THE ELUSIVE QUEST: SOFTWARE PRODUCT QUALITY EVALUATION

THE ELUSIVE QUEST: SOFTWARE PRODUCT QUALITY EVALUATION

By
Silviya Grigorova, B.A. in Computer Science,
B.A. in Economics

A Thesis
Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree of

M. Sc. in Computer Science
Department of Computing and Software
McMaster University

MASTER OF SCIENCE (2009) McMaster University

(Computer Science) Hamilton, Ontario

TITLE: The Elusive Quest: Software Product Quality Evaluation

AUTHOR: Silviya Grigorova,

B.A. in Computer Science (American University in Bulgaria)

B.A. in Economics (American University in Bulgaria)

SUPERVISOR: Dr. T.S.E. Maibaum

NUMBER OF PAGES: vi, 129

Abstract

Quality has many definitions, and even more models and methods for assurance and evaluation associated with it. After an overview of existing concepts, we provide a comprehensive methodology for evaluating the quality of a software product, complete with methods for model structure and parameter elicitation and a way for mapping the quantitative results obtained from the evaluation to qualitative rankings of product characteristics (e.g. Poor, Good, Excellent). This is complemented with a compendium of quality characteristics and metrics associated with them and a generic quality model combining the works of McCall and Boehm with the international standard for software product quality ISO/IEC 9126-1. A discussion of the advantages that the use of Bayesian Belief Networks (BBNs) can provide in this framework is also offered, as well as a method for transforming quality models to a form suited for BBN use. The methodology has been applied to a case study.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Tom Maibaum for the support and academic guidance. He taught me patience, perseverance and critical judgment, and helped me accomplish what I had set out to do.

I would also like to thank the other members of the examination committee, Dr. Alan Wassyng and Dr. Spencer Smith for the time and consideration they have given to my work, as well as for their helpful comments and suggestions.

I would also like to acknowledge the help of the team of experts that participated in the case study detailed in this thesis and all the people who filled in the questionnaires; their cooperation is greatly appreciated!

Last but not least, I would like to thank my family, and especially my husband Orlin and my daughter Ema for the understanding they have shown when I had to focus on my work and for the unconditional love.

Table of Contents

1. Introduction

The importance of having high-quality software is undeniable – more and more of our day-to-day activities rely on the use of software, whether to conduct business, educate ourselves, or for leisure activities – yet there is not a single widely accepted method for evaluating software quality. Instead, there are different approaches, standards and models, each having its own advantages and disadvantages. The objective of this thesis is to provide a comprehensive methodology for evaluating the quality of a software product, complete with methods for model structure and parameter elicitation and a way for mapping the quantitative results obtained from the evaluation to qualitative rankings of product characteristics (e.g. Poor, Good, Excellent). A discussion of the advantages that the use of Bayesian Belief Networks (BBNs) can provide in this framework is also offered, as well as a method for transforming quality models to a form suited for BBN use. The methodology has been applied to a case study. This is complemented with a compendium of quality characteristics and metrics associated with them and a generic quality model combining the works of McCall and Boehm with the international standard for software product quality ISO/IEC 9126-1 (2001). The main goal of our work is to serve as a knowledge base facilitating the evaluation process.

The thesis is organized as follows:

The second chapter, titled "Background" presents a brief overview of some of the more popular ideas concerning software quality, providing definitions and justification of the importance of software quality evaluation for improving project management. The process-oriented and product-oriented approaches to software quality evaluation and assurance are briefly discussed, and examples of the former are provided.

Chapter 3 ("Models for Software Product Quality Evaluation") offers an in-depth consideration of software quality models. The chapter offers an overview and comparison of the quality models proposed by McCall, Boehm, ISO/IEC 9126, FURPS+ and Dromey, as well as an introduction of the Bayesian Belief Net approach to software quality evaluation.

Chapter 4 ("Towards a Generic Model") presents the creation of a generic model combining the ISO/IEC 9126-1 (2001) external quality model with characteristics from the McCall and Boehm quality models. It also discusses a method for transforming the model for use in BBN software that has been proposed in the literature, and suggests several modifications to the approach.

Chapter 5 ("Customization of the Quality Model, Collecting Metrics and Reading the Results") describes the essence of our methodology. It details the evaluation process from beginning to end, providing ways for eliciting the structure of the software product

quality model, approaches for model parameter elicitation, and alternatives for metric collection, together with ways of interpreting the obtained quantitative results.

Chapter 6 ("Case Study") presents a case study following the outlined methodology. It uses the slightly modified ISO/IEC 9126-1 (2001) model, omitting only the compliance characteristics from the model. Parameter elicitation is performed using the mutual comparison method from the perspective of user, developer and manager, and a discussion of the differences in perception is presented. Metrics collection follows the suggestions of ISO/IEC 9126-2 (2003). The metrics values are transformed to make possible the comparison of quality characteristic values across perspectives and several applications of the obtained results are discussed at the end of the chapter.

Chapter 7 ("Conclusion") provides a summary and concluding remarks, directions for future work and a discussion of the potential of Bayesian Net software in a quality evaluation setting.

Appendix A ("Compendium of Quality Characteristics") provides definitions, specification of the relative position of the quality characteristics in the quality models of which they are a part, and suggested metrics.

Appendix B ("Questionnaire") provides a sample questionnaire for eliciting model parameters, as well as an example of the elicitation technique used in this work.

2. Background

This chapter presents a brief overview of some of the more popular ideas concerning software quality. It provides several definitions of quality, which are examples of the different views that experts hold. Five perspectives of quality – transcendent, product-based, user-based, manufacturing-based and value-based, as defined by Garvin (1984) are discussed. We stress the importance of software quality evaluation as a means for improving project management, and statistics are provided to support that claim. The process-oriented and product-oriented approaches to software quality evaluation and assurance are considered, and as examples of the former we briefly review the Waterfall and Spiral Models, as well as CMM and CMMI. A more in-depth consideration of the product-oriented approach, which was chosen for our methodology, and of software quality models, in particular, is left for Chapter 3: Models for Software Product Quality Evaluation.

2.1. Software Quality Definitions and Perspectives

Most definitions of quality focus either on the degree of conformance to predefined requirements, or on the ability of the product to satisfy customer expectations. (Milicic) Milicic provides as examples definitions proposed by Crosby, Deming, Feigenbaum, Ishikawa, Juran and Shewhart. The definition provided in the international standard ISO/IEC 9126-1 (2001) can be seen as an example of the latter view – "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs". Quality evaluation is defined in the same document as "systematic examination of the extent to which an entity is capable of fulfilling specified requirements", thus taking into consideration both views of quality. Another definition of quality, provided by McCall, Richards and Walters (1977) is "a general term applicable to any trait or characteristic, whether individual or generic, a distinguishing attribute which indicates a degree of excellence or identifies the basic nature of something". This is an example of a definition that does not provide an understanding of the nature of the evaluation process that has to be employed to assess quality. The reason why there is such a divergence in the definitions provided by experts is the difference in the perspectives taken, which are based on competing analyses of the notion of quality, as discussed in the work of David Garvin (1984).

