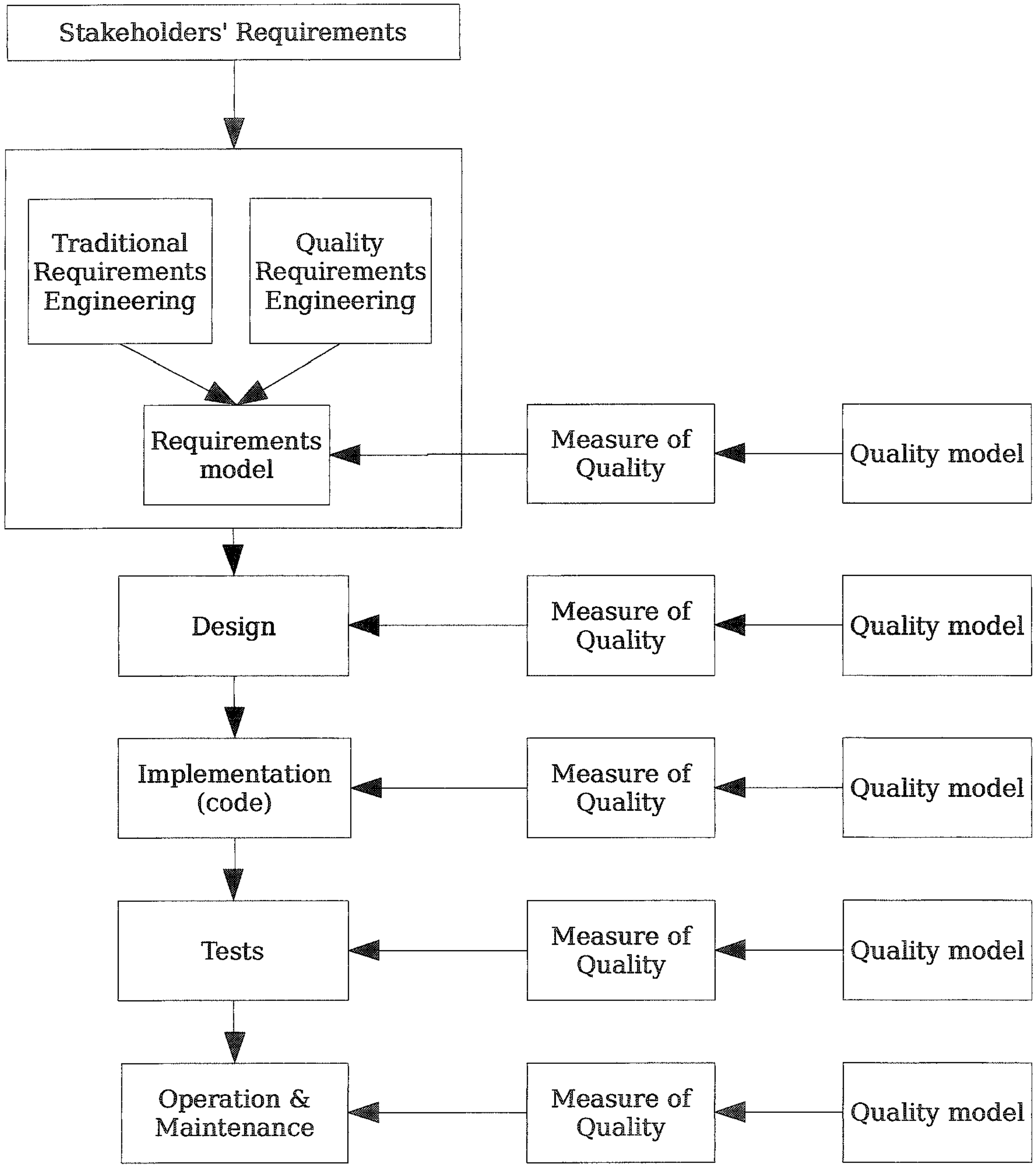
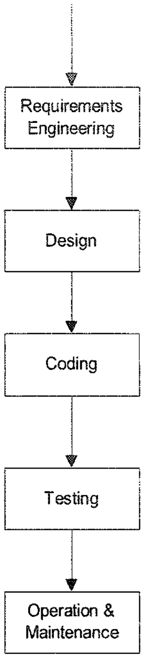


Figure 4 An approach to Quality Engineering - 1



1-From top to bottom, use a quality model to specify quality needs and requirements

2-From bottom to top, use a quality model to evaluate the quality and determine if it is sufficient to meet quality needs and requirements

Figure 5 An approach to Quality Engineering - 2

It is an emerging research hypothesis that the way to resolve the quality problem in software engineering lies in such an approach to software quality engineering (Suryn, 2003). Quality concerns should be addressed at the beginning of and throughout the lifecycle.

More than 400 years ago, René Descartes, the famous French philosopher, is reported to have said that "*it is far better never to contemplate investigating the truth about any matter than to do so without a method*". Carefully following such an advice, it is the opinion of the author that software quality engineering, in order to advance in its maturity, should select an appropriate quality model as a long-term foundation.

This thesis presents the search for a long-term foundation to software quality engineering.

CHAPTER 2

SELECTION OF A QUALITY MODEL

Every methodology that aspires to be used as a foundation for quality engineering should be firmly grounded in an appropriate quality model or framework. As a premiss to this work, it was stated that an appropriate quality model for software quality engineering should be usable from top to bottom and from bottom to top. IEEE Std 1061-1998 (IEEE, 1998) provides guidance on the usage of such a quality framework:

> *From top to bottom the [quality] framework facilitates:*
>
> *-Establishment of quality requirements factors, by customers and managers early in a system's life cycle;*
>
> *-Communication of the established quality factors, in terms of quality subfactors, to the technical personnel;*
>
> *-Identification of metrics[2] that are related to the established quality factors and quality subfactors.*
>
> *From bottom to top the [quality] framework enables the managerial and technical personnel to obtain feedback by*
>
> *-Evaluating the software products and processes at the metrics level;*
>
> *-Analyzing the metric values to estimate and assess the quality factors.*

[2] In 2002, the ISO/IEC JTC1 sub-committee SC7 – Systems and Software Engineering – replaced the term "metric" by "measure" to align its vocabulary with the one used in metrology. This thesis will use the term measure whenever possible.

As was pointed out in the previous section, quality frameworks tend to be used in a bottom to top approach. The prevalence of the manufacturing and product perspectives encourage frequent measurement of internal attributes as a control variable. Furthermore, it is possible to link prevalence of the bottom to top approach to the following points:

- Software engineering tools automatically measure some quality attributes. For example, tools like Borland's Together automatically measure quality attributes for the design and implementation of software.
- Up till now, quality models have emphasized the evaluation of quality, rather than helping decide which quality attributes should be emphasized.
- The assumption that the quality of the individual artifacts will be indicative of the end product's quality. This could be traceable to an over-emphasis on the product perspective of quality, as was explained previously.
- An emphasis on the manufacturing perspective of quality will lead to an evaluation of some specific quality measures throughout the life cycle. Those specific measures include but are not limited to: error/defect density, error rate and cycle time.

On the other end, it is suggested as a premiss to this research that quality engineering may be accomplished by a methodology that leverages the power of *both* the top to bottom and bottom to top approaches. Starting with an inadequate model will render the task of quality engineering too difficult to attain. All models explicitly or implicitly support the bottom up approach to quality engineering, because at some point measurements are necessary. What is needed is a model that explicitly supports the top to bottom approach to quality engineering.

As a first step towards reaching the stated objective of identifying a quality model suitable for software quality engineering, this section reviews the most popular quality models with respect to the following criteria inspired from the IEEE standard (IEEE, 1998):

- **Model selection question 1:** Can the framework be used by stakeholders to set quality factors early in a system's lifecycle?
- **Model selection question 2:** Can the established quality requirements based on the model be effectively communicated to the technical personnel?
- **Model selection question 3:** Is it possible to identify measures related to the establishment of quality factors and quality subfactors?

2.1 McCall's quality model

2.1.1 Description

McCall (McCall, Richards & Walters, 1977) introduced his quality model in 1977. According to Pfleeger (2001), it was one of the first published quality models. Figure 6 presents this quality model. Each quality factor on the left hand side of the figure represents an aspect of quality that is not directly measurable. On the right hand side are the measurable properties that can be evaluated in order to quantify the quality in terms of the factors. Table II presents the quality factors while Table III describes the measurable properties. McCall proposes a subjective grading scheme ranging from 0 (low) to 10 (high).

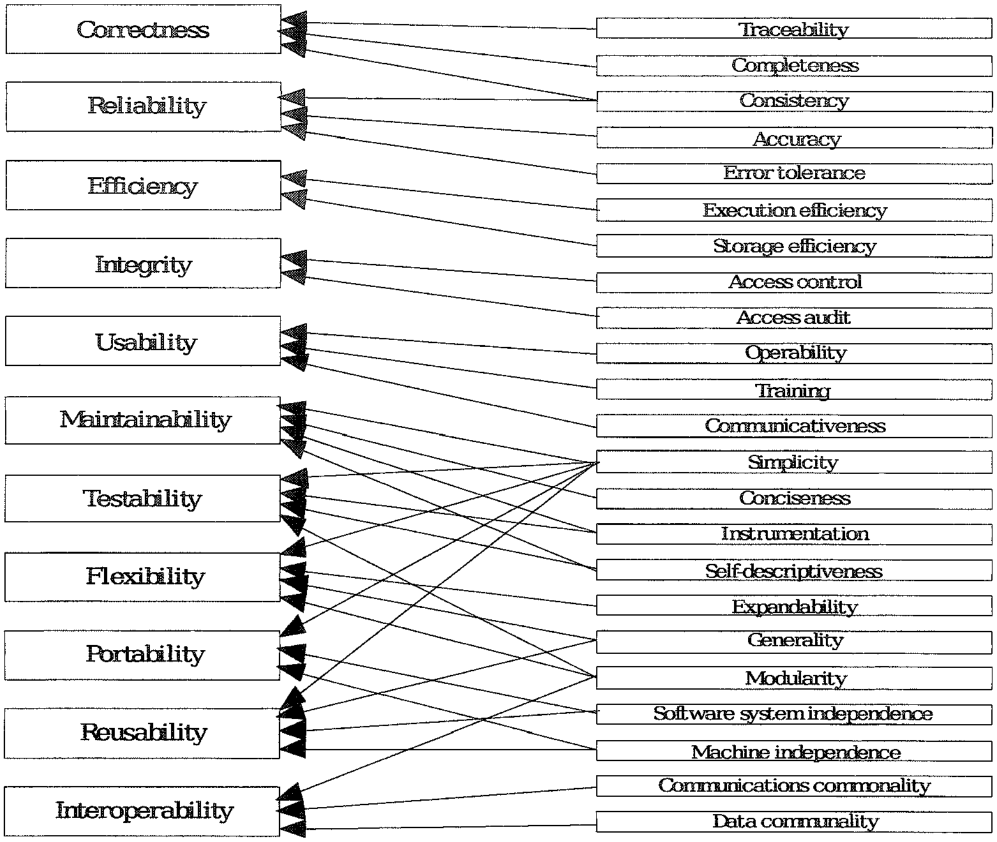


Figure 6 McCall's quality model
Adapted from Pleeger (2003) and McCall et al. (1977)

Table II

McCall's quality factors
(Adapted from Pressman (2001))

Quality Factor	***Definition***
Correctness	The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform its function.
Integrity	Extent to which access to software or data by unauthorized persons can be controlled.
Usability	Effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability	Effort required to locate and fix an error.
Flexibility	Effort required to modify an operational program.
Testability	Effort required to test a program to ensure that it performs its intended function.
Portability	Effort required to transfer the program from one hardware and/or software system environment to another.
Reusability	Extent to which a program can be reused in other applications – related to the packaging and scope of the functions that the program performs.
Interoperability	Effort required to couple one system to another.

Table III

McCall's measurable properties
(Adapted from Pressman (2001))

Measurable Property	***Definition***
Auditability	The ease with which conformance to standards can be checked.
Accuracy	The precision of computations and control.
Communication commonality	The degree to which standard interfaces, protocols, and bandwidth are used.
Completeness	The degree to which full implementation of required function has been achieved.
Conciseness	The compactness of the program in terms of lines of code.
Consistency	The use of uniform design and documentation techniques throughout the software development project.
Data commonality	The use of standard data structures and types throughout the program.
Error tolerance	The damage that occurs when the program encounters an error.
Execution efficiency	The run-time performance of a program.
Expandability	The degree to which architectural, data, or procedural design can be extended.
Generality	The breadth of potential application of program components.
Hardware independence	The degree to which software is decoupled from the hardware on which it operates.
Instrumentation	The degree to which the program monitors its own operation and identifies errors that do occur.
Modularity	The functional independence of program components.
Operability	The ease of operation of a program.
Security	The availability of mechanisms that control or protect programs and data.
Self-documentation	The degree to which the source code provides meaningful documentation.
Simplicity	The degree to which a program can be understood without difficulty.

Table III (continued)

Measurable Property	***Definition***
Software system independence	The degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints.
Traceability	The ability to trace a design representation of actual program component back to requirements.
Training	The degree to which the software assists in enabling new users to apply the system.

2.1.2 Discussion and evaluation

As was noted previously, this is one of the first documented attempt at defining a universal quality model for software systems. Some other corporate models like MITRE's SQAE (Martin & Shaffer, 1996) are partially based on this work.

Pressman notes that "*unfortunately, many of the metrics*[3] *defined by McCall et al. can be measured only subjectively*" (Pressman, 2001). It is therefore difficult to use this framework to set precise and specific quality requirements. Furthermore, some of the factors and measurable properties, like *traceability* and *self-documentation* among others, are not really definable or even meaningful at an early stage for non-technical stakeholders.

Pressman states that "*The metrics may be in the form of a checklist that is used to "grade" specific attributes of the software*". This statement highlights that McCall's quality model is better suited to the bottom to top evaluation of quality rather than the specification of quality needs.

[3] This lapse of Pressman needs to be pointed out. McCall's model does not define metrics, but measurable properties that can be measured through the use of metrics.

2.1.3 Conclusion

This model is not applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top to bottom approach to quality engineering. It is therefore not suited as a foundation for software quality engineering according to the stated premises.

Table IV

Evaluation of McCall's model

Model selection question	***Answer***
Can the framework be used by stakeholders to set quality factors early in a system's lifecycle?	WITH A CERTAIN LEVEL OF DIFFICULTY. It is difficult to use this model to specify high level quality needs at the beginning of the lifecycle. The high level attributes can be mostly considered as things every software product should exhibit.
Can the established quality requirements based on the model be effectively communicated to the technical personnel?	PROBABLY. The model is already quite technical and uses terms that should be used by technical personnel.
Is it possible to identify measures related to the establishment of quality factors and quality subfactors?	WITH A CERTAIN LEVEL OF DIFFICULTY. Some of the measurable properties are loosely defined.

2.2 Boehm's quality model

2.2.1 Description

Boehm's quality model improves upon the work of McCall and his colleagues (Boehm, Brown, Kaspar, Lipow & MacCleod, 1978). As Figure 7 shows, this quality model loosely retains the factor-measurable propety arrangement. However, for Boehm and his colleagues, the prime characteristic of quality is what they define as “general utility”. According to Pfleeger (2001), this is an assertion that first and foremost, a software system must be useful to be considered a quality system. For Boehm, general utility is composed of as-is utility, maintainability and portability (Boehm et al., 1976):

- *How well (easily, reliably, efficiently) can I use it [software system] as-is?*
- *How easy is it to maintain (understand, modify, and retest)?*
- *Can I still use it if I change my environment?*

If the semantics of McCall's model are used as a reference, the quality factors could be defined as: *Portability, Reliability, Efficiency, Human Engineering, Testability, Understandability* and *Modifiability.* These factors can be decomposed into measurable properties such as *Device Independence, Accuracy, Completness, etc.* Portability is somewhat incoherent in this classification as it acts both as a top level component of general utility, and as a factor that possesses measurable attributes.

The definitions for the factors and measurable attributes are given in Table V and Table VI respectively.

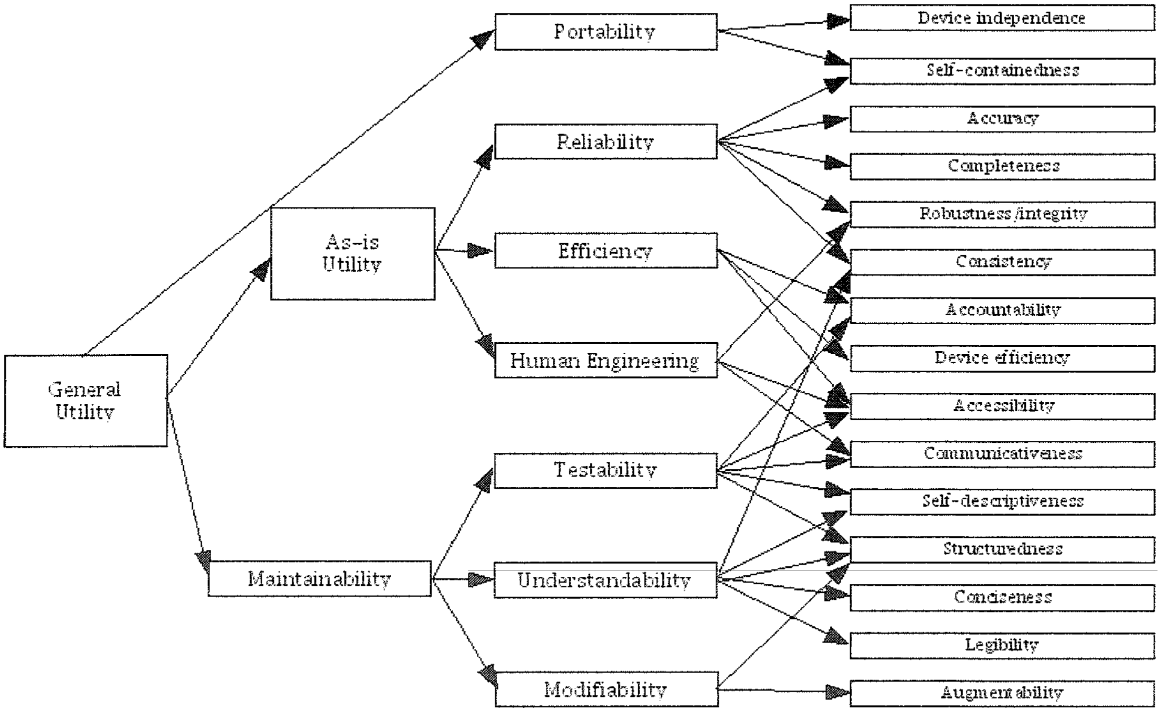


Figure 7 Boehm's quality model
Adapted from Pfleeger (2001), Boehm et al. (1976; 1978)

Table V

Boehm's quality factors
(Adapted from Boehm et al. (1976))

Quality Factor	*Definition*
Portability	Code possesses the characteristic of portability to the extent that it can be operated easily and well on computer configurations other than the current one.
Reliability	Code possesses the characteristic of reliability to the extent that it can be expected to perform its intended functions satisfactorily.
Efficiency	Code possesses the characteristic of efficiency to the extent that it fulfills its purpose without waste of resource.
Human Engineering	Code possesses the characteristic of human engineering to the extent that it fulfills its purpose without wasting the users' time and energy, or degrading their morale.
Testability	Code possesses the characteristic of testability to the extent that it facilitates the establishment of verification and supports evaluation of its performance.
Understandability	Code possesses the characteristic of understandability to the extent that its purpose is clear to the inspector.
Modifiability	Code possesses the characteristic of modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined.
Maintainability	Code possesses of maintainability to the extent that it facilitates updating to satisfy new requirements or to correct deficiencies.

Table VI

Boehm's measurable properties
(Adapted from Boehm et al. (1976))

Measurable Property	***Definition***
Device independence	Code possesses the characteristic of device-independance to the extent that it can be executed on computer hardware configurations other that its current one.
Self-containedness	Code possesses the characteristic of self-containedness to the extent that it performs all its explicit and implicit functions within itself.
Accuracy	Code possesses the characteristic of accuracy to the extent that its outputs are sufficiently precise to satisfy their intended use.
Completeness	Code possesses the characteristic of completeness to the extent that all its parts are present and each part is fully developed.
Robustness/integrity	Code possesses the characteristic of robustness to the extent that it can continue to perform despite some violation of the assumptions in its specification.
Consistency	Code possesses the characteristic of internal consistency to the extent that it contains uniform notation, terminology and symbology within itself, and external consistency to the extent that the content is traceable to the requirements.
Accountability	Code possesses the characteristic of accountability to the extent that its usage can be measured. This means that critical segments of code can be instrumented with probes to measure timing, whether specified branches are exercised, etc.
Device efficiency[4]	Code possess the characteristic of device efficiency to the extent that it fulfills its purpose without waste of hardware resources.
Accessibility	Code possesses the characteristic of accessibility to the extent that it facilitates selective use of its parts.
Communicativeness	Code possesses the characteristic pf communicativeness to the extent that it facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and useful.

[4] Boehm et al. (1976) does not define this property in the appendix to his paper. However, the definition can be reconstructed from the definition of the Efficiency characteristic.

Table VI (continued)

Measurable Property	***Definition***
Self-descriptiveness	Code possesses the characteristic of self-descriptiveness to the extent that it contains enough information for a reader to determine or verify its objectives, assumptions, constraints, inputs, outputs, components, and revision status.
Structuredness	Code possesses the characteristic of structuredness to the extent that it possesses a definite pattern of organization of its interdependent parts.
Conciseness	Code possesses the characteristic of conciseness to the extent that excessive information is not present.
Legibility	Code possesses the characteristic of legibility to the extent that its function is easily discerned by reading the code.
Augmentability	Code possesses the characteristic of augmentability to the extent that it can easily accommodate expansion in component computational functions or data storage requirements.

2.2.2 Discussion and evaluation

It is interesting to note that in opposition to McCall's model, Boehm's model is decomposed in a hierarchy that at the top addresses the concerns of end-users while the bottom is of interest to technically inclined personnel. However, this interest wanes when one reads Boehm's definition of the characteristics of software quality. Except for *General Utility* and *As-is Utility*, all definitions begin with "*Code possesses the characteristic [...]*". The measurable properties and characteristics therefore concentrate on highly technical details of quality that are difficult to grasp for non-technical stakeholders that are typically involved early in the software lifecycle. The characteristics *General Utility* and *As-is Utility* are too generic and imprecise to be useful for defining verifiable requirements. Like the McCall model, this model is mostly useful for a bottom to top approach to software quality (i.e. it can effectively be used to define measures of software quality, but is more difficult to use to specify quality requirements).

While this model is a step forward in the sense that it provides basic support for a top to bottom approach to software quality, this support is too ephemeral to be considered as a solid foundation for quality engineering.

2.2.3 Conclusion

Table VII

Evaluation of Boehm's model

Model selection question	***Answer***
Can the framework be used by stakeholders to set quality factors early in a system's lifecycle?	WITH A CERTAIN LEVEL OF DIFFICULTY. It is difficult to use this model to specify high level quality needs at the beginning of the lifecycle. The high level attributes can be mostly considered as things every software product should exhibit. It is difficult to imagine asking an end user the following question: "On a grade from one to ten, please rate how useful you would like the system to be?". How is the usefulness measured then?
Can the established quality requirements based on the model be effectively communicated to the technical personnel?	PROBABLY. The model is already quite technical and uses terms that should be used by technical personnel.
Is it possible to identify measures related to the establishment of quality factors and quality subfactors?	WITH A CERTAIN LEVEL OF DIFFICULTY. Measures can be defined from the measurable properties.

This model is not applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top to bottom approach to quality engineering. It is therefore not suited as a foundation for software quality engineering according to the stated premises.

2.3 Dromey's quality model

2.3.1 Description

Dromey's (1995) model takes a different approach to software quality then the two previously presented models. For Dromey, a quality model should clearly be based upon the product perspective of quality:

> *"What must be recognized in any attempt to build a quality model is that software does not directly manifest quality attributes. Instead it exhibits* ***product*** *characteristic that imply or contribute to quality attributes and other characteristics (****product*** *defects) that detract from the quality attributes of a* ***product****. Most models of software quality fail to deal with the* ***product*** *characteristics side of the problem adequately and they also fail to make the direct links between quality attributes and corresponding* ***product*** *characteristics." (Dromey, 1995) (Emphasis added to support the argument)*

Dromey has built a quality evaluation framework that analyzes the quality of software *components* through the measurement of tangible quality properties (Figure 9). Each artifact produced in the software lifecycle can be associated with a quality evaluation model. Dromey gives the following examples of what he means by software components for each of the different models:

- *Variables, functions, statements, etc.* can be considered components of the implementation model;
- A *requirement* can be considered a component of the requirements model;
- A *module* can be considered a component of the design model;
- Etc.

According to Dromey (1995), these components all possess intrinsic properties that can be classified into four categories:

- Correctness: Evaluates if some basic principles are violated.

- Internal: Measure how well a component has been deployed according to its intended use.
- Contextual: Deals with the external influences by and on the use of a component.
- Descriptive: Measure the descriptiveness of a component (for example, does it have a meaningful name?).

These properties are used to evaluate the quality of the components. This is illustrated in Figure 8 for a variable component present in the implementation model.

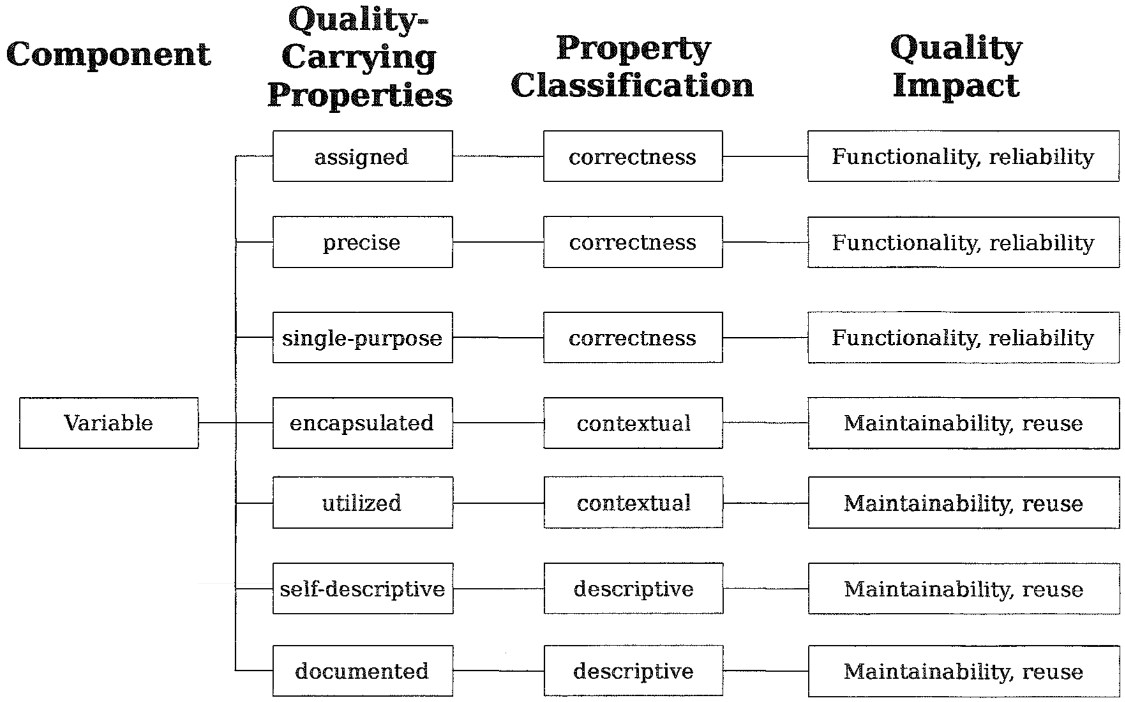


Figure 8 Quality evaluation of a variable component

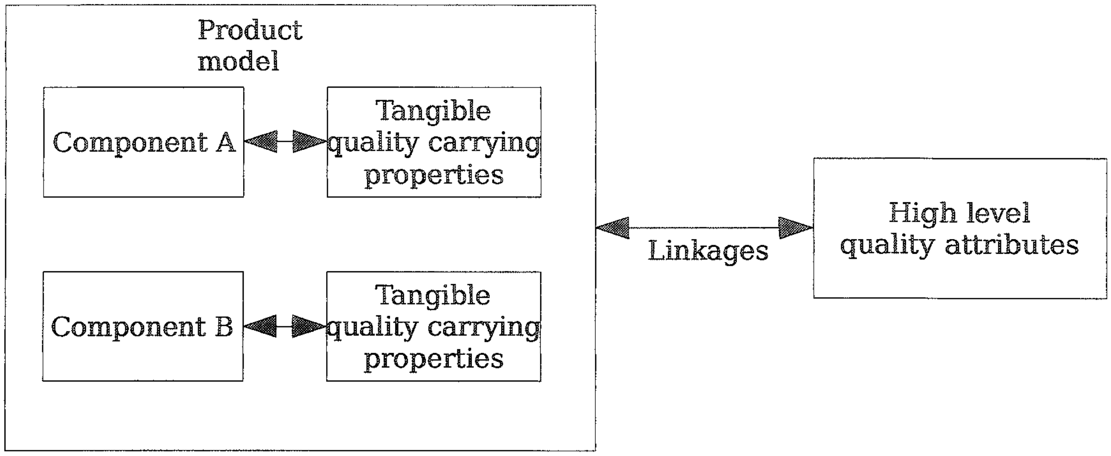

Product model
Component A
Tangible quality carrying properties
Component B
Tangible quality carrying properties
Linkages
High level quality attributes

Figure 9 Dromey's Quality Model

2.3.2 Discussion and evaluation

It seems obvious from the inspection of the previous figures that Dromey's model is focused on the minute details of quality. This is stated explicitly:

> *"What we can do is identify and build in a consistent, harmonious, and complete set of product properties (such as modules without side effects) that result in manifestations of reliability and maintainability." (Dromey, 1996)*

For Dromey, the high level characteristics of quality will manifest themselves if the components of the software product, from the individual requirements to the programming language variables[5], exhibit quality carrying properties. Dromey's hypothesis should be questioned. If all the components of all the artifacts produced during the software lifecycle exhibit quality carrying properties, will the resulting product manifest characteristics such as maintainability, functionality, and others?

The following analogy will be useful in answering this question.

> *If you buy the highest quality flour, along with the highest quality apples and the highest quality cinnamon, will you automatically produce an apple pie that is of the highest quality?*

[5] Dromey's description of his quality evaluation framework begins with requirements and ends with the implementation.

The answer is obviously negative. In addition to quality ingredients, at least three more things are needed in order to produce an apple pie of the highest quality.

- A recipe (i.e. an overall architecture and an execution process). Dromey acknowledges this by identifying process maturity as a desirable high level characteristic. However, it is only briefly mentioned in both his publications on the subject (Dromey, 1995; Dromey, 1996).
- The consumer's tastes must be taken into account. In order for the result to be considered of the highest quality by the consumer, it needs to be tuned to his tastes. This is akin to what is commonly called *user needs* in software engineering. User needs are completely ignored by Dromey. However, as it was demonstrated in the introduction, they are an integral and indissociable part of software quality.
- Someone with the qualifications and the tools to properly execute the recipe.

While Dromey's work is interesting from a technically inclined stakeholder's perspective, it is difficult to see how it could be used at the beginning of the lifecycle to determine user quality needs. Dromey (1995) states that software quality "*must be considered in a systematic and structured way, from the tangible to the intangible*". By focusing too much on the tangible, Dromey fails to build a model that is meaningful for stakeholders typically involved at the beginning of the lifecycle. Do end users care about the variable naming convention or module coupling? In most cases, it is doubtful that this question can be answered affirmatively. Therefore, this model is rather unwieldy to specify user quality needs. This does not mean that it cannot be useful later on as a checklist for ensuring that product quality is up to standards. It can definitely be classified as a bottom to top approach to software quality.

2.3.3 Conclusion

This model is not applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top to bottom approach to quality engineering.

Table VIII

Evaluation of Dromey's model

Model selection question	*Answer*
Can the framework be used by stakeholders to set quality factors early in a system's lifecycle?	NO. It is difficult to use this model to specify high level quality needs at the beginning of the lifecycle. Dromey's model defines what could be termed as software quality checklists for individual components. However, these are things that are usually of little concern for a customer.
Can the established quality requirements based on the model be effectively communicated to the technical personnel?	YES. Checklist can be provided to technical personnel to ensure that they perform the work correctly.
Is it possible to identify measures related to the establishment of quality factors and quality subfactors?	YES. Checklists are convenient to verify.

2.4 ISO/IEC 9126 quality model

2.4.1 Description

In 1991, the International Organization for Standardization introduced a standard named ISO/IEC 9126 (1991): Software product evaluation - Quality characteristics and guidelines for their use. This standard aimed to define a quality model for software and a set of guidelines for measuring the characteristics associated with it. ISO/IEC 9126 quickly gained notoriety with IT specialists in Europe as the best way to interpret and measure quality (Bazzana, Anderson & Jokela, 1993). However, Pfleeger (2001) reports some important problems associated with the first release of ISO/IEC 9126:

- There are no guidelines on how to provide an overall assessment of quality.
- There are no indications on how to perform the measurements of the quality characteristics.

- Rather than focusing on the user view of software, the model's characteristics reflect a developer view of software.

According to Pfleeger, this first incarnation of ISO/IEC 9126 is not usable as a bottom to top approach to quality engineering, and even less usable as a top to bottom approach.

In order to address these concerns, an ISO committee began working on a revision of the standard. The results of this effort are the introduction of a revised version of ISO/IEC 9126 focusing on the quality model, and a new standard, ISO/IEC 14598 (ISO/IEC, 1999a) focusing on software product evaluation. ISO/IEC 14598 addresses Pfleeger's first concern while the revision to ISO/IEC 9126 aims to resolve the second and third issues. ISO/IEC 9126 is now a four part standard:

- ISO/IEC 9126-1 (ISO/IEC, 2001a) defines an updated quality model.
- ISO/IEC 9126-2 (ISO/IEC, 2003a) defines a set of external measures.
- ISO/IEC 9126-3 (ISO/IEC, 2003b) defines a set of internal measures.
- ISO/IEC 9126-4 (ISO/IEC, 2001b) defines a set of quality in use measures.