Garvin (1984) distinguishes between five perspectives of quality – transcendent, product-based, user-based, manufacturing-based and value-based. The transcendent perspective was adopted by scholars in the field of philosophy and states that "quality cannot be defined precisely; rather, it is a simple, unanalyzable property that we learn to recognize only through experience." Thus, quality is perceived as an inherent and axiomatic attribute to objects. This approach seems to best reflect the definition provided by

McCall et al. (1977). However, this definition implies that quality cannot be accurately measured and this in our opinion makes it unsuitable for use in a quality evaluation methodology.

According to Garvin, the product-based approach, characteristic to economics, interprets quality as a variable that can be estimated definitively based on the existence or lack of existence of certain measurable product characteristics. It is therefore not influenced by subjective opinions. The user-based perspective is held by scholars from the fields of economics, marketing and operations management. It postulates that quality depends on the individual's perception, and is therefore subjective. In the words of C. D. Edwards, "[q]uality consists of the capacity to satisfy wants..." (Garvin, 1984) There are two problems that arise when considering this view of quality - one is how to consolidate the individual opinions in order to come up with a definition of quality from the perspective of a multitude of users; the other how to distinguish between characteristics that signify quality and characteristics that only improve user satisfaction. If we consider the first definition quoted from the ISO/IEC 9126-1 (2001) standard, it is a blend of the product and user perspective, taking into account the subjective nature of user expectations, and at the same time focusing on attributes of the product which influence the level of satisfaction experienced by the user.

The manufacturing-based definition interprets quality as "conformance to requirements". (Crosby, as taken from Garvin, 1984) According to Garvin, this view is better suited for design and production control, and has an internal focus. This is the perspective that has been used in the second definition quoted from ISO/IEC 9126-1 (2001). It appears that the standard made an attempt at reconciling the different perspectives on quality in order to take advantage of their strong points, as the standard itself offers models for evaluation of internal and external quality, as well as quality in use.

The last perspective described in the paper is the value-based one. It postulates that "[q]uality is the degree of excellence at an acceptable price and the control of variability at an acceptable cost." (Broh, as taken from Garvin, 1984) Neither the price nor the cost of the product has been modeled in the ISO/IEC 9126-1 (2001) standard, nor in McCall's model of software quality. However, ISO/IEC 9126-1 (2001) allows for the inclusion of new quality factors or characteristics in the model, in the case where they are considered to have a significant contribution to the overall quality of the product, and can therefore accommodate the inclusion of this perspective as well. Garvin goes on to say that, according to a survey, the value-based perspective is becoming increasingly more common among users, and even though the product-based view is most used in particular domains, overall, quality is progressively being interpreted with regards to price. According to the author, the value-based perspective is not suitable in practice, as "affordable excellence" is difficult to delimit.

Garvin (1984) claims that "[r]eliance on a single definition of quality is a frequent source of problems." He suggests that the different approaches be used in the different phases of

the software life-cycle. In his opinion, the requirements elicitation needs to be conducted with the user-based perspective in mind, design should take a product-based view, and the development process should follow a manufacturing-based view, to guarantee requirement satisfaction. Following this logic ought to guarantee that the product is acceptable to customers.

2.2. Importance of Software Quality Evaluation

Quality assessment should be considered an important aspect of each phase of the software life-cycle. If the quality evaluation is postponed until late in the software development life-cycle, it is more likely that the estimation of the time it takes to complete the project will be incorrect, because if an inconsistency is discovered this late, it takes a lot longer to amend it, and the cost of doing so goes up. A concise way of expressing the same idea is provided in the words of Tom DeMarco, "You can't control what you can't measure." (DeMarco, 1986) Software quality evaluation is therefore key to producing a high-quality product. Judging by the statistics presented by Boehm and Valerdi (2008) in figure 1, there is a need to improve project performance in terms of time taken till completion and budget, and quality evaluation can prove instrumental for that purpose.

Table 1

The performance of 8,000 projects in 350 organizations[1]

	2000	2002	2004	2006
Percentage of projects delivered within budget and schedule	28	34	29	35
Percentage of projects cancelled before completion	23	15	18	19
Percentage of projects overrun on budget and schedule	49	51	53	46

Figure 1. Boehm and Valerdi's Project Performance Data (2008)

Additional project statistics can be found in the Standish Group's Chaos Report. The Standish Group has been collecting project statistics every two years since 1994. The first such report was published in 1995 and has been widely quoted. It groups projects according to requirements satisfaction and meeting of cost and time till completion forecasts. Interested parties can purchase the report online through the Standish Group's website (https://secure.standishgroup.com/reports/reports.php).

The information provided in the 1995 Chaos Report is analyzed in Barry Boehm's article "Project Termination Doesn't Equal Project Failure". (Boehm, 2000) Barry Boehm discusses the top 10 reasons for cancellation of projects before they are completed and presents his view on whether these projects should indeed be considered failures as claimed in the report, or not. After each reason, a percentage is provided which shows how many of the 31.1% canceled projects in the 1995 Chaos report were attributable to the relevant cause:

- Incomplete Requirements (13.1%) - most of the time caused by lack of understanding of users' needs, but sometimes might be due to conflicting user requirements and inability to reach consensus.
- Lack of user involvement (12.4 %) - inability to communicate with users.
- Lack of resources (10.6 %) - budget cuts, downsizing.
- Unrealistic expectations (9.9 %) - either because no feasibility analysis for requirements satisfaction was performed or because the analysis showed it is impossible to meet user expectations.
- Lack of executive support (9.3 %) - either because the project manager was far too optimistic with regard to executive support, or because of a change of executives.
- Changing requirements (8.7 %) - most are caused by a change of scope not associated with adjustments to budget and time needed, but might also result when the benefits introduced by the change cannot compensate for the higher costs.
- Lack of planning (8.1 %) - because of incompetent project managers.
- Absence of need (7.5 %) - caused by a shift of needs.
- Lack of IT management (6.2 %) - due to inadequate management.
- Technology illiteracy (4.3 %) - referring to either developer or manager.

Boehm contends that "[m]ost of the top sources of termination apply about equally to well- and poorly managed projects", and therefore we cannot always claim that a cancelled project is a failed project. (Boehm, 2000) In his opinion, project termination is acceptable, and even preferable, when things cannot be sufficiently amended, regardless of whether it is the result of unforeseeable changes or not.

Looking at the top reasons for project cancellation, we see that some issues (given in parentheses) can be alleviated with the help of quality modeling and assessment. Discussing explicitly quality requirements increases the common understanding of what attributes the software product has to possess in order to satisfy user needs (Incomplete Requirements). Having an established quality assessment framework might facilitate user involvement, in the cases where it was hampered by a lack of shared understanding and common ground for discussion (Lack of user involvement). Quality assessment is also an important facet of adequate project planning (Lack of planning).