The new quality model defined in ISO/IEC 9126-1 recognizes three aspects of software quality and defines them as follows: (the full definition is given as it is pertinent to the discussion that ensues)

- Quality in Use:

> *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. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself. (ISO/IEC, 2001a)*

- External quality:

> *External quality is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. During testing, most faults*

> *should be discovered and eliminated. However, some faults may still remain after testing. As it is difficult to correct the software architecture or other fundamental design aspects of the software, the fundamental design remains unchanged throughout the testing. (ISO/IEC, 2001a)*

- Internal Quality:

> *Internal quality is the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the Internal Quality requirements. Details of software product quality can be improved during code implementation, reviewing and testing, but the fundamental nature of the software product quality represented by the Internal Quality remains unchanged unless redesigned. (ISO/IEC, 2001a)*

The Internal and External Quality model is inspired from McCall and Boehm's work. It is a three layer model composed of quality characteristics, quality subcharacteristics and quality measures. Figure 10 illustrates this model and Tables IX to XV give the definition of the characteristics and subcharacteristics. More than 100 measures of Internal and External Quality are provided as part of the standard. It is important to note that these are informational[6], meaning that other measures can also be used.

[6] It is important to explain what the adjective "informational" means in ISO-speak. An informational part is something against which conformance is not measured. In the case of ISO/IEC 9126, the measures form an informational part of the standard. This means that using these measures is a good step towards compliance. However, the authors of the standard recognize that there is no universal set of measures. Therefore, the standard allows for other measures to be defined in order to replace and/or complement the given measures. Informational could be thought of as "proposed".

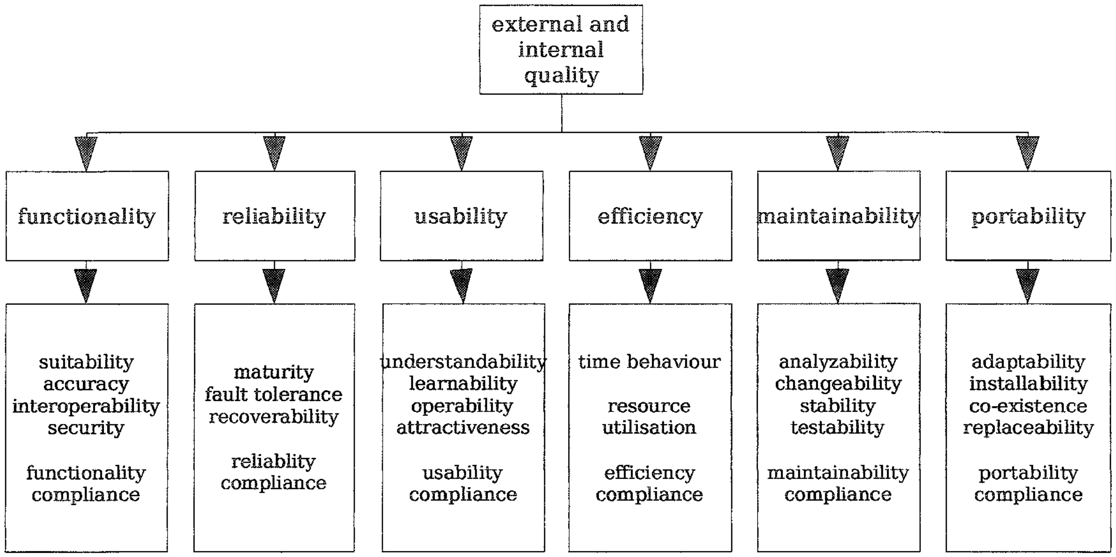


Figure 10 3-layer model for internal and External Quality
Adapted from (ISO/IEC, 2001a)

Table IX

Definition of Quality Characteristics
Adapted from (ISO/IEC, 2001a)

Quality Characteristic	***Definition***
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
Functionality	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
Reliability	The capability of the software product to maintain a specified level of performance when used under specified conditions
Usability	The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
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.
Portability	The capability of the software product to be transferred from one environment to another.

Table X

Definition of Efficiency Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Efficiency Subcharacteristics	***Definition***
Time Behavior	The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
Resource Utilization	The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.
Compliance	The capability of the software product to adhere to standards or conventions relating to efficiency.

Table XI

Definition of Functionality Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Functionality Subcharacteristics	***Definition***
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.
Interoperability	The capability of the software product to interact with one or more specified systems.
Security	The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.
Compliance	The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality.

Table XII

Definition of Reliability Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Reliability Subcharacteristics	***Definition***
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.
Compliance	The capability of the software product to adhere to standards, conventions or regulations relating to reliability.

Table XIII

Definition of Usability Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Usability Subcharacteristics	***Definition***
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.
Compliance	The capability of the software product to adhere to standards, conventions, style guides or regulations relating to usability.

Table XIV

Definition of Maintainability Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Maintainability Subcharacteristics	***Definition***
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.
Compliance	The capability of the software product to adhere to standards or conventions relating to maintainability.

Table XV

Definition of Portability Subcharacteristics
Adapted from (ISO/IEC, 2001a)

Portability Subcharacteristics	***Definition***
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.
Compliance	The capability of the software product to adhere to standards or conventions relating to portability.

Finally, Quality in Use is modeled in a different way than Internal and External Quality. Figure 11 illustrates the two layer Quality in Use model composed of characteristics and quality measures. Table XV provides the definition of the characteristics.

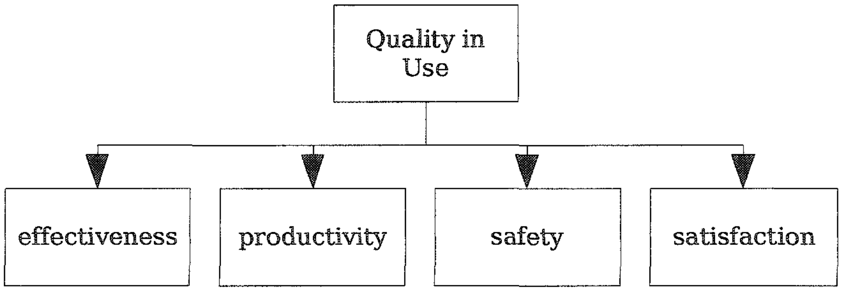


Figure 11 Quality in Use model
Adapted from (ISO/IEC, 2001a)

Table XVI

Definition of Quality in Use Characteristics
Adapted from (ISO/IEC, 2001a)

Quality in Use Characteristics	***Definition***
Effectiveness	The capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use.
Productivity	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	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	The capability of the software product to satisfy users in a specified context of use.

Theoretically, Internal Quality, External Quality and Quality in Use are linked together with a predictive model[7]. This is illustrated in Figure 12.

[7] Note that this discussion is about the concepts of Internal Quality, External Quality and Quality and Use. The implementation of these concepts in ISO/IEC 9126 will be discussed below.

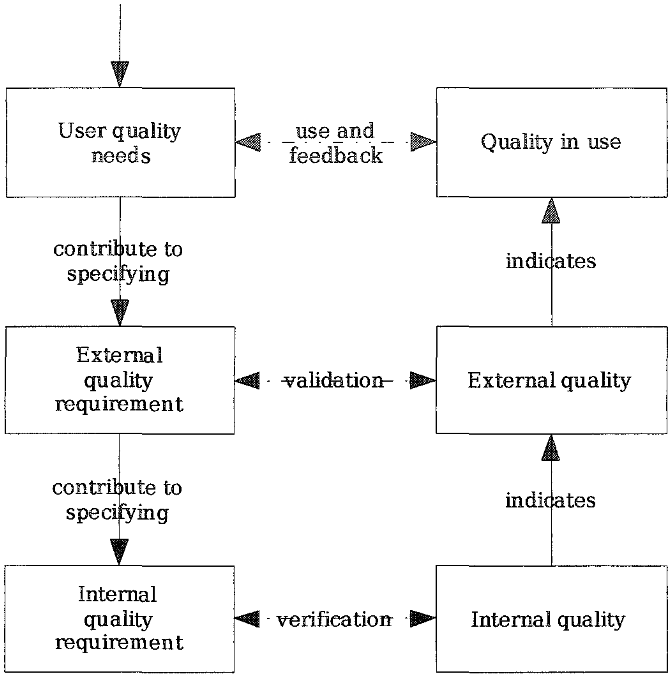


Figure 12 Relationships between the different aspects of quality
Adapted from (ISO/IEC, 2001a)

This prediction relationship states that user quality needs should first be established and specified using the Quality In Use model. From these requirements as well as other sources, External Quality requirements should be established using the External Quality model. Finally, the Internal Quality requirements should be constructed from the External Quality requirements and other sources. Once the requirements are established and software construction is under way, the quality model can be used to predict the overall quality. For example, measurement of Internal Quality can be useful in predicting External Quality. Likewise, measurement of External Quality can be useful in predicting Quality in Use.

The above paragraphs describe the ideal theoretical model that links these three aspects of quality. However, in reality, no model may claim to follow perfectly this predictive model. Although the ISO/IEC 9126 model follows this approach closely, no claims are made as to the real predictive power of the model. While the links between

Internal and External Quality seem rather obvious because the models are essentially the same, caution must be exercised. While the name of the characteristics and subcharacteristics are the same, the links between Internal and External Quality must be verified empirically. The same reasoning applies to the links between External Quality and Quality in Use.

2.4.2 Discussion and evaluation

The new version of ISO/IEC 9126 is gaining momentum in the industry. Some corporate quality models, for example MITRE's SQAE (Martin & Shaffer, 1996), are beginning a migration from a model based on McCall's and Boehm's research to one based on ISO/IEC 9126 (Côté, Suryn, Martin & Laporte, 2004a; Côté, Suryn, Martin & Laporte, 2004b; Côté, Suryn, Laporte & Martin, 2005). This new version of ISO/IEC 9126 is thus seen as an improvement upon the older quality models.

It is interesting to see how the three aspects of quality defined above can be directly linked to the perspectives of quality that were outlined in section 2. More specifically:

- ISO/IEC 9126-4, which defines Quality in Use, is directly related to the user and value-based perspectives. The definition of the user perspective of quality states that it is concerned with the appropriateness of a product for a given context of use. Quality in Use is defined as the capability of the software product to enable specified users to achieve specified goals in specified contexts of use. The relationship between the two is clear. Quality in Use and the value based perspective of quality are linked essentially through the Satisfaction characteristic. This characteristic inherently recognizes that quality can have a different meaning and/or value for different stakeholders. Satisfaction levels can thus be set according to those levels of perception.

- ISO/IEC 9126-3, which defines Internal Quality, and ISO/IEC 9126-2, which defines External Quality, are directly related to both the manufacturing and product perspectives. The definitions of the quality characteristics Functionality and Reliability can be linked with the manufacturing perspective of quality. Reliability, Usability, Efficiency, Maintainability and Portability are all inherent characteristics of the product and a manifestation of the product perspective of quality.

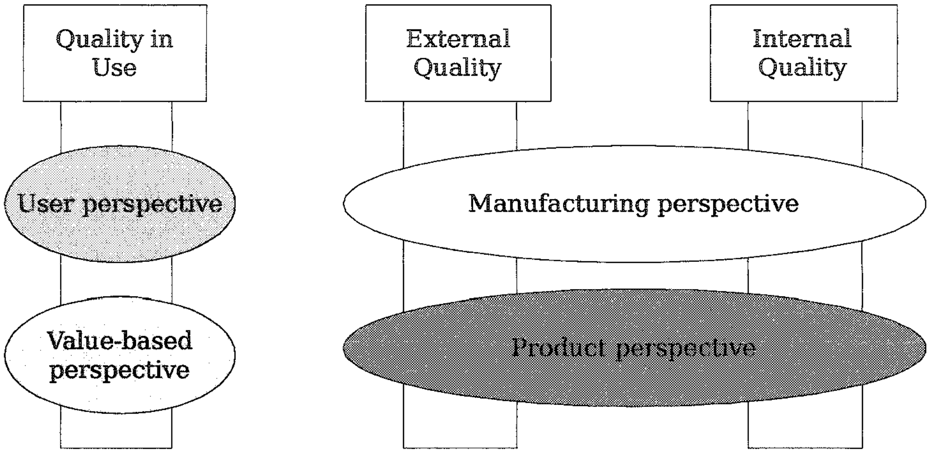


Figure 13 Relationships between ISO/IEC 9126 and the perspectives of quality

From the review of the different quality models, one might point out that none seem to address the transcendental perspective of quality. One might even ask the following pertinent question: Does ISO/IEC 9126 address the transcendental perspective of quality? Recall that the transcendental perspective of quality relates to quality as something that is recognized but not defined. At this point, the following hypothesis will be made:

As the transcendental perspective of quality cannot be defined, it cannot be explicitly implemented in a software product. However, the transcendental aspect of quality will emerge when a holistic approach to quality engineering is taken.

This model seems to recognize all the perspectives of quality as important contributors to the overall assessment of quality. It takes an incremental approach to software quality that begins with Quality in Use, something that is easy to grasp for non-technical stakeholders, and ends with Internal Quality, something more technically inclined stakeholders will feel more comfortable with. Furthermore, there is a comprehensive set of suggested measures that allow for the assessment of software quality.

2.4.3 Conclusion

Table XVII

Evaluation of ISO/IEC 9126

Model selection question	***Answer***
Can the framework be used by stakeholders to set quality factors early in a system's lifecycle?	YES. ISO/IEC 9126 implicitly suggests that requirements engineering should begin by specifying quality in use needs. As these needs are rather high level, at least compared to external and Internal Quality needs, they can be specified early in the lifecycle. This is supported by Figure 12.
Can the established quality requirements based on the model be effectively communicated to the technical personnel?	YES. In theory, by following a quality specification process based on the one illustrated in Figure 12, high level needs can be decomposed into more specific external and Internal Quality needs that can be understood by technical personnel.
Is it possible to identify measures related to the establishment of quality factors and quality subfactors?	YES. ISO/IEC 9126 contains more than 100 measures. This seems to indicate a positive answer to this question.

At first glance, this model seems to be the only one to fully satisfy the requirements that were previously established for a model to be suitable for a top to bottom approach to quality engineering. This conclusion is based on an assessment of what is stated in ISO/IEC 9126-1. Before selecting this model as a foundation of quality engineering, a more thorough analysis is needed.

CHAPTER 3

ISO/IEC 9126 AS A FOUNDATION FOR QUALITY ENGINEERING

The previous analysis of the text of ISO/IEC 9126 showed that this standard is a promising foundation for Quality Engineering. According to the text of the standard, it can indeed be used to specify quality requirements early in the lifecycle as well as be useful throughout the rest of the lifecycle. Before asserting that this standard is a solid foundation for Quality Engineering, the claims made in the text must be verified. This chapter presents the methodology and the results of such a verification.

Before proceeding with the analysis, its scope must be defined. ISO/IEC 9126 is a complex multi-part standard. As it was argued before, most quality models implicitly and explicitly support a bottom to top approach because of their measure orientation. ISO/IEC 9126 is not different it this matter, since it offers more than 100 measures. What seems more important to evaluate is the ability of ISO/IEC 9126 to be useful from top to bottom (i.e. at the beginning of the software development life cycle). Figure 12 on page 40 clearly illustrates that the entry point in a top to bottom approach lies in a definition of the Quality in Use requirements. The second step in such an approach is to verify how the Quality in Use needs can influence the definition of External Quality requirements. The ability of ISO/IEC 9126 to be useful in these first two steps is crucial to its ability of being a solid foundation to Software Quality Engineering. Such an analysis already requires a considerable effort. Therefore, the study of External Quality, Internal Quality and the links between External Quality and Internal Quality are beyond the scope of this research.

3.1 Analysis methodology

ISO/IEC 9126-1 (ISO/IEC, 2001a) states that the quality model and its associated measures should fulfill the following requirements:

1. *The model must be usable in "defining quality requirements.*" (page iv, paragraph 3)
2. *The model must be "applicable to every kind of software."* (page 1, paragraph 3)

3. *The model must "provide consistent terminology."* (page 1, paragraph 3)
4. *The quality model must be usable for setting quality goals for software products and intermediate products."* (page 6, paragraph 8)
5. *The model should be "hierarchically decomposed into a quality model composed of characteristics and subcharacteristics."* (page 6, paragraph 8)
6. *The model must be predictive. This means that Internal Quality should be predictive of External Quality. Likewise, External Quality must be predictive of quality in use.* (page 3, figure 2 and page 4, figure 3)
7. *Conformance to the model shall be judged either by the usage of the characteristics and subcharacteristics or by a mapping to those characteristics and subcharacteristics.* (page 2, clause 2) The model must therefore be exhaustive enough to provide the user with a thorough selection or to provide an unambiguous mapping.

If the model and the associated measures fulfill these stringent requirements, this model would indeed be a suitable one on which to base a quality engineering methodology. Therefore, the following questions will be thoroughly answered to verify the implementation of these requirements in the standard:

1. Can the quality model and its measures be used to thoroughly set quality requirements at the beginning of the lifecycle? (from points 1 and 4)
2. Are the quality model and its measures exhaustive and hierarchical? (from points 1, 2, 4, 5 and 7)
3. Can the External Quality model be used as a prediction of the actual Quality in Use? In other words, is there an unambiguous mapping between the External and Quality in Use models? (from point 6)

An answer to the questions presented above lies in a thorough analysis of the measures associated with the Quality in Use model. The results of this analysis will be presented first.

3.2 Analysis of the measures of Quality in Use

This analysis of the measures will be used to answer the three questions that were asked in the previous section. Therefore, they must generate enough data for the answers to be credible.

The following four angles are used to answer the first and third question:

- The relative *impact* of the measure will first be analyzed. The IEEE (1998) defines the impact as an "*indication of whether a metric can be used to alter or halt the project*". The impact can also be analyzed by asking the following question: "Can the measure be used to indicate deficient software quality?" A measure that has low or average impact will not be a useful Quality in Use measure.

- The second angle to be analyzed will be the approximate cost of applying the measure and using it as a requirement. A measure that has a prohibitive cost will not be widely usable as a foundation for quality engineering.

- For each measure, the following question will be answered: "Can this measure be used to thoroughly set quality goals and requirements?" If a measure cannot be used to thoroughly set quality requirements, then it cannot be used for software quality engineering as defined in this thesis.

- Finally, the fitness of the measure in the predictability model proposed by ISO/IEC 9126-1 (see Figure 12) will be assessed on a measure by measure basis by answering the following questions: "Which, if any, External Quality characteristics and subcharacteristics ***may*** predict the value of this measure?" It is important to note that the goal of this analysis is not to prove that links *do* exist between External Quality and Quality in Use, but rather that such links *may* exist under certain conditions. If a measure does not fit into the predictive model, then it will be difficult to define External Quality requirements from the Quality in Use model, making software quality engineering more difficult.

The links between the questions and the angles will now be explained.

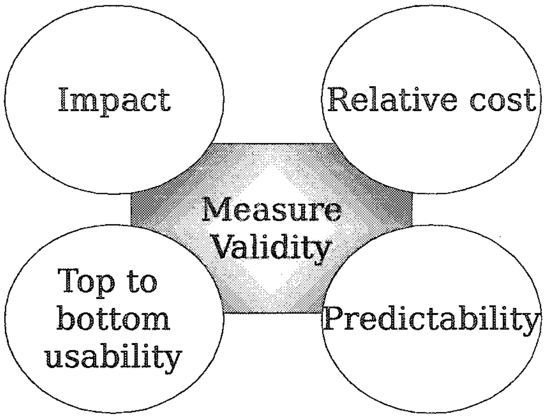


Figure 14 Analysis angles

The first question deals with the *usefulness* of the measure at the *beginning of the life cycle*. In order for a measure to be *useful*, it needs to have sufficient impact and have an acceptable cost. This is dealt with the first and second angles. In order for a measure to be useful at the *beginning of the life cycle* in needs to be useful in setting quality goals and requirements, which is one of the first activities performed in the software life cycle. The third angle is concerned with answering this part of the question.

The third question, which deals with predictability, is unambiguously answered by the data generated from analyzing the fourth angle.

The second question, which deals with completeness, can not be answered by looking only at the measures. This question will be answered by comparing the model to other quality models.

The suitability of each measure will be rated qualitatively according to the following scale:

- *N* : The measure is considered to be non-applicable. Such a grade is given when the measure is clearly lacking with respect to at least one of the analysis angle described above.

- *C* : The applicability of the measure is conditional. Such a grade is given when the measure is conceptually applicable but could still be improved with respect to a number of angles.
- *A*: The measure is considered to be applicable. Such a grade is given when there are no obstacles to using this measure.

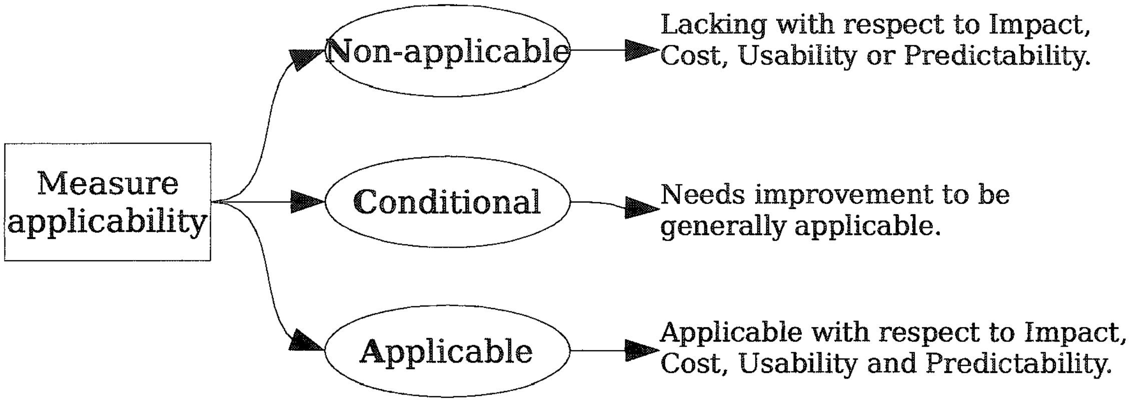

Measure applicability
Non-applicable
Lacking with respect to Impact, Cost, Usability or Predictability.
Conditional
Needs improvement to be generally applicable.
Applicable
Applicable with respect to Impact, Cost, Usability and Predictability.

Figure 15 Possible measure ratings

Because the results of the analysis of the measures are quite voluminous, they are presented as an appendix to this document (Appendix 1 – Analysis of ISO/IEC 9126-4 Quality In Use Measures).

The results of the analysis are presented in summary form in Table XVIII.

Table XVIII

ISO/IEC 9126-4's suitability for Quality Engineering

	Impact	Cost	Usability	Predictability	Conclusion
Effectiveness					
Task Effectiveness	C	A	N	A	**N**
Task Completion	A	A	C/N[8]	A	**C/N[8]**
Error Frequency	C	A	C	A	**C**
Productivity					
Task Time	A	A	C/A[9]	A	**N/A[9]**
Task Efficiency	A	A	N	A	**N**
Economic Productivity	C	A	N	A	**N**
Productive Proportion	A	C	A	A	**A**
Relative User Efficiency	C	A	A	A	**A**
Safety					
User Health and Safety	A	C	A	C	**C**
Safety of People...	A	C	A	A	**A**
Economic Damage	C	C	A	A	**C**
Software Damage	C	C	A	A	**C**
Satisfaction					
Satisfaction Scale	A	C	C	A	**C**
Satisfaction Questionnaire	A	A	A	A	**A**
Discretionary Usage	C	C	C	A	**C**

Legend:
N stands for Non-applicable
C stands for Conditional applicability
A stands for Applicable

Of the 15 measures that are proposed by ISO/IEC 9126-4, 3 or[10] 4 were found to be clearly non applicable. On the other end, between 4 and 6 measures are clearly applicable and fulfill the goals of the quality model expressed in ISO/IEC 9126-1 as

[8] This measure is conditionally applicable when the tasks are not composed of multiple goals.

[9] Would be applicable with minor modifications

[10] See footnotes 8 to 9 for an explanation.

evaluated by the criteria outline previously. The majority of the measures, between 5 and 8, have a conditional applicability.

Most of the time, the applicability is conditional because the measure has been judged unsuitable for expressing quality goals and requirements. While ISO/IEC 9126-4 is clearly the best hope as a foundation for software quality engineering, it is not sufficient.

For example, the measure *Task Effectiveness* was judged to be non applicable because it lacked usability for expressing quality goals and requirements. The purpose of this measure is to evaluate the proportion of the goals of the task that is achieved correctly. It is applied as a user test described by the following equation: $M_1 = |1 - \sum A_i|$, where each A_i is a proportional value of each missing or incorrect component in the task output. ISO/IEC 9126-4 provides the following clarifications concerning the application of this measure:

> *"Each potential missing or incomplete component is given a weight A_i based on the extent to which it detracts from the value of the output to the business or user. (If the sum of the weights exceeds 1, the metric is normally set to 0, although this may indicate negative outcomes and potential safety issues.) The scoring scheme is refined iteratively by applying it to a series of task outputs and adjusting the weights until the measures obtained are repeatable, reproducible and meaningful."*

This measure can clearly be used to indicate deficient software quality. However, it might be difficult to set a threshold for quality (for example: "- a value below *x* indicates low quality"). This measure can be applied to almost any kind of software, as all software must in the end accomplish a task, and most tasks can be decomposed into a set of goals. The analysis of this measure found that there are at least two reasons which complicate the task of using this measure for setting quality goals and requirements:

1. First, it is difficult to set a threshold separating sufficient quality from insufficient quality. The reason for this is that the standard specifies that the sum of the A_i must not necessarily equal 1. The consequence of this is that the measure is unbounded. Although the standard specifies that negative

results are normalized to 0, such results are the only *clear* indication of insufficient quality.

2. The second possible obstacle to the usability of this measure is the suggestion that the scoring scheme be refined iteratively. The standard specifies that the "*scoring scheme is refined iteratively by applying it to a series of task outputs and adjusting the weights until the measures obtained are repeatable, reproducible and meaningful*". By requiring that the measure use task outputs for adjusting the weights, it makes it difficult to use this measure before task outputs are available. In the phase of requirements definition, expectations for the measure would have to be based on an expert's judgment or statistical data (if available/applicable). In some cases, either could prove inaccurate. It is doubtful that software contractors would agree to having such a clause based on such a measure in a contract.

Because the usability of this measure relies on too many conditions, it is at the very least difficult to use this measure as a quality goal or a requirement.

The situation can be improved by reformulating the measure and explaining it properly. By trying to be too concise, the standard obfuscates the usefulness of this measure. The standard should first state that each task that a software product must accomplish should be decomposed into goals. The accomplishment of those goals, whether partial or complete, should result in the success of the task. Each goal (*G*) should be given a value representing the *approximate* percentage of the task (P_G) that is attained when the goal is accomplished. The sum of those percentages should be 100%. Some of the goals might be marked as "*essential*", meaning that failure to accomplish those goals will result in 0% task effectiveness. The task effectiveness could be measured by the following sum:

$$TE = \begin{matrix} \sum_{task} P_G, & When\,all\,essential\,goals\,are\,attained \\ 0\%, & Otherwise \end{matrix}$$

A threshold for acceptable quality can then be set on a task by task basis by determining which goals :

- are essential
- are desirable
- are "nice to have"

This classification can be made from many perspectives: user, business, economic, etc. The target task effectiveness is the sum of the percentages associated to the essential and desirable goals. The task effectiveness can then be analyzed for many users and meaningful conclusions can be drawn from its application.

The results of these proposed changes may be considered essential as:

- The impact of the measure is now very important. Each task that falls below the target task effectiveness has an unacceptable level of quality.
- The cost of the application remains negligible.
- The scoring scheme does not need to be refined iteratively anymore. The percentages associated to each goal are not even really important. They only help in quantifying the contribution of each goal.
- The usage of this measure will help stakeholders define a clear acceptance criterion on a task by task basis.

Implementation of these changes would radiate positively throughout the Quality in Use model because many other measures depend on this one. For example, it would help defining a clear acceptance for tasks that are composed of multiple goals and thus make the task completion measure generally applicable.

Appendix 2 proposes a set of enhancements similar to the one presented above that makes the proposed measures of ISO/IEC 9126 suitable for thoroughly setting quality goals and requirements at the beginning of the lifecycle. References to ISO/IEC 9126-4 measures beyond this point refer to these *enhanced* measures. Appendix 3 shows that it is possible to express these changes in a concise language and format similar to the ISO/IEC 9126 standard.

The three questions that led to this analysis can now be answered, based on this improved standard.

3.3 ISO/IEC 9126 and Requirements Engineering

The first question that was asked was:

> *Can the quality model and its measures be used to thoroughly set quality requirements at the beginning of the lifecycle?*

Figure 12 illustrated that it is indeed the goal of the quality model proposed by ISO/IEC 9126-1 to be useful early in the software engineering lifecycle as a mean for determining quality requirements. However, this quality prediction framework relies heavily on the Quality in Use model as the entry point by which user quality needs can be specified. It has been shown that while the model itself is suitable for software quality engineering, the measures associated to the model focus on *a posteriori* use (i.e. they are biased toward usage at the end of the lifecycle and of little use for specifying quality requirements). For example, the measures *Task Effectiveness*, *Task Completion*, *Error Frequency*, *Task Time*, *Task Efficiency*, *Economic Productivity*, *Satisfaction Scale* and *Discretionary Usage* were found to be non-applicable or of conditional applicability with respect to usability as a goal or requirement. This represents 8 out of 15 measures.

The enhancements proposed in Appendix 2 and 3 aim to correct this situation.

The ability ISO/IEC 9126-4 to express quality goals and define quality requirements at the beginning of the lifecycle can be improved if the enhancements proposed in Appendix 2 and 3 are implemented.

3.4 ISO/IEC 9126 and the exhaustiveness criterion

The second question that was asked was:

> *Are the quality model and its measures exhaustive and hierarchical?*

First of all, there is no doubt that the model proposed in ISO/IEC 9126-1 is hierarchical. Figures 10 and 11 clearly illustrate that fact. Recall that the Internal and External Quality model is composed of an orthogonal[11] three-layer hierarchy and the Quality in Use is composed of an orthogonal two-layer hierarchy.

As for its exhaustiveness, the Internal and External Quality model proposes 27 subcharacteristics that spawn over 6 characteristics. It is difficult to judge the exhaustiveness based on this data alone since new aspects of Internal and External Quality could be discovered. However, the model seems to cover most of the aspects encountered in the study of other quality models. With respect to exhaustiveness, it is important to note that more than 100 measures are associated to the subcharacteristics. However, preliminary results from another ongoing research at l'*École de Technologie Supérieure* (Berrazouane, 2004) indicates that there are some concerns as to the validity of these measures. One of the main reported concern is that some of the measures are outdated and inapplicable to current software development techniques.

On the other hand, the Quality in Use model is much more concise. While it satisfies the hierarchical decomposition criteria, the exhaustiveness requirement is more difficult to evaluate because the model is composed of four characteristics and 15 measures. If it is taken into account that the set of measures associated to the Quality in Use model is sufficient to help uncover requirements about almost every External Quality characteristics, then it is possible to answer this question affirmatively. This will be demonstrated in the following section. However, it is interesting to note that another ongoing research at Concordia's Human Centered Software Engineering Group (Seffah, Kececi & Donyaee, 2001) has identified other components of usability and Quality in Use that are not addressed by ISO/IEC 9126-4, namely characteristics such

[11] Orthogonality in this case means that there is a one to one relationship between all the layers (i.e. Each measure is associated to only one sub-characteristic, which is in turn associated to only one characteristic).

as *Efficiency*, *Internationability* and *Accessibility*. In this respect, the exhaustiveness of the Quality in Use model could be improved.

3.5 ISO/IEC 9126 as a predictive model

The third and final question was:

> *Can the External Quality model be used as a prediction of the actual Quality in Use? In other words, is there an unambiguous mapping between the External and Quality in Use models?*

The following figure, taken from ISO/IEC 9126-1 (ISO/IEC 2001a), confirms that it is indeed a goal of the model to be *useful* in a predictive manner (as opposed to being *used* in a predictive manner).

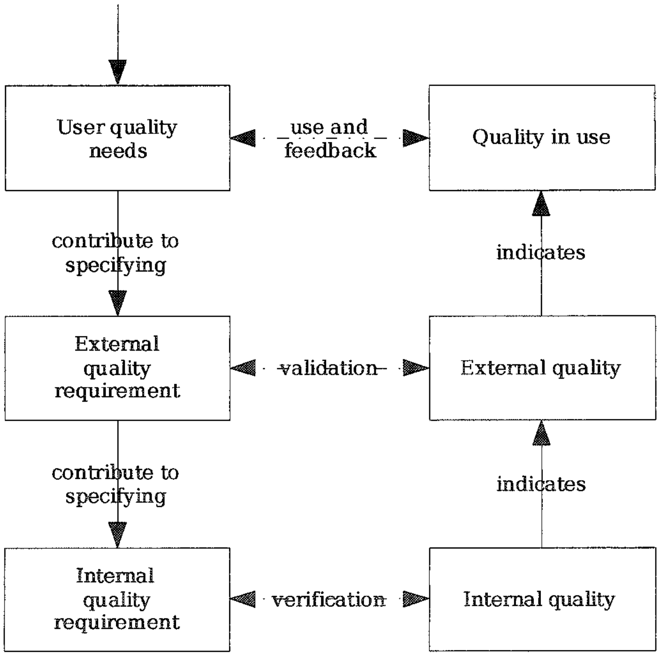


Figure 16 Predictive nature of quality
Quality in Use may be used to specify parts of External Quality.
On the other hand, an evaluation of External Quality should be indicative of Quality in Use.

However, neither ISO/IEC 9126-1, ISO/IEC 9126-2 nor ISO/IEC 9126-4 specify the links between Quality in Use and External Quality. The analysis of the ISO/IEC 9126-4 measures presented in Appendix 1 allows links to be drawn between the two models. A summary is presented below.

Table XIX

Links from Quality in Use to External Quality

Quality in Use External Quality	Task Effectiveness	Task Completion	Effort Frequency	Task Time	Task Efficiency	Economic Productivity	Productive Proportion	Relative User Efficiency	User Health and Safety	Safety of People ...	Economic Damage	Satisfaction Scale	Satisfaction Questionnaire
Functionality													
Suitability	X[12]	X			X	X							
Accuracy	X	X			X	X							
Interoperability													
Security													
Functionality Compliance													
Reliability													
Maturity											X		
Fault Tolerance											X		
Recoverability											X		
Reliability Compliance													
Usability													
Understandability	X	X	X	X	X	X	X	X	X	X	X	X	X
Learnability								X				X	X
Operability	X	X	X	X	X	X	X	X	X	X	X	X	X
Attractiveness												X	X
Usability Compliance													
Efficiency													
Time Behavior				X	X	X	X						X
Resource Utilization						X	X						X
Efficiency Compliance													

[12] An X means that there is a potential link. The strength of the link could be verified empirically and vary depending on the context of use.

Table XIX (continued)

Quality in Use External Quality	Task Effectiveness	Task Completion	Effort Frequency	Task Time	Task Efficiency	Economic Productivity	Productive Proportion	Relative User Efficiency	User Health and Safety	Safety of People ...	Economic Damage	Satisfaction Scale	Satisfaction Questionnaire
Maintainability													
Analyzability										X	X		
Changeability										X	X		
Stability										X	X		
Testability										X	X		
Maintainability Compliance													
Portability													
Adaptability													X
Installability													X
Co-existence													X
Replaceability													X
Portability Compliance													

Note: Almost all External Quality characteristics can be related to the Satisfaction Scale and the Satisfaction Questionnaire, depending on their contents. Only the most important relationships are shown.