The most recent Standish Group report is from 2009, and Jim Johnson, chairman of The Standish Group, says that "[t]his year's results show a marked decrease in project success rates, with 32% of all projects succeeding which are delivered on time, on budget, with required features and functions". (Lynch, 2009) He goes on to say that "44% were challenged which are late, over budget, and/or with less than the required features and functions and 24% failed which are cancelled prior to completion or delivered and never used." (Lynch, 2009) Jim Crear, Standish Group CIO asserts that these numbers are the worst in the last 10 years, and bear the highest failure rate. (Lynch, 2009) Even though we saw that some of the cancelled projects cannot be labeled failed, a significant portion is never completed because of inadequate management. The increased percentage of cancellations, as well as the decrease in successful projects shows that there is a need to improve project management, and one way to accomplish that is to continuously monitor quality.

2.3. Approaches to Software Quality Assurance and Evaluation

In general, there can be two approaches to developing or selecting a high-quality software product – one can either look at the quality of the development process or, alternatively, at the quality of the product being developed (process-oriented vs. product-oriented quality assurance and evaluation). (Pressman, 2000) It is more intuitive for the user to consider the product-oriented perspective of software quality, while it is more natural for the developer to strive to achieve high quality of the software product by following a process with established procedures. In a way, even though both approaches can be used for both assurance and evaluation, it makes more sense to use the process view for assurance, and the product view for evaluation. As stated in the Software quality analyst article in Wikipedia, "Software Testing is product oriented, Software Quality Assurance is process oriented." (2009) This work is concerned with quality evaluation, and will focus on the product perspective, but we recognize that it is best to use both approaches in conjunction.

2.3.1. Process-oriented Approach

In order to improve software quality, various models for the software life-cycle have been proposed. They represent the process-oriented perspective. Some of the more prominent ones are the Capability Maturity Model (CMM), ISO/IEC 12207, ISO 9001/9000-3, the waterfall model, the spiral model and the SPICE model. Very well-known among them are the CMM and ISO 9001. ISO 9001:2000 is more general in nature and provides clues for designing and implementing a quality assurance system, while CMM gives details for software process improvement. (Xu, Liu, Zhu & Xing, 2005)

According to the definition provided in (Chapman, 2007), "[t]he documented collection of policies, processes and procedures used by a development team or organization to practice software engineering is called its software development methodology (SDM) or system development life cycle (SDLC)". Chapman goes on to state that the best way to approach a methodology is to treat it as a means for risk management. The following excerpt is taken from the same text and lists some major steps of the SDLC:

- Project charter and business case
- Definition of the business process and business requirements
- Documentation of user, functional and system requirements
- Top level architecture, technical approach, and system design
- System decomposition into component and unit specifications and design
- Coding, unit test planning, and unit test
- Generation of test data for unit testing and system testing
- System integration and testing
- Implementation, delivery and cut-over
- Training and user support
- System upgrades and routine software maintenance

These activities can be complemented by the following:

- Configuration management (version identification, baseline management and change control)
- Requirements management and tracability [*sic*]
- Quality management (quality assurance, quality reviews, defect tracking)
- System engineering reviews (requirements review, prelim. and critical design reviews, etc.)
- Support environment (development tools, libraries, files management, data management) (excerpted from Chapman, 2007)

The author elaborates that a methodology provides understanding of how these activities are to be conducted. Since the policies should be described unambiguously and explicitly, it pays to focus on the core processes and not get into too much detail. This also leaves some freedom of choice for the manager and development team.

2.3.1.1. Waterfall Methodology (Chapman, 2007)

The waterfall model illustrates how a hierarchy of steps can help manage a project. (Fig. 2)

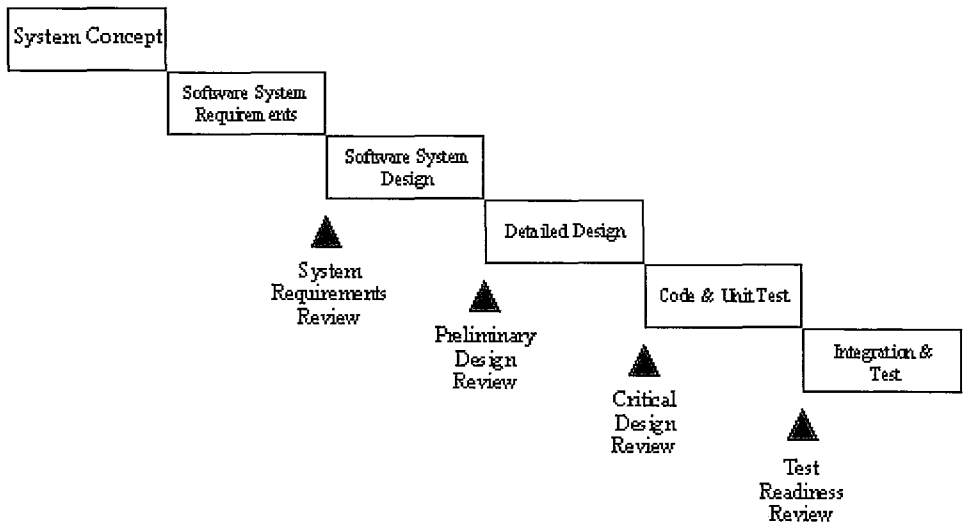


Figure 2. Waterfall Model (Chapman, 2007)

This model assumes that requirements are fairly stable and have already been specified. It was created for defense systems development. Several key aspects of a good methodology are evident here:

- The project follows discrete steps,
- Before a new step is taken, the old one is reviewed, and
- Based on the reviews decisions are made whether to proceed.

The waterfall model establishes the quality, reliability, and maintainability of the software through the reviews conducted in-between steps. Even though this methodology is slow and burdensome, it demonstrates some good principles of SDLC.

2.3.1.2. Spiral Methodology (Chapman, 2007)

The slowness of the waterfall model might make it unfeasible for projects which require quick release. The waterfall model has been modified to accommodate several deliveries or handoffs, and these attempts have culminated in the spiral model. (Fig. 3) The development team can thus begin small, and if there is enough time and resources, go through another iteration, incrementally improving the functionality of the product.

The spiral methodology involves rapid prototyping, increased parallelism, and concurrency in design and build activities. Each phase in the spiral needs to have clearly specified deliverables.

Documentation is very important for the improvement of process and product quality. (Fig. 4) It involves additional overhead and attempts should be made to automate it where possible. The benefits from documenting include, but are not limited to, better

understanding of user's needs, increased maintainability of software and increased reliability. Preparing the documentation should be included in the project scheduling and budgeting.