From the results presented in the table, it is possible to observe that with the exception of all the *Compliance* subcharacteristics, only the *Interoperability* and *Security* subcharacteristics are not clearly associated with Quality in Use (other than loosely with the *Satisfaction Scale* or *Satisfaction Questionnaire* characteristics).

Figure 16 highlights the duality of the relationship between External Quality and Quality in Use. The first part of this relationship is the influence of Quality in Use requirements in uncovering External Quality requirements. The second part of the relationship is the ability of External Quality to be predictive of Quality in Use. These two aspects are intrinsically linked. There is no doubt that if meaningful Quality in Use requirements are uncovered, they will help uncover External Quality requirements. For example, one only has to look at the definitions of Suitability (Table XI), Accuracy (Table XI), Understandability (Table XIII) and Operability (Table XIII) to conclude that they are a prerequisite of proper Task Effectiveness. Evaluation of Suitability, Accuracy Understandability and Operability requirements derived from the Task Effectiveness requirement should therefore result in a *predicted* Quality in Use. Whether this prediction reflects reality will be dependent on how many other factors influence the required Task Effectiveness (i.e. Task Effectiveness is not only dependent on External Quality). Meeting the External Quality requirements is a *necessary but not sufficient* condition for attaining the Quality in Use requirements. Therefore, External Quality is predictive of Quality in Use in the sense that if the External Quality requirements are not met, attaining the necessary Quality in Use should not be possible.

In conclusion, there is no doubt that there exists a relationship between External Quality and Quality in Use. It has been shown that a possible manifestion of this relationship could be in links between the measures of Quality in Use and the subcharacteristics of External Quality. Unfortunately, this relationship is not unambiguous (i.e. it is not explicitly specified in the standard).

3.6 Conclusion

This deeper analysis of the model has shown that ISO 9126 **model** is indeed applicable with respect to the criteria outlined in the IEEE Standard for a Software Quality Metrics Methodology for a top to bottom approach to quality engineering. However there are serious concerns with the **measures** associated to the model with respect to usability for specifying quality goals and requirements. Although the situation can be improved if the modifications outlined in Appendix 2 and 3 are made, this will be further discussed in the following section.

DISCUSSION

Analysis of the methodology

The primary objective of this research was to identify a quality model that can serve as a basis for the improvement of software quality in a continuous, systematic, disciplined and quantifiable way. In order to accomplish this objective, a pyramid like approach has been followed(Figure 17).

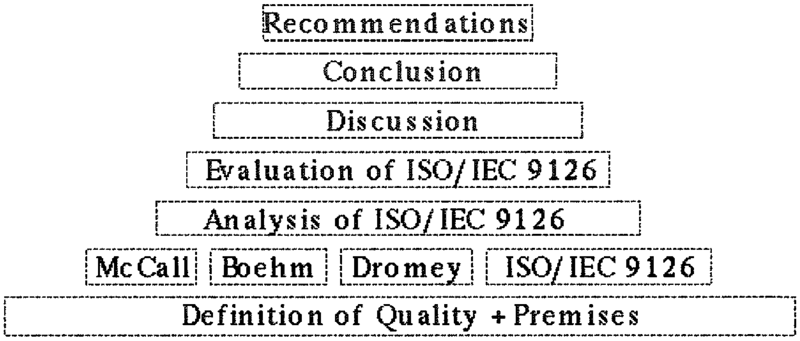


Figure 17 Pyramid-like approach

The rationale for following such an approach was to build a solid base for the recommendations. Each *storey* of this pyramid will now be reviewed.

- The purpose of the literature review was to define the notion of quality in software engineering as well as establish premises on which the rest of the thesis would rely.

  A proper definition of quality is essential for analyzing and selecting a quality model, as it will provide a foundation for comparison and evaluation. Instead of relying on a narrow and traditional definition such as “Quality is conformance to requirements”, it was elected to use a broad definition based on the teachings of David Garvin. This definition sees quality as a combination of 5 perspectives: transcendental, user, manufacturing, product and value-based. This broad definition includes the narrower definition. The advantage of using such a broad definition is that it allows for a wide-reaching evaluation.

The premises which sustain this thesis can be seen as ramifications of this definition. It was first stated that a possible part of the solution to improving software quality in general was to establish quality requirements. This is a sensible suggestion, as it is impossible to improve something which is not defined. The second premiss stated that a possible reason that quality requirements are not established today is because no quality model has been identified as suitable for this purpose. The third and fourth premises establish requirements that such a model should fulfill. Namely, a quality model suitable for software quality engineering should be congruous with the definition of quality and support both the evaluation and specification of quality requirements. Congruence is necessary for the model to be broadly applicable. Support for both evaluation and specification is primordial for the activity to be considered as an engineering discipline. These last two premises are sensible requirements for a model to fulfill and form a solid foundation to the search for a suitable quality model.

- The second step in this approach was to identify the quality models recognized by the industry and select the one that was the most promising foundation to Software Quality Engineering for further analysis. The literature review established that an essential characteristic for a model to be suitable for Software Quality Engineering is to be usable both in a *bottom to top* and a *top to bottom* approach. This characteristic is essential for the model to be useful in both specification and evaluation of software quality. It was found in this preliminary analysis that the *text* of the ISO/IEC 9126 standard meets expectations with respect to the requirements for a model to be used as a foundation for Software Quality Engineering. ISO/IEC 9126 was the only model to meet these requirements.

- The third step was to analyze the measures of ISO/IEC 9126-4 more in depth in order to produce data for evaluating this standard's suitability for Software Quality Engineering. The reason for focusing on the Quality in Use model (ISO/IEC 9126-4) is because of it's importance in the top to bottom approach. Indeed, the Quality in Use model is the entry point for specification of quality needs and evaluation of Quality in Use informs the client if those requirements were met.

In order to generate data for the evaluation of the standard, four analysis angles were chosen. These angles are inspired from the IEEE standard on software quality metrics.

The first angle was to analyze the impact of the measure. The impact of a measure relates to it's ability to discriminate good quality from bad quality.

The second angle was to analyze the relative cost of the measure. While such a measure might have great scientific value, it will not be widely applicable as a foundation for quality engineering.

The third angle was to analyze if a meaningful requirement could be set from the measure. As has been argued before, this is crucial for the model to be useful as a foundation for Software Quality Engineering.

The fourth angle was to analyze the possibility of a relationship between the Quality in Use measures and External Quality. A relationship between a Quality in Use measure and External Quality characteristics and subcharacteristics is necessary for the requirements to be further decomposed into implementable elements. The goal was not to define the strength of the relationship between External Quality and Quality in Use, but rather to indicate where and why such links could exist. Defining the strength of links is beyond the scope of this research.

While not exhaustive, these four angles form a sufficient set to reach a conclusion on the applicability of ISO/IEC 9126 as a foundation for Software Quality Engineering.

- The fourth and final step leading to a conclusion on the suitability of ISO/IEC 9126 as a foundation for Software Quality Engineering was to answer three questions that were deemed a sufficient requirement for the model to be suitable using the data produced during the analysis. The first question addressed the ability of the model to be usable in setting quality requirements at the beginning of the lifecycle. The second question addressed the exhaustiveness of the model. The third and final question addressed the predictability of the model.

This approach has led to the identification ISO/IEC 9126 as the best potential foundation for Software Quality Engineering according to the selection criteria. This result will be further analyzed in the following section.

Analysis of the results

The evaluation of ISO/IEC 9126 found that there is no question as to whether the intent of the standard (i.e. its *text*) is a suitable foundation for Software Quality Engineering. Indeed, section 3.1 of this thesis quoted several places where the standard indicates that it would be a suitable foundation.

Unfortunately, verification of the implementation of this intent (i.e. the measures) indicates weaknesses that require improvements in order for the standard to be a foundation for Software Quality Engineering as defined in this thesis. These needed improvements are threefold:

- Of the 15 measures proposed by the standard, 11 fail to be clearly applicable with respect to the selection criteria. Particularly, a majority of the measures fail to be clearly usable for setting meaningful quality requirements. As it has been discussed before, it is critical for measures to be useful in this respect in order for the standard to be considered a solid foundation for Software Quality Engineering.

- The exhaustiveness of the standard for External and Internal Quality has been challenged in other studies. As for the Quality in Use model, other models on this specific subject have introduced other characteristics that are not covered by ISO/IEC 9126-4. Exhaustiveness is not a problem per se if the model can easily be improved. However, ISO/IEC 9126 being an international standard must go through a lengthy modification process.

- As was discussed previously, it is impossible to universally prove the existence of links between Quality in Use and External Quality. Empirical verification is necessary for different contexts of use. One of the failings of the standard is that it does not specify where such links could exist, and how their existence could be establised for different contexts of use.

The question that must be asked is: Are these three points enough to declare that ISO/IEC 9126 is an unsuitable foundation for Quality Engineering?

The answer to this question is negative. Regardless of how many measures fail to meet expectations, they form an *informative*[13] part of the standard. In other words, their usage is not mandatory. The most important part is therefore the normative part of the standard. This part has been found to be suitable framework for Software Quality Engineering.

However, this does not mean that the set of measures is not important. They should be seen as an important supportive element. In their present state, this set is clearly lacking in its usability and its exhaustiveness. This thesis presents possible improvements to the ISO/IEC 9126-4 standard that aim to correct this situation.

As for the existence of links between External Quality and Quality in Use, there is no doubt that they exist. ISO/IEC 9126 is a first and necessary step towards a truly usable predictive framework. Subsequent versions of the standard may, and should, reinforce this embryonic support for a predictive quality framework.

[13] Please refer to the footnote on page 33 for a discussion on the meaning of *informational*.

CONCLUSION

This thesis has followed a path that leads to the identification of ISO/IEC 9126 as a model that is a suitable foundation for Software Quality Engineering. Although Software Quality Engineering is an emerging discipline, it is important because it recognizes the primordial significance of *quality* in Software Engineering and defines a systematic approach to achieve quality. Identification of a model suitable for this purpose is essential because such a model will be at the heart of a Software Quality Engineering methodology.

It was found through the analysis that ISO/IEC 9126's framework is clearly supportive of the idea of Software Quality Engineering:

- It recognizes the importance of both specifying and evaluating quality needs.
- It defines a predictive model that supports the top to bottom **and** the bottom to top approach to software quality.
- It wants to be applicable to every kind of software.

The selected model is however far from perfect. The main grievance with the model is that the promises of the framework fail to materialize themselves in the software quality measures that support it. These weaknesses are particularly visible when looking at the failure of the majority of the measures to be clearly useful in specifying meaningful requirements. These failures have been thoroughly documented in Appendix 1. Improvements that aim to correct this situation have been detailed in Appendix 2.

This thesis has raised interesting and important questions that could be the subject of further research. First among these in the author's view is the need for a better elucidation of the links between External Quality and Quality in Use. While this thesis as hinted to links that could exist between these two aspects of quality, a more formal specification of the links for different contexts of use could lead to better CASE tools that thoroughly assist stakeholders uncover External Quality requirements from Quality in Use needs. In turn, stakeholders would be assisted in uncovering Internal Quality requirements from their External Quality needs. Such research could lead to improvements that would reinforce ISO/IEC 9126's position as a solid foundation for Software Quality Engineering. Furthermore and more importantly, in today's world of complex software projects, such tools could prove crucial to improving software quality by helping stakeholders provide software that has the Quality in Use required by the users of the system and the External and Internal Quality characteristics necessary to provide this quality in specified contexts of use.

RECOMMENDATIONS

Three recommendations emanate from this research.

- The first recommendation is to overhaul the measures of ISO/IEC 9126-4. This thesis can serve as a guide to improving the measures:
    - The reasons explaining why improvements are necessary are detailed in Appendix 1.
    - Detailed improvements to the measures are presented in Appendix 2.
    - Appendix 3 presents the measures in the tabular format adopted by ISO/IEC 9126.
- The second recommendation is to compare each part of the ISO/IEC 9126 standard (Internal Quality, External Quality and Quality in Use) with quality models specific to these aspects. For example, there exists in the literature models that are specific to Quality in Use. These models usually express new ideas. Comparison with such models could help uncover areas not covered by ISO/IEC 9126, therefore improving its exhaustiveness and provide some new links between the different aspects of quality.
- The third and final recommendation is to better elucidate the links between the different aspects of quality. While this thesis has given hints as to where such links may exist, it is necessary for a wider applicability of the predictive framework that this existence be confirmed through empirical research.

APPENDIX 1

ANALYSIS OF ISO/IEC 9126-4 QUALITY IN USE MEASURES

Task Effectiveness

Description

Measure Name	Task Effectiveness
Purpose	Measure the proportion of the goals of the task that is achieved correctly.
Application	The measure is applied as a user test: $M1 = \lvert 1 - \sum A_i \rvert$ Where each A_i is a proportional value of each missing or incorrect component in the task output. ISO 9126-4 further specifies: Each potential missing or incomplete component is given a weight A_i based on the extent to which it detracts from the value of the output to the business or user. (If the sum of the weights exceeds 1, the metric is normally set to 0, although this may indicate negative outcomes and potential safety issues.) The scoring scheme is refined iteratively by applying it to a series of task outputs and adjusting the weights until the measures obtained are repeatable, reproducible and meaningful.

Analysis

Impact	This measure can clearly be used to indicate deficient software quality. However, it might be difficult to set a threshold for quality (for example, below *x* indicates low quality). This measure can be applied to almost any kind of software, as all software must accomplish a task, and most tasks are composed of different goals.
Cost of application	The cost of data item collection necessary for the application of this measure is negligible, as the collection can be integrated to the test phase.

Can this measure be used to thoroughly set quality goals and requirements?	At least two elements make it difficult to use this measure as a quality goal or requirement: ◦ First of all, it is difficult to set a threshold separating sufficient quality from insufficient quality. The reason for this is that the standard specifies that the sum of the Ai must not necessarily equal 1. The consequence of this is that the measure is unbounded. Although the standard specifies that negative results are normalized to 0, such results are the only clear indication of insufficient quality. ◦ The second possible obstacle to the usability of this measure is the suggestion that the scoring scheme be refined iteratively. The standard specifies that the "scoring scheme is refined iteratively by applying it to a series of task outputs and adjusting the weights until the measures obtained are repeatable, reproducible and meaningful". By requiring that the measure use task outputs for adjusting the weights, it makes it difficult to use this measure before task outputs are available. If used as requirement, it would have to be based on an expert judgment or statistical data (if available/applicable). In some cases, these might prove to be inaccurate. It is doubtful that software contractors would agree to having a clause based on such a measure in a contract.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	In order to complete the goals of a task, the proper **Functionality** must be present. There is therefore a strong link to **suitability** measures and to a lesser extent **accuracy**. **Usability** is also critical. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the effectiveness.

Conclusion	
Discussion	Although the cost of this measure seems negligible and it fits into the predictive model, it does not meet the requirement that it should be clearly usable in defining quality goals.
Rating	This measure is therefore **non applicable**.

Task Completion

Description	
Measure Name	Task Completion
Purpose	Measure the proportion of the tasks that are completed.
Application	The measure is applied as a user test: $X = A / B$ Where: • A is the number of tasks completed • B is the total number of tasks attempted ISO 9126-4 further notes: This metric can be measured for one user or a group of users. If tasks can be partially completed the Task effectiveness metric should be used.

Analysis	
Impact	This measure can clearly be used to indicate deficient software quality. A low ratio implies that the users are unable to complete the tasks that the software was built to assist. This measurement could also be applied to mock-ups and prototypes in order to guide the development team.
Cost of application	The cost of data item collection necessary for the application of this measure is negligible, as the collection can be integrated to the test phase. Such a test should usually be conducted as part of acceptance tests.

Can this measure be used to thoroughly set quality goals and requirements?	At first glance, it is doubtful that a meaningful requirement could be set using this measure. A requirement based on such a measure could read as follows: *"Task completion ratio shall be above 90%"*. This raises the following concerns: ◦ Is this a meaningful requirement? ◦ Wouldn't a high ratio of task completion be implicitly expected of most software? ◦ More importantly, are all tasks given the same weight? This last concern is of the utmost importance. This measure mixes and matches all the tasks together. It is doubtful that all the tasks that a system must accomplish are of the same importance. The standard should therefore state that tasks should be weighted or that task completion should be measured on a task by task basis. However, the inclusion of such a requirement in a software requirements specification can be seen as a safety net against incompetence on the part of the supplier. It may even force the supplier towards the good practice of close interaction with the end user. Lower expectations about this measure could also lower the initial cost of the software. It could be developed faster with little communication with the end users. It is important to point out that this is not considered a good practice, but it might be justifiable with respect to time and costs constraints. It is important to note that the measure offers no guidance on what constitutes a completed task other than referring to the task effectiveness measure when tasks can be partially completed. Because the task effectiveness measure has been shown to be non-applicable, it will influence negatively the applicability of this measure when such situations arise.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	In order to complete a task, the proper **Functionality** must be present. There is therefore a strong link to **suitability** measures and to a lesser extent **accuracy**. **Usability** is also critical. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the effectiveness.

Conclusion	
Discussion	The cost of applying this measure is negligible as it can be integrated to the test phase. To a certain extent, this measure can be used as quality goal and requirement in the simple case where there are few tasks. General applicability in this regard is conditional to reformulation and clarifications about the weighting problem. Finally, there are External Quality characteristics and subcharacteristics that stand a good chance of having a predictive value. This measure fulfills almost all the goals and objectives of ISO/IEC 9126 in the simple case where the tasks are simple and can either be accomplished or not. In the case where tasks are complex and can be partially accomplished, the task effectiveness measure must be used. Since that measure has been shown to be non-applicable, it influences negatively the rating of this measure. For more information, refer to the evaluation of the task effectiveness measure.
Rating	This measure is considered of **conditional** applicability in simple cases where task can either be accomplished or not. It is considered **non-applicable** in situations where tasks can be partially accomplished and must rely on the Task effectiveness measure. Improvements to the Task effectiveness measure would reflect positively on the applicability of this measure.

Error Frequency

Description	
Measure Name	Error Frequency
Purpose	Measure the frequency of errors.
Application	The measure is applied as a user test: $X = A / T$ Where: • A is the number of errors made by the user • T is the time or number of tasks ISO 9126-4 further notes: This metric is only appropriate for making comparisons if errors have equal importance, or are weighted.

Analysis	
Impact	This measure can clearly be used to indicate deficient software quality. A high value in *X* will indicate that users make a lot of errors while attempting to accomplish tasks. There are two possible interpretations of this measure, depending on the unit of *T*: • When *T* is the amount of time, the result is the number of errors per unit of time. • When *T* is the number of tasks, the result is the number of errors per task. Intuitively, it can be stated that the lower the result, the better the quality. However, it might be difficult to set a finite threshold to separate good quality from deficient quality.
Cost of application	The cost of data item collection necessary for the application of this measure is negligible, as the collection can be integrated to the test phase.

Can this measure be used to thoroughly set quality goals and requirements?	The primary concern with the usability of this measure as a quality goal or requirement is that it seems to encourage evaluating all the tasks at once instead of one by one. Although the standard specifies that errors should be weighted, it is believed that it is not sufficient to establish clear goals and requirements. While the standard acknowledges that errors do not have the same importance, *it fails to acknowledge that all tasks do not have the same importance.* This is of the upmost importance in the signification and interpretation of this measure and needs further discussion. While it might be important to know how many errors a user will make when performing a given task set, it is more important to know how many errors a user will make when performing a single given task. The standard could also clarify that tasks that are grouped together when evaluating the error frequency should be logically and functionally linked together. A secondary concern is the fact that this measure should be split into two seperate entities. This is because both the number of errors/unit of time and the number of errors/task are important.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** is critical to a low error frequency. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the effectiveness.

Conclusion

Discussion	Although the cost of applying this measure and its potential predictive value are satisfying, there are conditions to the usability of the Error Frequency measure in defining meaningful quality goals and requirements. Namely, clarifications
Rating	The applicability of this measure is therefore **conditional**.

Task Time

Description	
Measure Name	Task Time
Purpose	Measure the time needed to complete a task.
Application	The measure is applied as a user test: $X = Ta$ Where Ta is the task time.

Analysis	
Impact	This measure can clearly be used to indicate deficient software quality if and only if there exists a reference task time to compare it with. As a standalone number, this measurement is meaningless as an indication of software quality. However, ISO/IEC misleads the reader into thinking that this number is very relevant by stating that "the smaller [the result], the better". A smaller task time does not indicate better quality; a task time closer to a target time is indicative of better quality.
Cost of application	The cost of data item collection necessary for the application of this measure is negligible, as the collection can be integrated to the test phase.
Can this measure be used to thoroughly set quality goals and requirements?	As is explained in the "impact" section, this measure is only useful if there is reference task time to compare it with. ISO/IEC 9126-4 states that the smaller the result, the better. While not false, this statement is misleading. It would be better, and more useful as a quality requirement, if this measure were to be stated as follows: $X = \frac{T_m}{T_e}$ Where: • Tm is the measured task time • Te is the expected task time When used as a requirement, this forces the stakeholders to think about the time a task should take and the acceptable difference between the measured time and the reference time. Otherwise, the stakeholders are more likely to require a certain fixed task time without giving regards to variance between different users.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** is critical to a low task time. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the of the task time. **Efficiency** is also very important in this regard. The **time behaviour** measures will clearly be indicative of task time.

Conclusion	
Discussion	This measure can be used to indicate deficient quality (if used properly) at an acceptable cost. Furthermore, this measure fits into the predictive model proposed by ISO/IEC 9126-1. The usability of this measure as a quality goal or requirement is however questionable without a reference or target task time. This could however easily be fixed by modifying the definition of the measure.
Rating	This measure is **non-applicable** as is. However, it could easily be considered **applicable** with slight modifications.

Task Efficiency

Description	
Measure Name	Task Efficiency
Purpose	Measure how efficient the users are.
Application	This measure is applied as a user test: $X = M1/T$ Where: • M1 is the task effectiveness (see task effectiveness measure) • T is the task time ISO/IEC 9126-4 further notes: Task efficiency measures the proportion of the goal achieved for every unit of time. A high value indicates that a high proportion of the task is achieved in a small amount of time. It enables comparisons to be made, for example between fast error-prone interfaces and slow easy interfaces. If Task completion has been measured, task efficiency can be measured as Task completion/task time. This measures the proportion of users who were successful for every unit of time. A high value indicates a high proportion of successful users in a small amount of time.

Analysis	
Impact	A large value of X will theoretically correlate with high software quality. Therefore, this measure could be used to indicate deficient quality if the task efficiency is not satisfying.
Cost of application	The cost of applying the measure is negligible, as it uses data items collected in other measures.

Can this measure be used to thoroughly set quality goals and requirements?	This measure fails to be useful in setting quality goals and requirements. The reason for this is that it is very difficult to state that task efficiency should be greater than a certain value if ∘ No target task time has been set. ∘ No target value for task effectiveness has been set. As it has been seen previously, it is questionable that a specific goal could be set for task effectiveness (in its current state). This makes this measure inapplicable for setting quality goals and requirements. Furthermore, the units of X (*undefined value / time*) make this measure difficult to interpret and use.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	Since this is a derived measure, the same characteristics and subcharacteristics that were predictive for task time and task effectiveness are applicable here. Namely **Functionality** (with subcharacteristics **suitability** and **accuracy**) **Usability** (with subcharacteristics **understandability** and **operability**) and **Efficiency** (with subcharacteristic **time behaviour**) should be predictive of task efficiency.

Conclusion

Discussion	The applicability of this derived measure is very questionable, because the measurements composing this measure are not satisfying themselves as quality goals and requirements. It is not questionable that this measure can indeed be used to measure software quality *aposteriori*; what is very questionable is the *a priori* usability of this measure to set quality goals and requirements. In order for this measure to be applicable, improvements are needed to both the Task efficiency and Task time measures.
Rating	This measure is therefore **non applicable** in its current state. Modifications to the Task effectiveness measure would reflect positively on the applicability of this measure.

Economic Productivity

Description	
Measure Name	Economic Productivity
Purpose	Measure the cost-effectiveness of the user
Application	The measure is applied as a user test: $X = M1/C$ Where: ◦ M1 is the task effectiveness (see task effectiveness measure) ◦ C is the total cost of the task ISO/IEC 9126-4 further notes: Costs could for example include the user's time, the time of others giving assistance, and the cost of computing resources, telephone calls, and materials.

Analysis	
Impact	The higher the result, the better the economic productivity. An economic productivity that is too low might indicate deficient software quality. However, there is no way to set a threshold between good quality and bad quality.
Cost of application	There are two data items necessary to compute a result for this measure. ◦ First, the task effectiveness must be measured as described previously. This does not incur additional costs over those already incurred. ◦ The total cost of the task must be evaluated. Depending on the task and the necessary thoroughness of the evaluation, this might require an exhaustive and expensive investigation..

Can this measure be used to thoroughly set quality goals and requirements?	The applicability of this measure to set quality goals and requirements is questionable to start with because it relies on a measure that has been shown to have questionable applicability. In order for this measure to be useful as a requirement, the total acceptable cost of the task must be carefully estimated. Even if this measure can be indicative of low software quality, it is doubtful that it can be used as a software requirement because it would require two estimations (one for task effectiveness and another one for the cost) that will surely contain errors and render the resulting estimation of the economic productivity unusable. Furthermore, the units of X (*undefined value / $*) make this measure difficult to interpret and use.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	Since this is a derived measure, the same characteristics and subcharacteristics that were predictive for task effectiveness are applicable here. Namely **Functionality** (with subcharacteristics **suitability** and **accuracy**) and **Usability** (with subcharacteristics **understandability** and **operability**) should be predictive of task efficiency. **Efficiency** will also play an important role in predicting economic productivity, especially the **time behaviour** and **resource utilization** subcharacteristics.

Conclusion

Discussion	The applicability of this derived measure is very questionable, because the measurements composing this measure are not satisfying themselves as quality goals and requirements. Furhermore, the resulting unit of X lacks a useful interpretation.
Rating	This measure is therefore **non applicable**.

Productive Proportion

Description	
Measure Name	Productive Proportion
Purpose	Measure the proportion of the time a user is performing productive actions.
Application	This measure is applied as a user test: $X = Ta/Tb$ Where: • Ta is the productive time (task time – help time – error time – search time) (Note: this is not the same "Ta" then in other measures) • Tb is the task time (this corresponds to the Ta of other metrics) ISO/IEC 9126-4 further notes: This metric requires detailed analysis of a videotape of the interaction

Analysis	
Impact	This measure can clearly be used to indicate deficient software quality. The closer the result is to 1, the less time the user wastes in unproductive tasks like browsing the online help. This measure is sufficient to show deficient software quality. However, it is a necessary, but not sufficient condition to demonstrate adequate software quality.
Cost of application	Because this measure requires a detailed analysis, its application might be more expensive than other measures.
Can this measure be used to thoroughly set quality goals and requirements?	Because this measure uses a ratio of (partial time)/(total time), it is easier to set a quality goal or requirement. For example, a requirement based on this measure could be expressed as follows: *"The productive proportion for task* OrderBook *shall be greater than 90%".* Such a requirement will coerce the developers into producing software that is intuitive and ergonomic in order to diminish the time that is used searching for the right function or browsing the online or offline help.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** will be critical to achieving a satisfying productive proportion. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the effectiveness. **Efficiency** will also play an important role in predicting the productive proportion, especially the **time behaviour** and **resource utilization** subcharacteristics.

Conclusion	
Discussion	Although this measure might prove expensive to apply, it can clearly be used as an indicator of software quality and can also be used as a software quality goal or requirement.
Rating	This measure is considered **applicable**.

Relative User Efficiency

Description	
Measure Name	Relative User Efficiency
Purpose	Measure the efficiency of a user compared to an expert.
Application	This measure is applied as a user test: $X = A / B$ Where: • A is the task efficiency of an ordinary user • B is the task efficiency of an expert user The task efficiency is the same measure as previously discussed ISO/IEC 9126-4 further notes: The user and expert carry out the same task.

Analysis	
Impact	According to ISO/IEC 9126-4, the closer the ratio is to 1, the better the quality of the software. The accuracy of the previous statement is questionable. A concrete example will illustrate why this statement is not always accurate. This example will focus on the popular text editor emacs. This text editor is renowned to be relatively complex to learn and has a steep learning curve. However, once the learning phase is complete, it allows for productivity that few text editors can attain. The cost of this power is increased complexity. Therefore, the relative user efficiency will be low. However, one can not say on this measure alone that emacs is of poor quality. Therefore, one can not blindly say that a high relative user efficiency is synonymous of quality. The only case where this is true is when the expert's efficiency is equal to the theoretical maximum.
Cost of application	The cost of applying the measure is negligible, as it uses data items collected in other measures.

Can this measure be used to thoroughly set quality goals and requirements?	Even if the given example seems to undermine the usability of this measure, it does not deter from its instinctive meaning. In some cases, a high relative user efficiency might be desirable if a shallow learning curve is needed. For example in a call center, it might be cheaper to add more operators than to pay for more clever software that allows for higher raw efficiency from more experienced operators.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** is critical to a low task time. Measures from the **understandability**, **operability** and especially **learnability** subcharacteristics should therefore be predictive to a certain extent of the effectiveness.

Conclusion

Discussion	This measure is an example of one that is not very useful as an absolute measure of quality, but can be very useful as a quality requirement. Its usability as an *a posteriori* measurement is more than questionable, because it is doubtful that high relative user efficiency correlates directly with high software quality. However, it is useful *a priori* (i.e. as a quality requirement). Indeed, it might be necessary for certain types of applications to have a relative user efficiency close to 1.
Rating	This measure is considered **applicable**.

User Health and Safety

Description

Measure Name	User Health and Safety
Purpose	Measure the incidence of health problems among users of the product.
Application	This measure is applied by analyzing usage statistics: $X = 1 - A / B$ Where: • A is the number of problems reported. Problems can include Repetitive Strain Injury (RSI), fatigue, headaches, etc. • B is the total number of users.

Analysis

Impact	Properly applied, this measure can be used to demonstrate deficient software quality. However, it might be difficult to prove that the software itself is the root cause of the problems. For example, wrist problems might be traceable to the way the user uses the mouse rather than to the software.
Cost of application	This measure entails a detailed analysis of the usage statistics. When statistics reveal a problem, more analysis and interviews might be needed in order to find the root cause of the problem. Depending on the depth of the analysis, the application of this measure might prove costly.
Can this measure be used to thoroughly set quality goals and requirements?	Because this measure is in the form of an absolute ratio, it is readily usable as a goal or requirement. It is plausible that the stakeholders require that the software does not cause any prejudice to the user health and safety. Such a requirement would however force the stakeholders to reflect on the definition of problems to user health and safety. The standard offers little guidance in this sense.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	No External Quality characteristics and subcharacteristics directly relate to user health and safety. **Usability** subcharacteristics are loosely related to user health and safety. Software that possesses **understandability** and **operability** are less likely to let users do something that might endanger them.

Conclusion	
Discussion	This measure can be used to specify quality goals and requirements. However, its usability in the predictive model is questionable. No External Quality characteristics or subcharacteristics directly relate to user health and safety.
Rating	The applicability of this measure is therefore **conditional** to definition of a health problem traceable to software use.

Safety of People Affected by Use of the System

Description	
Measure Name	Safety of People Affected by Use of the System
Purpose	Measure the incidence of hazard to people affected by use of system.
Application	This measure is applied by analyzing usage statistics: $X=1-A/B$ Where: ◦ A is the number of people put at hazard ◦ B is the total number of people potentially affected by the system. ISO/IEC 9126-4 further notes: An example of this metric is Patient Safety, where A = number of patients with incorrectly prescribed treatment and B = total number of patients.

Analysis	
Impact	This measure can clearly be used to indicate deficient software quality. An interesting fact about this measure is that it measures damages not only to the end user, but also to anybody who might be affected by the system. For example, if there is a power failure attributable to a software failure in a power plant, then ◦ A is the number of people affected by the blackout. ◦ B is the number of people who could potentially have been affected (worst case scenario). Therefore, this measure can be very important for measuring the quality in use of embedded applications, even though no one directly interacts with it.
Cost of application	This measure entails a detailed analysis of the usage statistics. When statistics reveal a problem, more analysis and interviews might be needed in order to find the root cause of the problem. A detailed analysis will also be necessary to uncover the number of people who have been put to hazard and the number of people potentially affected by the system. Depending on the depth of the analysis, the application of this measure might prove costly.

Can this measure be used to thoroughly set quality goals and requirements?	This measure can be used in a number of ways to set quality goals and requirements. Usage of this measure as a quality goal or requirement will force the stakeholders to: ◦ Consider and define the number of people who can potentially be affected by the usage of the software. ◦ Consider and define what percentage of the potentially affected population must be kept safe at all times. ◦ It can help the stakeholders define an upper limit on the number of people who should potentially be affected by usage of the software. It is important to note that it might be hard to demonstrate a level of safety if the **testability** of the software is not sufficient. Finally, it would be important to define what "potentially affected by" and "put to hazard" means. The standard offers no guidance in this case, but it is a clarification that must be made by the stakeholders in order for this measure to be usable.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** is important to the safety of people affected by the system. Measures from the **understandability** and **operability** subcharacteristics should therefore be predictive to a certain extent of the relative safety of people. **Maintainability** will also be very important. **Analyzability**, **changeability**, **stability**, and **testability** subcharacteristics are critical to continued safety.

Conclusion	
Discussion	This measure has an important impact, can be used to thoroughly set quality in use requirements and fits into the predictive model of ISO/IEC 9126-1.
Rating	This measure is considered **applicable**.

Economic Damage

Description	
Measure Name	Economic Damage
Purpose	Measure the incidence of economic damage.
Application	This measure is applied by analyzing usage statistics: $X=1-A/B$ Where: • A is the number of occurrences of economic damage. • B is the total number of usage situations. ISO/IEC 9126-4 further notes: This can also be measured based on the number of occurrences of situations where there was a risk of economic damage.

Analysis	
Impact	Software that causes unforeseen economic damages clearly possesses deficient quality. This measure can therefore be used to a certain extent to measure quality. However, the impact of this measure is greatly mitigated by the fact that the economic damage is not weighted. For example, economic damages of 10$, 1,000$ and 1,000,000$ are considered on the same level.
Cost of application	This measure entails a detailed analysis of the usage statistics. When statistics reveal a problem, a thorough analysis will be necessary to uncover the parameters of the measure.