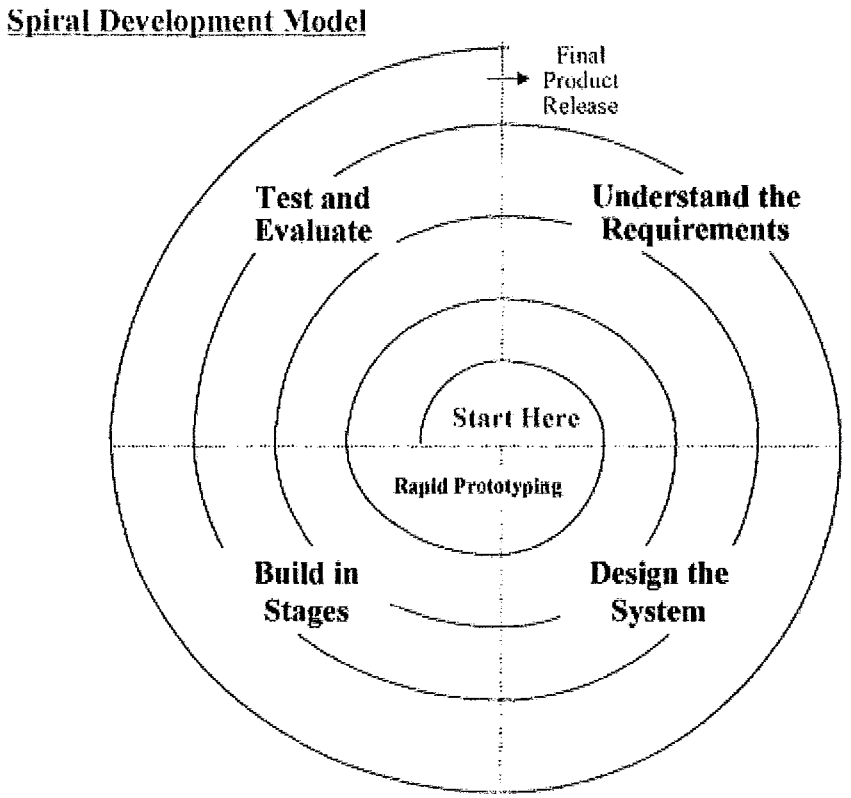


Figure 3. Spiral Model (Chapman, 2007)

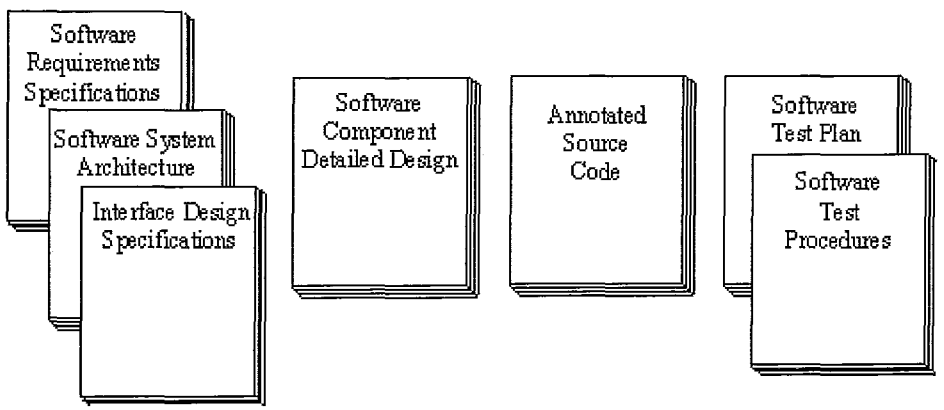


Figure 4. Sample Software Documentation Work Products (Chapman, 2007)

2.3.1.3. Capability Maturity Model (CMM) (Chapman, 2007) **and Capability Maturity Model® Integration (CMMI)** (Carnegie Mellon Software Engineering Institute, 2006)

The Capability Maturity Model (CMM) was originated by the Software Engineering Institute at Carnegie Mellon. It is especially prominent in companies developing large-scale software. CMM distinguishes between five levels of company maturity, and a given company's rating is becoming progressively more important to customers. The following list is excerpted from (Chapman, 2007):

- **Level 1 (Initial)** - Processes are *ad hoc* and occasionally chaotic. Few processes are defined, and success depends on individual effort and heroics. *(A street-person with a laptop would be at Level 1.)*
- **Level 2 (Repeatable)** - Basic project management processes are established to track cost, schedule and functionality. A process discipline is in place to repeat earlier successes on projects with similar applications.
- **Level 3 (Defined)** - Management and engineering processes are documented and integrated into a standard software process. Projects use an approved, tailored version of the organization's standard software process.
- **Level 4 (Managed)** - Detailed measures of the software process and product quality are collected. Processes and products are quantitatively understood and controlled.
- **Level 5 (Optimizing)** - Continuous process improvement is aided by quantitative feedback from the process and from piloting innovative ideas and technologies.

CMM was replaced around 2000 by the SEI Capability Maturity Model® Integration (CMMI). (Chapman, 2007) As pointed out in (Carnegie Mellon Software Engineering Institute, 2006), CMMI was developed as an integration of the following three models:

1. The Capability Maturity Model for Software (SW-CMM)
2. The Systems Engineering Capability Model (SECM)
3. The Integrated Product Development Capability Maturity Model (IPD-CMM)

The goal was to use a single framework for promoting better process practices within the organization. The models to integrate were chosen based on their prominence and their distinctive approach to the problem at hand. (Carnegie Mellon Software Engineering Institute, 2006)

Best practices are collected into so-called constellations, and at the moment there are three constellations – CMMI for Development, CMMI for Services and CMMI for Acquisition. CMMI for Development deals with the development and maintenance of products and services (CMMI for Services deals with the delivery of services). It is used in many domains, e.g. aerospace, banking, computer hardware, software, defense, automobile manufacturing, and telecommunications, etc. It establishes best practices for

project management, process management, systems engineering, hardware engineering, software engineering, etc. (Carnegie Mellon Software Engineering Institute, 2006)

However, CMMI is more general in nature than the SW-CMM, because it covers more domains, and there is no mention of the word "software" in its definitions, making it quite abstract. (Capability Maturity Model Integration, 2009)

Colleen Frye wrote an article titled "CMMI: Good process doesn't always lead to good quality" based on an interview with Bill Curtis, a co-author of CMMI. (Frye, 2008) In the interview, Curtis says "... just because I have a high maturity process doesn't mean I don't have defects. It means I have processes in place. The CMM or CMMI, the successor to CMM, is not a quality standard."(Frye, 2008) In essence, he claims that it is not sufficient to only utilize best practices; in order to guarantee a high-quality product, one needs to be able to evaluate the product's quality, and this is the role of quality models and standards.

In the interview Bill Curtis recollects three waves in the approach to software quality improvement:

- The initial wave was characterized by the use of higher-order programming languages (70-ies)
- The second wave implemented new design methods and took advantage of better tools, e.g. CASE tools (80-ies)
- The third wave introduced a focus on the process (90-ies)
- The new wave is introducing a focus on the product (2000-s) – the quality of its architecture, its cost, its modifiability.