Can this measure be used to thoroughly set quality goals and requirements?	The usability of this measure to thoroughly set quality goals and requirements is very questionable. The stakeholders will probably be more interested in minimizing the total amount of economic damages rather than the number of occurrences of economic damages. Therefore, in order for this measure to be usable as a quality requirement, it must be complemented with such information.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Reliability** subcharacteristics are directly related to economic damages. Software that possesses **maturity**, **fault tolerance** and **recoverability** is less likely to cause economic damages. If economic damages do occur, their impact might be lessened. **Usability** subcharacteristics are loosely related to possible economic damages. Software that possesses **understandability** and **operability** are less likely to let users do something that cause economic damages. **Maintainability** might also play an important role in the long-term. **Analyzability**, **changeability**, **stability**, and **testability** subcharacteristics will prove important to prevent corruption when modifications to the software are made.

Conclusion	
Discussion	This could be an important measure of quality in use. It is very important to consider the potential economic damages when building software. However, it does not take into account the value of the economic damages, which makes its impact and usability as a requirement questionable.
Rating	The applicability of this measure is therefore **conditional** to the inclusion of the notion of maximum damage.

Software Damage

Description	
Measure Name	Software Damage
Purpose	Measure the incidence of software corruption.
Application	This measure is applied by analyzing usage statistics: $X=1-A/B$ Where: ◦ A is the number of occurrences of software corruption. ◦ B is the total number of usage situations. ISO/IEC 9126-4 further notes: This can also be measured based on the number of occurrences of situations where there was a risk of software damage. This metric can also be measured as X = cumulative cost of software corruption / usage time.

Analysis	
Impact	In this analysis, "software" is taken as the program itself and the data it manipulates. Software that corrupts itself or unwillingly comprises data it uses will undoubtedly have poor quality. Therefore, this measure can be used to evaluate software quality.
Cost of application	This measure entails a detailed analysis of the usage statistics. Detecting corruption of data might require analysts to manually go through the computations carried out by the program. The costs of such an analysis will vary with the complexity of the operations carried out by the program.

Can this measure be used to thoroughly set quality goals and requirements?	For reasons similar to those explained in the analysis of the economic damage measure, the applicability of this measure to thoroughly set quality goals and requirements is questionable. While this measure is certainly useful, the stakeholders will probably be more interested in limiting or defining the extent of corruption that is allowable on specific data sets rather than on the exact number of times corruption occurs. The problem with analyzing only the number of times corruption occurs is that there are different levels of corruption. For example, if a software program corrupts a document by replacing every period by a coma, it is certainly less damageable than if it makes the document unreadable. The standard specifies that another possible measurement of software damages is: X = cumulative cost of software corruption / usage time. This amounts to computing the economic damages related to software corruption and is more relevant for stakeholders.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Reliability** subcharacteristics are directly related to economic damages. Software that possesses **maturity**, **fault tolerance** and **recoverability** is less likely to cause economic damages. If economic damages do occur, their impact might be lessened. **Usability** subcharacteristics are loosely related to possible economic damages. Software that possesses **understandability** and **operability** are less likely to let users do something that cause might corruption **Maintainability** might also play an important role in the long-term. **Analyzability**, **changeability**, **stability**, and **testability** subcharacteristics will prove important to prevent corruption when modifications to the software are made.

Conclusion

Discussion	Measuring the extent of corruption the software can cause on itself or data it analyzes is undoubtedly an important measure of software quality. While this measure addresses that issue, it does not do so in a way that is the most relevant to stakeholders that specify software requirements.
Rating	The applicability of this measure is therefore **conditional** to the usage of the alternative definition of the measure.

Satisfaction Scale

Description

Measure Name	Satisfaction Scale
Purpose	Measure the satisfaction of the user.
Application	The measure is applied as a user test: $X = A / B$ Where: • A is a questionnaire producing psychometric scales. • B is the population average.

Analysis

Impact	There is no doubt that a questionnaire producing psychometric scales can be used to analyze the satisfaction of users with the software product. To the extent that satisfaction is a measure of software, then this measure is a relevant indicator of quality in use. The analysis of the mathematical formula states that the larger the result, the better. It could be further said that results above one will be indicative of a score that is above the population average. The existence of psychometric tests that are relevant for the software being analyzed is necessary in order for this measure to be applicable. The population average must also be known in order for a comparison to be made. ISO/IEC 9126-4 contains references to psychometric that have been used by the industry. However, there are no standardized tests.
Cost of application	The cost of application will vary with the complexity of the psychometric test. In some cases, a license for the test might be needed. In other cases, the application of the test might necessitate the help of a specialist. In any cases, the users that will be subjected to the test must be chosen with care.

Can this measure be used to thoroughly set quality goals and requirements?	Psychometrics form a well understood and well recognized body of knowledge. Psychometrics tests are interesting for setting quality requirements because such tests always have a known population average. This average can serve as guidance for setting a relevant requirement.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	**Usability** is surely the External Quality characteristic that is the most important to user satisfaction. Measures from the **understandability**, **learnability**, **operability** and **attractiveness** subcharacteristics should therefore be predictive to a certain extent of the satisfaction of end users. Depending on the questionnaire, all characteristics and subcharacteristics except those from **maintenance** could be predictive the result.

Conclusion

Discussion	The impact of this measure was shown to be positive (i.e. there is no doubt that this measure is an indicator of user satisfaction and therefore quality). However, the cost of applying this measure could be prohibitive depending on the thoroughness of the analysis. It is important to note that this measure as the potential to be one where almost all External Quality subcharacteristics might prove predictive of the result.
Rating	The applicability of this measure varies with the level of understanding of the stakeholders and the complexity of the psychometric tests. If the stakeholders have a thorough understanding of psychometrics, there is no doubt that this measure is **applicable**.

Satisfaction Questionnaire

Description

Measure Name	Satisfaction Questionnaire
Purpose	Measure how satisfied the user is with specific software features.
Application	The measure is applied as a user test: $X=\sum(A_i/n)$ Where: • Each A_i is a response to the same question • n is the total number of responses ISO 9126-4 further notes: If the questionnaire items are combined to give an overall score, they should be weighted, as different questions may have different importance.

Analysis

Impact	This measure is one of the most powerful tools to measure user satisfaction and to evaluate quality as perceived by the users of the system.
Cost of application	The cost of applying this measure is negligible in most cases. If an expert is hired to construct the questionnaire and to perform the evaluation, the cost of application could rise.
Can this measure be used to thoroughly set quality goals and requirements?	Because of its relative simplicity and its expressiveness, this measure can be used to thoroughly set quality goals and requirements. Stakeholders can determine in advance what questions should be asked to users and what their level of satisfaction should be for the software product to be considered successful. From those questions and the target satisfaction level, new External Quality requirements can be discovered that will help satisfy the objectives. In software where there is user interaction and feedback, this measure can be used to discover many requirements.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	Depending on the questions that are asked, almost every subcharacteristic can be predictive of the result of the application of this metric. The relation between External Quality and this measure is almost assured, but will have to be evaluated on a case by case basis.

Conclusion

Discussion	The impact of this measure is important and the cost of applying it is negligible. It gives the end users the chance to express their satisfaction with the software program. Furthermore, it allows stakeholders to thoroughly set quality goals and requirements. Finally, the role of this measure in the predictive model is almost assured, but will have to be determined depending on the questions submitted to the user. This measure is of critical importance to the quality in use model.
Rating	This measure is considered **applicable**.

Discretionary Usage

Description	
Measure Name	Discretionary Usage
Purpose	Measure the proportion of potential users who choose to use the system.
Application	The measure is applied observing users: $X = A / B$ Where: • A is the number of times that specific software functions/applications/systems are used. • B is the number of times that the specific software functions/applications/systems are intended to be used. ISO 9126-4 further notes: This metric is appropriate when usage is discretionary.

Analysis	
Impact	When users choose to use a system on their own, it is undoubtedly a sign of their satisfaction. To the extent that satisfaction is a measure of software quality, then this measure can be used to evaluate quality in use. However, in most business cases, usage of the software is not discretionary. Therefore the impact of this measure is limited in those cases.
Cost of application	The application of this measure can not be automated and requires an analysis of user actions and potentially interviews with users. It can thus prove costly.
Can this measure be used to thoroughly set quality goals and requirements?	In the case where usage of the software is discretionary, this measure may be used to set quality goals and requirements. However, discretionary usage should be a goal that is always strived for.
Which, if any, External Quality characteristics and subcharacteristics may predict the value of this measure?	Most **Usability** subcharacteristics are loosely related to discretionary usage. Software that possesses **learnability** and **attractiveness** are more likely to encourage users to use them. Also, software that possesses the proper **Functionality** is more likely to favour discretionary usage. Finally, **Portability** could be an important to discretionary usage. The **replaceability** subcharacteristic will prove to be particularly important, but **adaptability**, **installability**, and **co-existence** will also be meaningful.

Conclusion

Discussion	Discretionary usage is undoubtedly a sign of good quality. There is almost no greater consecration of quality that when users rush to use your product by their own free will. However, in its current state, this measure can not be used to do more than just state a broad goal. Furthermore, such a goal should be at the heart of almost any product development effort, even when the user will be forced to use the system. Furthermore, the standard clearly states that this measure is only applicable when usage is discretionary. It is important to note that this measure is one of the few that is traceable to the portability External Quality characteristic.
Rating	The applicability of this measure is **conditional** to usage being discretionary.

APPENDIX 2

SUGGESTED IMPROVEMENTS TO ISO/IEC 9126-4

Task Effectiveness

This measure is complicated and needs to be more thoroughly explained. By trying to be concise, the standard obfuscates the usefulness of this measure.

The solution in making this measure applicable and usable requires a complete reformulation. The standard should first state that each task that a software product must accomplish should be decomposed into goals. The accomplishment of those goals, whether partial or complete, should result in the success of the task. Each goal (G) must be given a value representing the *approximate* percentage of the task (P_G) that is attained when the goal is accomplished. The sum of those percentages should be 100%. Some of the goals might be marked as "essential", meaning that failure to accomplish those goals will result in 0% task effectiveness. The task effectiveness is the following sum:

$$TE = \begin{matrix} \sum_{task} P_G, & \mathit{When\ all\ essential\ goals\ are\ attained} \\ 0\%, & \mathit{Otherwise} \end{matrix}$$

A threshold for acceptable quality can then be set on a task by task basis by determining which goals :

- are essential
- are desirable
- are "nice to have"

This classification can be made from many perspectives: user, business, economic, etc. The target task effectiveness is the sum of the percentages associated to the essential and desirable goals. The task effectiveness can then be analyzed for many users and meaningful conclusions can be drawn from its application.

The results of these changes are important:

- The impact of the measure is now very important. Each task that falls below the target task effectiveness has unacceptable quality.

- The cost of the application remains negligible.
- The scoring scheme does not need to be refined iteratively anymore. The percentages associated to each goal are not even really important. They only help in quantifying the contribution of each goal.
- The usage of this measure will help stakeholders define a clear acceptance criterion on a task by task basis.

Implementation of these changes would radiate positively throughout the Quality in Use model because many other measures depend on this one. For example, it would help defining a clear acceptance for tasks that are composed of multiple goals and thus make the task completion measure generally applicable.

Task Completion

This measure can not be used to measure the task completion when complex tasks are involved. Complex tasks are those that are composed of multiple tasks. This is due to the reliance of this measure on the Task Effectiveness measure. According to the analysis, the Task Effectiveness measure has been shown to be non-applicable.

In order to make this measure applicable in every situation, the modifications discussed above for the Task Effectiveness measure should be implemented.

Error Frequency

The impact of this measure and its applicability to thoroughly set quality requirements have been judged inadequate. By clarifying the application method, these two issues can be resolved and this measure can be made relevant in the context of ISO/IEC 9126-4.

First of all, the measure should be separated into two in order to account for and emphasize the different aspects. Therefore, there should be a measure called “Temporal Error Frequency” and another one named “Task Error Frequency”.

The “Task Error Frequency” could be defined as follows:

1. Select a task.

2. Determine the error condition(s).
3. For every 100 times the task is executed, determine the acceptable number of errors.
4. Measure for each task.

A single user or a group of user could be used for measurement. Points 1 to 3 can and should be carried out during requirements engineering. Point 4 can be carried out at any time to measure quality.

The “Temporal Error Frequency” should be defined as follows:

1. Select a task
2. Determine the maximum amount of time that is allowable per task. Failure to accomplish the task within the given amount of time results in an error.
3. For every 100 times the task is executed, determine the acceptable number of temporal errors.
4. Measure for each task.

Once again, points 1 to 3 can be carried out during requirements engineering while point 4 can be carried out to measure quality at any time.

Task Time

As was explained in the analysis of this measure, the measure of task time does not correlate with Quality in Use. This is in opposition with what is stated in the ISO/IEC 9126 document.

The reason that task time does not correlate with Quality in Use is that some tasks need to take place in a defined amount of time. If the task is done faster, the *quality* of the system will not be any better. Thus what is important is not the task time itself, but rather the difference between the expected task time and the actual task time.

The measure can be enhanced by redefining the metric as follows:

$$X=\frac{T_m}{T_e}$$

Where:

- Tm is the measured task time
- Te is the expected task time

During requirements engineering, the stakeholders must define the estimated task time for each task. A range of acceptable values must also be determined for *Tm*. For example the following values could be determined:

$$T_e=10s$$
$$T_{mr}\in[8,11]$$

Where T_{mr} is the range of acceptable values for T_m.

This means that the expected task time is 10 seconds. The range means that task times between 8 and 11 seconds are acceptable. The acceptable range for X is therefore:

$$X\in[0.8,\ 1.1]$$

This definition of task time allows for a range of possibilities. For example, it is possible to define a case where there is no limit to how fast the task can be executed:

$$T_e=10s$$
$$T_{mr}\in[0,10]$$

The range of acceptable values for X is now between 0 and 1. Therefore, one can not blindly say that a value for X closer to 1 is synonymous with higher Quality in Use.

The proposed modifications will allow this measure to be usable in thoroughly setting quality goals and requirements, while being a clear indication of Quality in Use.

Task Efficiency

This measure was judged non-applicable because it relied on two measures that were themselves of questionable applicability.

The applicability of this measure must be judged anew now that the Task Effectiveness and Task Time measures have been changed.

Application: The Task Efficiency is measured as:

$$X = \frac{TE}{T_m}$$

Where

- TE is the measured Task Effectiveness as proposed in this document.
- T_m is the time that is measured for the task.

Impact: The result of the measurement can be interpreted as the percentage of a task that is accomplished by unit of time. Generally, the greater the better.

Cost: The cost remains negligible.

Usability to set quality goals and requirements: During the requirements engineering phase, the following computation should be made:

$$X = \frac{TE}{T_{mr}}$$

Where

- TE is the estimated Task Effectiveness as proposed in this document.
- T_{mr} is the range of acceptable values for the task time (refer to Task Time measure).

X will thus be a range of values that represent the minimum acceptable Task Efficiency. Stakeholders should evaluate this result and judge if is seems reasonable and acceptable. If it is not, they should review their estimates for the Task Effectiveness and Task Time.

This measure is useful in the requirements engineering phase as a validation of the values expected for the Task Effectiveness and Task Time. Stakeholders should not try to directly set this measure, as it depends on two other measures.

Predictability: There is no change to the role of this measure in the predictability model.

Conclusion: This measure is now applicable and conformant to the objectives of ISO/IEC 9126. While it can not be used as a requirement, it can certainly be used to validate two other measures that are critical to the effectiveness of the model.

Economic Productivity

This measure was judged non-applicable because it relied on a measure that was itself of questionable applicability.

The applicability of this measure must be judged anew now that the Task Effectiveness and measure has been changed.

Application: The Economic Productivity is measured as:

$$X = \frac{TE}{C}$$

Where

- TE is the measured Task Effectiveness as proposed in this document.
- C is the cost of accomplishing the task.

Impact: The result of the measurement can be interpreted as the percentage of a task that is accomplished by unit of cost. Generally, the greater the better. It is still difficult to set a threshold distinguishing deficient from sufficient quality. However, it is easier to interpret the value because the ratio involves units that are easier to understand.

Cost: The cost remains negligible.

Usability to set quality goals and requirements: During the requirements engineering phase, the following computation should be made:

$$X = \frac{TE}{C_e}$$

Where

- TE is the estimated Task Effectiveness as proposed in this document.
- C_e is the estimated cost of the task. When estimating the total cost of the task, stakeholders should take into account the estimated task time and the cost of computing resources.

X will represent the estimated Economic Productivity. As is the case for the Task Efficiency measure, it is not recommended that it be used directly as a requirement or goal, as it is dependant on an estimation of the cost and the Task Effectiveness measure. Furthermore, the estimated cost is directly related to the estimated Task Time.

This measure is useful in the requirements engineering phase as a validation of the values expected for the Task Effectiveness. Stakeholders should not try to directly set this measure, as it depends on another measure If the resulting estimation of the Economic Productivity is not satisfying, the stakeholders should review the estimation of the Task Efficiency.

Predictability: There is no change to the role of this measure in the predictability model.

Conclusion: This measure is now applicable and conformant to the objectives of ISO/IEC 9126. While it can not be used as a requirement, it can certainly be used to validate another measure that is critical to the effectiveness of the model.

User Health and Safety

The applicability of this measure was questionable because it did not play an important role in the predictable model proposed by ISO/IEC 9126-1. The best way to improve this wouldn't lie in modifying this measure, but rather in modifications to the External Quality model. If there were an "Ergonomics" subcharacteristics attached to the Usability characteristic, it would be predictive of User Health and Safety. Such a subcharacteristic should include measures evaluating:

- The choice of colors

- The disposition of the widgets
- etc.

A complete definition of such a subcharacteristic is beyond the scope of this work.