Curtis recognizes that there exist standards providing a structure for quality evaluation as well as metrics to be used, but what he finds lacking is a benchmark that would establish whether a given software product is of poor quality. He believes it would be useful if a group of experts were to compile a list of principles and thresholds against which one can assess and certify a product. Even though we agree that this would be a very useful contribution, we think it would be impossible to come up with a number that is applicable across product lines within a domain, let alone across domains. In this sense we believe it is more reasonable to establish methods for the assessment of these thresholds and apply them for the specific product line within the specific domain as necessary. It is also very important to document thoroughly the decisions made and the process followed for establishing the quality thresholds, thus providing transparency. In this way users of the software product would be aided in their decision-making.

The product and process perspectives of quality assurance and evaluation complement each other and are best used in parallel. We have chosen the product perspective for the focus of this thesis because we want to evaluate the quality of a finished product, and it

cannot be guaranteed by following best practices for product development. Instead, a quality standard needs to be used.

2.3.2. Product-oriented Approach

As quality is defined not only in terms of functional, but also in terms of non-functional properties of the product (the so-called -ilities), in order to successfully model and assess quality one needs to be able to specify both. One popular approach is to use a quality model, defined as "a model which deploys quality into a set of characteristics and shows the relationships between them". (Azuma, 1996) There are alternative approaches, suited for modeling the non-functional aspects of timed execution and those of hierarchical component-based systems (using UML, etc.), which have been summarized by (Colin et al., 2008).

In order to facilitate the cost-benefit analysis in software development, researchers have attempted to model the relationships between the various quality characteristics and the contribution each of them makes to overall quality. There are a number of different uses for a quality model:

- To provide structure for the requirements elicitation process, by explicitly outlining product characteristics influencing quality, and thus facilitating the discussion of what would constitute a quality product
- To set boundary values for quality assurance in all stages of the development process
- To predict the quality of the finished software product based on intermediate measures
- To compare and choose among several products
- To decide whether a software product is ready for release, comparing the results from the evaluation with expected quality
- To determine whether a change in the development process has led to an improvement in the quality of the products developed
- To consider tradeoffs and decide how best to utilize limited resources
- To highlight areas that need improvement
- To decide whether a product's quality is acceptable
- To check requirements conformance

Of course, in order to arrive at valid conclusions and make meaningful inferences, one relies on the accuracy of the model. Quality evaluation is not an exact science, but when the same procedure is followed methodically conclusions can be drawn based on the relative values derived for various products or product characteristics. For example, it is possible to compare the relative quality of one product with another, or with expected quality formulated in the same terms. To check the validity of obtained results, it is very

important to perform sensitivity analyses, so that it can be established whether a minor change in measured values leads to a big change in quality. If the results are sensitive, inferences drawn from the model would be valid only if the quality characteristics have been precisely measured.

2.4. Summary

This chapter presented an overview of software quality concepts. Of the five perspectives of quality described by Garvin (1984), we believe that only the transcendent one can be ignored in the quality evaluation process, as it states that quality cannot be measured precisely. The methodology that we present in this work has a product-based focus, but it is suited to incorporate the user-based, manufacturing-based and value-based perspectives as well, through customizing the hierarchy of the quality evaluation model. The importance of software quality evaluation was demonstrated through the project statistics supplied, as we believe that some of the reasons for project cancellation established by Boehm (2000) could be greatly influenced by quality modeling and evaluation. The discussion of the process-oriented and product-oriented approaches to software quality evaluation and assurance lead us to the decision to focus on the product perspective because we want to evaluate the quality of a finished software product, and it cannot be guaranteed by following best process practices. The next chapter presents several quality models that have gained prominence and discusses their advantages and weaknesses.

3. Models for Software Product Quality Evaluation

The international standard ISO/IEC 9126-1 (2001) defines a quality model as "the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality". This chapter offers an overview and comparison of the quality models proposed by McCall, Boehm, ISO/IEC 9126, FURPS+ and Dromey, as well as an introduction of the Bayesian Belief Net approach to software quality evaluation. A number of different software quality models are presented in the literature, but none of them has been established as an archetype. One of the purposes of this chapter is to establish advantages and disadvantages of the proposed models, in order to make the greatest use of the expert knowledge available in the literature for inferring the model structure to use in our methodology. We believe that models cannot be labeled good or bad, they are only good (or bad) in a particular evaluation scenario, depending on the purpose of the evaluation and on the product being evaluated. Therefore, one key characteristic for a general model of software quality is customizability, and we want to offer that in our methodology, in addition to incorporating the quality characteristics suggested in the models presented in this chapter.

3.1. McCall Model (McCall et al., 1977)

One of the most popular and referenced quality models is the McCall model (Fig. 5), which was created in 1977 to serve the needs of the U.S. Air Force Electronic Systems Division (ESD), the Rome Air Development Center (RADC), and General Electric (GE). It aimed at providing a software quality framework to acquisition managers. This framework was created in order to enable the more precise specification of quality requirements, and to facilitate the timely evaluation of whether those requirements have been met. Up until that time, "the quality desired ha[d] historically been definable only in subjective terms". (McCall et al., 1977) McCall and his colleagues gave a definition of software quality and provided a way of quantifying it through metrics. The model describes software quality as a result of the presence of certain quality attributes in the software product. These attributes, called quality factors, were collected from the literature on software quality evaluation and grouped together in three groups - Product Operation, Product Revision, and Product Transition. They are *user oriented*[1], and are further broken down into measurable software criteria, which are *software oriented*[2].

Let us take the quality factor Efficiency as an example. It is broken down into the measurable characteristics of Execution Efficiency and Storage Efficiency. (Fig. 5) The model offers definitions for all the criteria, e.g.,

[1] The term is used to describe quality characteristics which are of interest to the user, e.g. how fast the program runs (efficiency) or how easy it is to maintain (maintainability).
[2] The term is used to describe attributes of the software or software products of the development process.

- Execution Efficiency – "Those attributes of the software that provide for minimum processing time."
- Storage Efficiency – "Those attributes of the software that provide for minimum storage requirements during operation." (McCall et al., 1977)

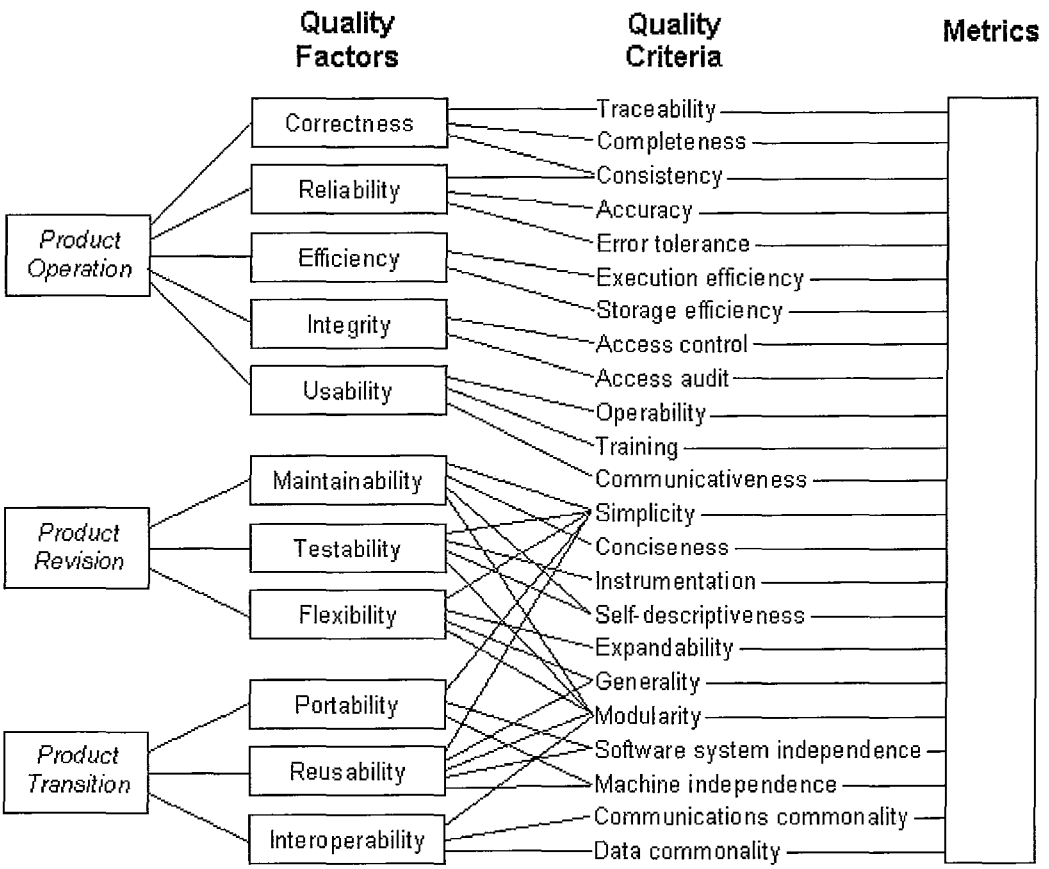


Figure 5. McCall Model (Burris, 2007)

It also provides metrics for the measurement of the criteria and stipulates the phase of the software life-cycle during which they should be collected. The metrics suggested by McCall et al. for our example can be reviewed in table 1. Such metrics were chosen for the quality criteria that would be easy to collect early in the development process, and if possible extracted automatically. The authors recognize that not all of the metrics proposed are objective, even though an attempt was made to minimize subjectivity.

As stated previously, McCall et al. define quality as "a general term applicable to any trait or characteristic, whether individual or generic, a distinguishing attribute which indicates a degree of excellence or identifies the basic nature of something". (McCall et al., 1977) The definition by itself does not provide any insights as to how quality is to be evaluated; it is closer to the transcendent view of quality defined by Garvin (1984) rather

FACTOR(S): EFFICIENCY

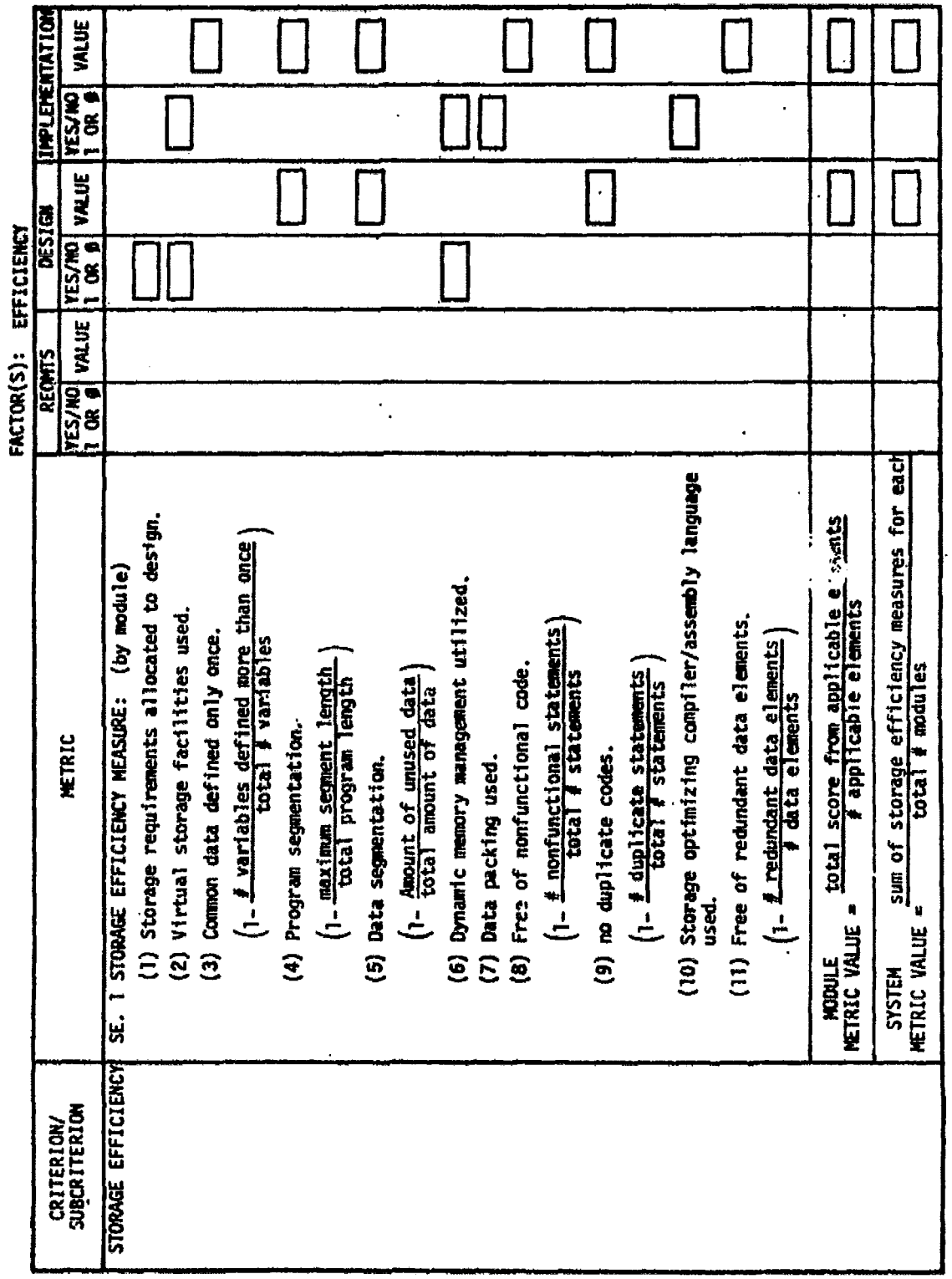
CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR #	VALUE	YES/NO 1 OR #	VALUE	YES/NO 1 OR #	VALUE
EXECUTION EFFICIENCY/ REQUIREMENTS	EE. 1 PERFORMANCE REQUIREMENTS ALLOCATED TO DESIGN			☐			
	SYSTEM METRIC VALUE = Same as line above				☐		
ITERATIVE PROCESSING	EE. 2 ITERATIVE PROCESSING EFFICIENCY MEASURE: (by module)						
	(1) Non-loop dependent computations kept out of loop. $\left(1 - \frac{\#\text{ nonloop dependent statements in loop}}{\text{total \# loop statements}}\right)$				☐		☐
	(2) Performance optimizing compiler/assembly language used.					☐	
	(3) Compound expressions defined once. $\left(1 - \frac{\#\text{ compound expression defined more than once}}{\#\text{ compound expressions}}\right)$						☐
	(4) Number of overlays. $\left(\frac{1}{\#\text{ of overlays}}\right)$				☐		☐
	(5) Free of bit/byte packing/unpacking in loops.			☐		☐	
	(6) Free of nonfunctional executable code. $\left(1 - \frac{\#\text{ nonfunctional executable code}}{\text{total executable statements}}\right)$						☐
	(7) Decision statements efficiently coded. $\left(1 - \frac{\#\text{ inefficient decision statements}}{\text{Total \# decision statements}}\right)$						☐