Economic Damage

Analyzing the economic damages that can occur due to the usage of a software product is an important part of measuring Quality in Use. However, it was shown that this measure can not be used effectively for this purpose in its original format.

In order to improve this measure, its definition should be changed. Instead of focusing on the number of occurrences of economic damages, it should focus on the monetary losses that can be associated to the economic damage.

As a first step in improving this measure, the standard should provide guidance on the evaluation of economic damages. For example, economic damages could be classified as follows:

- Damages to infrastructure that are traceable to software failure and for which the developers can be held responsible.
- Damages to people that are traceable to software failure and for which the developers can be held responsible.
- Loss in future business that is due to poor software performance.
- etc.

Then, potential economic damages should be first evaluated on a task by task basis. If tasks are dependent on one another, then potential economic damages should be evaluated for different scenarios that combine task failures.

The formula for computing economic damages should be as follows:

$$Ed = \sum_{\text{economic damage scnarios}} O_n \times D$$

Where:

- Ed is the amount of economic damages.
- O_n is the number of times a damage scenario has occurred.
- D is the measured[14] amount of damage.

During the requirements engineering phase, the probability of occurrence of an economically damaging scenario should be grossly evaluated.

The measure could then be used as follows:

$$Ed = \sum_{economic\ damage\ scnarios} O_p \times D_e$$

Where:

- Ed is the amount of damages that are probable to occur.
- O_p is the likeliness or probability of occurrence of an economically damaging scenario.
- D_e is the estimation of the economic damages[15].

If the amount of economic damages is deemed unacceptable, a Pareto analysis could then be conducted. The result of this analysis should be used to indicate in which scenarios reliability and usability should be reinforced. This measure would then be usable in setting External Quality goals and requirements.

By implementing these changes, the impact of this measure will be improved. It has also been shown that it would then be usable in setting quality goals and requirements. It would thus become usable with respect to the objectives of ISO/IEC 9126-1.

Software Damage

As is hinted to in the ISO/IEC 9126-4 standard, damage to the software will result in economic damages. Therefore, this measure could be merged into the economic damage measure. It has been proposed that causes of economic damage be classified into different categories. Software damage could be such a category.

[14] As evaluated by damage assessment experts. This number may include: direct losses, loss of business, etc.

[15] This should be evaluated by damage assessment experts.

Satisfaction Scale

This measure should only be used by stakeholders knowledgeable about psychometrics. Only expert stakeholders with the appropriate knowledge can fully understand the implications of setting requirements based on psychometric tests.

This measure can not be transformed to be applicable for stakeholders with less knowledge without diminishing its impact.

Therefore, the standard should clearly warn users that this measure should only be used as a requirement when stakeholders have the appropriate understanding of psychometrics.

Furthermore, the standard should include clear references to psychometrics test that are applicable to Quality in Use.

APPENDIX 3

REVISED ISO/IEC 9126-4 MEASURE TABLES

The following pages present the suggested modifications to the ISO/IEC 9126-4 standard as a result of the analysis presented in this thesis.

8.1 Effectiveness metrics

Effectiveness metrics assess whether the tasks performed by users achieve specified goals with accuracy and completeness in a specified context of use. They do not take account of how the goals were achieved, only the extent to which they were achieved (see E.2.1.2).

Table 8.1 Effectiveness metrics

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
Task Effectiveness	Measure the proportion of the goals of the task that is achieved correctly?	User test	$TE = \sum_{tasks} P_G$, Essential goals attained 0% , *Otherwise*	0.0≤ TE ≤ 1.0 The closer to 1.0 the better	-	PG = Percentage associated with an attained goal.	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer
NOTE To use this metric, the task to be analyzed should be decomposed into goals. The accomplishment of those goals, whether partial or complete, should result in the success of the task. Each goal (G) must be given a value representing the approximate percentage of the task (PG) that is attained when the goal is accomplished. The sum of those percentages should be 100%. Some of the goals might be marked as "essential", meaning that failure to accomplish those goals will result in 0% task effectiveness. An appropriate level for task effectiveness can be established during requirement engineering for different contexts of use.									
Task Completion	What proportion of the tasks are completed?	User test	$X=A/B$ A = number of tasks completed B = total number of tasks	0.0≤ X ≤ 1.0 The closer to 1.0 the better	Ratio	A=Count B=Count X=Count / Count	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer
NOTE This metric can be measured for one user or a group of users. If tasks can be partially completed the Task effectiveness metric should be used..									
Task Error Frequency	Measure the frequency of task errors.	User test	$X=A/T$ A = number of times a user made an error that resulted in task failure T = number of times the task was tried	0.0≤ X ≤ 1.0 The closer to 0.0 the better	Ratio	A=Count T=Count X=Count / Count	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer
NOTE The goals for this metric should be established during requirements engineering for each task to be measured. The error conditions that cause a task to fail should be determined. An acceptable failure ratio can then be set on a task by task basis.									

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
Temporal Error Frequency	Measure the frequency of task errors attributable to temporal reasons.	User test	$X = A / T$ A = number of times a user took too much or too little time to complete a task T = number of times the task was tried	0.0≤ X ≤ 1.0 The closer to 0.0 the better	Ratio	A=Count T=Count X=Count / Count	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer

NOTE It is important to distinguish this metric from the previous one. In this case, the errors that are observed are related to temporal mistakes. For example, a user took too much time to complete a task efficiently. The goals for this metric should be established during requirements engineering for each task to be measured. The temporal error conditions that cause a task to fail should be determined. An acceptable failure ratio can then be set on a task by task basis.

8.2 Productivity metrics

Productivity metrics assess the resources that users consume in relation to the effectiveness achieved in a specified context of use. The most common resource is time to complete the task, although other relevant resources could include the user s effort, materials or the financial cost of usage.

Table 8.2 Productivity metrics

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
Task time	Measure the difference between the desired task time and the actual task time.	User test	$X=\frac{T_m}{T_e}$ Tm = measured task time Te = expected task time	The closer X is to 1, the closer the closer the result is to the result is to the expected value. This is not necessarily a indication of quality.	Ratio	Tm = Time Te = Time	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer
NOTE Task time by itself is note a measure of quality in use. It is recommended that an acceptable range of values of Td be determined during requirements specification.									
Task efficiency	How efficient are the users?	User test	$X=TE/Tm$ TE = task effectiveness Tm = measured task time	0.0≤ X Generally, the larger X, the better.	-	TE = percentage Tm = Time X = percentage / time unit	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer
NOTE 1 Task efficiency measures the proportion of the goal achieved for every unit of time. A high value indicates that a high proportion of the task is achieved in a small amount of time. It enables comparisons to be made, for example between fast error-prone interfaces and slow easy interfaces (see for example F.2.4.4). NOTE 2 If Task completion has been measured, task efficiency can be measured as Task completion/task time. This measures the proportion of users who were successful for every unit of time. A high value indicates a high proportion of successful users in a small amount of time. NOTE 3 During requirements specification, this metric could be used as a validation of expectations for the Task Time and Task Effectiveness metrics.									

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
Economic productivity	How cost effective is the user?	User test	$X = TE / C$ TE = task effectiveness C = total cost of the task	$0.0 \leq X$ Generally, the larger X, the better.	-	TE = percentage C = monetary unit X = percentage / monetary unit	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer

NOTE 1 Costs could for example include the user s time, the time of others giving assistance, and the cost of computing resources, telephone calls, and materials.
NOTE 2 During requirements specification, this metric could be used as a validation of expectations for the Task Effectiveness metric.

Productive proportion	What proportion of the time is the user performing productive actions.	User test	$X = Ta / Tb$ Ta = productive time = task time – help time – error time – search time Tb = task time	$0.0 \leq X \leq 1.0$ The closer to 1.0 the better	Absolute	Ta = Time Tb = Time X = Time / Time	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer

NOTE This metric requires detailed analysis of a videotape of the interaction (see Macleod M, Bowden R, Bevan N and Curson I (1997) The MUSiC Performance Measurement method, Behaviour and Information Technology, 16, 279-293.).

Relative user efficiency	How efficient is a user compared to an expert?	User test	$X = A / B$ A = ordinary user's task efficiency B = expert user's task efficiency	$0.0 \leq X \leq 1.0$ Generally, the closer to 1.0 the better	Absolute	A = percentage B = percentage X = percentage / percentage	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer

NOTE The user and expert carry out the same task. If the expert was 100% productive, and the user and expert had the same task effectiveness, this metric would give a similar value to the Productive proportion.

8.3 Safety metrics

Safety metrics assess the level of risk of harm to people, business, software, property or the environment in a specified context of use. It includes the health and safety of the both the user and those affected by use, as well as unintended physical or economic consequences

Table 8.3 Safety metrics

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
User health and safety	What is the incidence of health problems among users of the product?	Usage statistics	$X = 1 - A / B$ A = number of users reporting RSI B = total number of users	0.0≤ X ≤ 1.0 Generally, the closer to 1.0 the better	Absolute	A = count B = count X = count / count	Usage monitoring record	5.4 Operation	User Human interface designer

NOTE Health problems can include Repetitive Strain Injury, fatigue, headaches, etc.

Safety of people affected by use of the system	What is the incidence of hazard to people affected by use of the system?	Usage statistics	$X = 1 - A / B$ A = number of people put at hazard B = total number of people potentially affected by use of the system	0.0≤ X ≤ 1.0 Generally, the closer to 1.0 the better	Absolute	A = count B = count X = count / count	Usage monitoring record	5.3 Qualification Testing 5.4 Operation	User Human interface designer Developer

NOTE 1 An example of this metric is Patient Safety, where A = number of patients with incorrectly prescribed treatment and B = total number of patients.
NOTE 2 If using this metric as a requirement, it is important to define "put to hazard" and "potentially affected".

Economic damage	What is the incidence of economic damage?	Usage statistics	$Ed = \sum_{\text{damage scenarios}} O_n * D$ On = number of times a damaging scenario has occurred. D = amount of economic damages	0.0≤ X The closer to 0. the better	Absolute	On = count D = monetary value Ed = monetary value	Usage monitoring record	5.3 Qualification Testing 5.4 Operation	User Human interface designer Developer

NOTE 1 Economic damages include but are not limited to: damages to infrastructure that are traceable to software failure and for which the developers can be held responsible, damages to people that are traceable to software failure and for which the developers can be held responsible, loss in future business that is due to poor software performance, etc.
NOTE 2 Potential economic damages should be first evaluated on a task by task basis. If tasks are dependent on one another, then potential economic damages should be evaluated for different scenarios that combine task failures.
NOTE 3 During requirements specification, the following formula could be used instead: $Ed = \sum_{\text{damage scenarios}} O_p * D_e$, where Op is the estimated probability of occurrence of the scenario and De is the estimated resulting damage. If the resulting evaluation of possible economic damages is too high, an analysis should be conducted to verify where External Quality characteristics should be insisted upon to reduce the risk.

8.4 Satisfaction metrics

Satisfaction metrics assess the user s attitudes towards the use of the product in a specified context of use.

NOTE: Satisfaction is influenced by the user's perception of properties of the software product (such as those measured by external metrics) and by the user's perception of the efficiency, productivity and safety in use.

Table 8.4 Satisfaction metrics

Metric Name	Purpose of the metrics	Method of application	Measurement, formula and data element computations	Interpretation of measured value	Metric scale type	Measure type	Input to measurement	12207 Reference	Target Audience
Satisfaction scale	How satisfied is the user?	User test	$X=A/B$ A = questionnaire producing psychometric scales. B = population average	0.0<X The larger the better.	Ratio	A = count X = count	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer Developer

NOTE 1 Examples of psychometric questionnaires can be found in F.3.
NOTE 2 Such a metric should only be used during requirements engineering if stakeholder have an appropriate knowledge of psychometrics.

Satisfaction Questionnaire	How satisfied is the user with specific software features?	User test	$X=\sum(A_i)/n$ Ai = response to a question n = number of responses	0.0≤ X ≤ 1.0 The closer to 1.0 the better	Ord.	A = count X = count	Operation (test report) User monitoring record	6.5 Validation 5.3 Qualification Testing 5.4 Operation	User Human interface designer

NOTE If the questionnaire items are combined to give an overall score, they should be weighted, as different questions may have different importance.

BIBLIOGRAPHY

Adey, C. A. & Hill, G. K. (2000). *Quality / ISO 9000 as a Marketing Tool*, [En ligne]. http://www.smps.org/mrc/articles/0200qualityiso.pdf

Bazzana, G., Anderson, O., & Jokela, T. (1993). *ISO 9126 and ISO 9000: Friends or foes?* Presented at Software Engineering Standards Symposium.

Berrazouane, A. (2005). *Title to be determined* (Master thesis to be presented at ETS in 2005)

Biehl, R. E. (2001). *Six sigma for Software.* IEEE Software, 21(2), 68-70.

Boddie, J. (2000). *Do We Ever Really Scale Down?*, IEEE Software, 17(5), 79-81.

Boehm, B. W., Brown, J. R., Kaspar, J. R., Lipow, M. L. & MacCleod, G. (1978). *Characteristics of Software Quality.* New York: American Elsevier.

Boehm, B. W., Brown, J. R., Lipow, M. L. (1976). *Quantitative Evaluation of Software Quality.* Proceedings of the 2nd international conference on Software engineering, San Fransisco, California, United States, 592-605, IEEE Computer Society Press.

Côté, M.-A., Suryn, W., Martin, R. A., Laporte, C. Y. (2004a). *Evolving a Corporate Software Quality Assessment Exercice: A Migration Path to ISO/IEC 9126*, Software Quality Professional, 6(3), 4-17.

Côté, M.-A., Suryn, W., Martin, R. A., Laporte, C. Y. (2004b). *The analysis of the industrial applicability of software product quality ISO standards: the context of MITRE's Software Quality Assessment exercise*, in Proceedings of the 12th International Software Quality Management & INSPIRE Conference (BSI) 2004, Canterbury, Kent, United Kingdom.

Côté, M.-A., Suryn, W., Laporte, C. Y., Martin, R. A. (2005). *The Evolution Path for Industrial Software Quality Evaluation Methods Applying ISO/IEC 9126:2001 Quality Model: Example of MITRE's SQAE Method*, Software Quality Journal, vol. 13, 17-30.

Crosby, P.B. (1979). *Quality is free: The art of making quality certain.* New York : McGraw-Hill.

Diaz M. & Sligo, J. (1997). *How Software Process Improvement Helped Motorola*, IEEE Software, 17(5), 75-81.

Dromey, R. G. (1995). *A model for software product quality*. IEEE Transactions on Software Engineering 21, 146-162.

Dromey, R. G. (1996). *Cornering the Chimera*. IEEE Software, 13(1), 33-43.

Eickelman, N. (2003). *An Insider's View of CMM Level 5*, IEEE Software, 20(4), 79-81.

Glass, R.L. (1997). *Software Runaways: Monumental Software Disasters*, Pearson Education POD

Haley, T. J. (1996). *Software Process Improvement at Raytheon*, IEEE Software, 13(6), 33-41.

Highsmith, J. (2002). *Agile Software Development Ecosystems*, Addison-Wesley Professional.

IEEE. 1998. *Std. 1061-1998 IEEE Standard for a Software Quality Metrics Methodology*.

ISO/IEC. 1999a. *ISO/IEC 14598-1: Software product evaluation-Part 1 : General overview*. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 1999b. *ISO/IEC 9000:2000 Quality management systems -- Fundamentals and vocabulary* . Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 2000. *ISO/IEC 15288: System Life Cycle Processes*. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 2001a. *ISO/IEC 9126-1: Software Engineering-Software product quality-Part 1 : Quality model*. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 2001b. *ISO/IEC DTR 9126-4: Software engineering-Software product quality-Part 4: Quality in use metrics*. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 2003a. *ISO/IEC TR 9126-2: Software Engineering-Software product quality-Part 2 : External metrics*. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. 2003b. *ISO/IEC TR 9126-3: Software engineering-Software product quality-Part 3: Internal metrics*. Geneva, Switzerland: International Organization for Standardization.

Kitchenham, S. L., Pfleeger (1996). *Software Quality: The Elusive Target*. IEEE Software, 13(1), 12-21.

Laitinen, M. (2000). *Scaling Down is Hard to Do*, IEEE Software, 17(5), 78-80.

Leffingwell, D. & Widrig, D. (1999). *Managing Software Requirements, A Unified Approach.* Addison-Wesley Professional.

Martin, R. A. & Shaffer, L. (1996). *Providing a framework for effective software quality assessment*. Bedford, Mass : MITRE Corporation.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in software quality*. Griffiths Air Force Base, N.Y. : Rome Air Development Center Air Force Systems Command.

NIST (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing,* [Online] http://www.nist.gov/public_affaires/releases/n02-10.html (Consulted June 3 2004)

Pfleeger, S. L. (2001). *Software Engineering: Theory and practice* (2nd ed.). Upper Saddle River, N.J. : Prentice Hall.

Pressman, R. S. (2001). *Software Engineering: A practitioner's approach* (5th ed.). Boston: McGraw-hill.

Seffah, A. Kececi, N. Donyaee, M. (2001). *QUIM: A Framework for Quantifying Usability Metrics in Software Quality Model, Quality Software*, 2001. Proceedings of the Second Asia-Pacific Conference on, 2001, 311-318.

SEI (2002). *TSP for Secure Systems*, [Online] http://www.sei.cmu.edu/tsp/tsp-secure-presentation/sld001.html (Consulted June 7 2004)

Suryn, W. (2003). *Course notes SYS861*. École de Technologie Supérieure, Montréal.

Voas, J. (2003). *Assuring Software Quality Assurance*. IEEE Software, 20(3), 48-49.