Table 1. Software Quality Metrics for the Factor Efficiency (McCall et al., 1977)

Table 1. Software Quality Metrics for the Factor Efficiency (Cont.) (McCall et al., 1977)

FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE	YES/NO 1 OR ∅	VALUE
	(8) Module linkages. $\left(1 - \frac{\text{module linkage time}}{\text{execution time}}\right)$ (9) OS linkages. $\left(1 - \frac{\text{OS linkage time}}{\text{execution time}}\right)$						☐ ☐
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{total \# applicable elements}}$				☐		☐
	SYSTEM METRIC VALUE = $\frac{\text{sum of iterative processing measures for each module}}{\text{total \# modules}}$				☐		☐
DATA USACE	EE. 3 DATA USAGE EFFICIENCY MEASURE: (by module) (1) Data grouped for efficient processing. (2) Variables initialized when declared. $\left(\frac{\text{\# initialized when declared}}{\text{total \# variables}}\right)$ (3) No mix-mode expressions. $\left(1 - \frac{\text{\# mix mode expressions}}{\text{\# executable statements}}\right)$ (4) Common choice of units/type. $\left(1/\text{\# occurrences of uncommon unit operations}\right)$ (5) Data indexed or referenced for efficient processing.			☐ ☐	☐	☐ ☐	☐ ☐ ☐ ☐
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\text{\# applicable elements}}$				☐		☐
	SYSTEM METRIC VALUE = $\frac{\text{sum of data usage measures for each element}}{\text{total \# modules}}$				☐		☐



FACTOR(S): EFFICIENCY

CRITERION/ SUBCRITERION	METRIC	REQMTS		DESIGN		IMPLEMENTATION	
		YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE	YES/NO 1 OR Ø	VALUE
STORAGE EFFICIENCY	SE. 1 STORAGE EFFICIENCY MEASURE: (by module) (1) Storage requirements allocated to design. (2) Virtual storage facilities used. (3) Common data defined only once. $\left(1-\frac{\#\text{ variables defined more than once}}{\text{total \# variables}}\right)$ (4) Program segmentation. $\left(1-\frac{\text{maximum segment length}}{\text{total program length}}\right)$ (5) Data segmentation. $\left(1-\frac{\text{Amount of unused data}}{\text{total amount of data}}\right)$ (6) Dynamic memory management utilized. (7) Data packing used. (8) Free of nonfunctional code. $\left(1-\frac{\#\text{ nonfunctional statements}}{\text{total \# statements}}\right)$ (9) no duplicate codes. $\left(1-\frac{\#\text{ duplicate statements}}{\text{total \# statements}}\right)$ (10) Storage optimizing compiler/assembly language used. (11) Free of redundant data elements. $\left(1-\frac{\#\text{ redundant data elements}}{\#\text{ data elements}}\right)$						
	MODULE METRIC VALUE = $\frac{\text{total score from applicable elements}}{\#\text{ applicable elements}}$						
	SYSTEM METRIC VALUE = $\frac{\text{sum of storage efficiency measures for each}}{\text{total \# modules}}$						

Table 1. Software Quality Metrics for the Factor Efficiency (Cont.) (McCall et al., 1977)

than the product-based one. The quality model amends this discrepancy by providing a group of attributes that are responsible for the presence or lack of quality. (Fig. 5) Looking at figure 5, we notice that Functionality has not been explicitly considered, even though most of the characteristics which influence it according to the other models presented in this chapter are also present here (Suitability and Security are examples of characteristics being omitted).

Among the contributions of this model are the following additions to the existing state of the art about software quality evaluation:

- The quality evaluation process is comprehensive, instead of focusing on just one aspect of evaluation
- A method for metrics validation is provided
- The factors are matched with corresponding life-cycle phases
- The proposed metrics are not influenced by the programming language employed
- Such metrics are chosen that can be used early on in development, thus reducing the cost of correcting problems
- The criteria are chosen so as to be as independent as possible
- Some automated metric collection tools are suggested

Another achievement of the model, which is domain-specific, is the linking of quality factors to Air Force applications, measuring factor applicability. This was done as a means for assessing the usability of the framework, but it gives recognition to the varying importance of the different quality factors for the various categories of Air Force applications. In addition, an attempt was made to quantify the correlation of subjective criteria to the quality factors, which is again domain-specific and no general conclusions can be drawn.

Among the disadvantages that we see are the fact that the model itself is static, and cannot be expanded, and that some of the metrics proposed are subjective, even though they would still be useful for the comparison of products by a single company.

3.2. Boehm Model (Boehm et al., 1976)

In his article "Quantiative Evaluation of Software Quality" written together with Brown and Lipow, Barry Boehm provides a framework for software quality analysis. The presented model features higher-level characteristics corresponding to the uses of the evaluation, and lower-level ones that are measurable. (Fig. 6) The work also provides a definition of software metrics which have been grouped and assessed in terms of usefulness (predictive power and potential benefit), quantifiability, and possibility for automation. This model is also fixed.

Boehm et al. consider as one of the great potential advantages that this model has to offer the increased cost-effectiveness of maintenance. It is evident from the figure that there is a focus on Maintainability, and the authors reason that higher quality associated with the Maintainability characteristic would lead to a lower cost incurred during the maintenance phase of the software life cycle. The other characteristics chosen as prerequisites for overall quality (labeled here General Utility) are As-is utility and Portability, reflecting the principal interests of users as the authors see them. Definitions of all quality characteristics used in the model can be found in Appendix A.

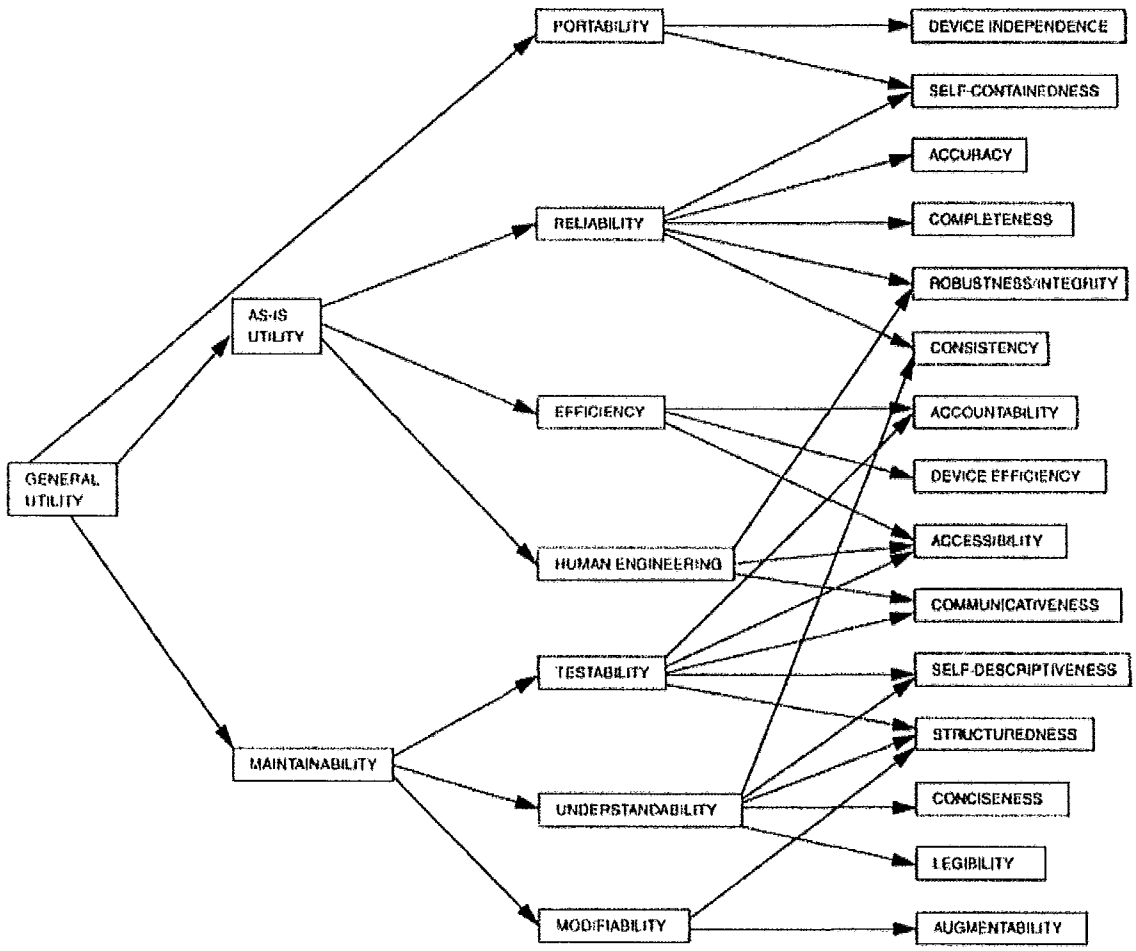


Figure 6. Boehm's model for software quality evaluation (Selby, 2007)

Boehm et al. recognize several key issues involved in software product quality evaluation, namely:

- Users have different needs and requirements, and because of this we cannot apply a single quality metric that would satisfy all users. Useful results can be obtained if the user's priorities can be input into the framework and taken into account.
- Metrics would be best utilized for indicators of problem areas in the development, test planning, acquisition and maintenance phases because "...the metrics are not exhaustive ... [and] the resulting overall rating would be more suggestive than conclusive or prescriptive". (Boehm et al., 1976)

The metrics used in this work are focused on the quality characteristics that the code exhibits, and this is a distinction from McCall's model. They are formulated as questions, and these questions can serve as guidelines for "best programming practices" when developing the product. This makes them useful for both product evaluation and process improvement. Boehm et al. point out that it is not only important to have useful and quantifiable metrics, they should also be easy to collect, so that there is no significant overhead incurred. The following list excerpted from (Boehm et al., 1976) gives definitions of the abbreviations used in the metrics evaluation table (Fig. 7) in order to signify the least expensive approach for quantification:

- AL - can be done cost-effectively via an automated algorithm
- CC - can be done cost-effectively via an automated compliance checker if given a checklist (Code Auditor is such a tool ...)
- UI - requires an untrained inspector
- TI - requires a trained inspector
- EI - requires an expert inspector
- EX- requires program to be executed

Similar rating scales have been provided for the quality metrics' correlation with quality, potential benefit, ease of developing automated evaluation and completeness of the automated evaluation. Even though the list of metrics was compiled to evaluate the quality of Fortran code, and is therefore not generally applicable, the way in which the metrics have been assessed can be utilized in other settings. A formalized comparison of proposed metrics according to the criteria outlined by Boehm et al. would be a valuable addition to any quality evaluation framework, as it will help the evaluator choose amongst alternative metrics. Unfortunately, most authors focus on correlation with quality alone, and leave the metrics comparisons to the evaluator's discretion.

The models proposed by McCall et al. and Boehm et al. have similar structure in the sense that both use a hierarchy of characteristics, in which the lower-level characteristics are prerequisites for the higher-level ones, but there is a difference in the relationships that have been established. For example, in McCall's model both Maintainability and Testability are quality factors, appearing at the same level, and in Boehm's model Testability is a prerequisite for Maintainability. This is a point that is difficult to resolve when constructing a general model incorporating the above-mentioned ones.

Primitive characteristics	Definition of metrics	Correlation with quality	Potential benefit	Quantifiability	Ease of developing automated evaluation	Completeness of automated evaluation
Device Independence DI-1	Are computations independent of computer word size for achievement of required precision or storage scheme?	A	5	AL + EX + TI	E	P
DI-2	Have machine-dependent statements been flagged and commented upon (e.g., those computations that depend upon computer hardware capability for addressing half words, bytes, selected bit patterns, or those that employ extended source language features)?	A	5	AL	M	P
Self-Containedness SC-1	Does the program contain a facility for initializing core storage prior to use?	A	5	AL	E	P
SC-2	Does the program contain a facility for proper positioning of input/output devices prior to use?	A	5	CC	E	P

Figure 7. Evaluation of quality metrics for Boehm's model (Boehm et al., 1976)

3.3. ISO/IEC 9126-1 (ISO/IEC, 2001) and ISO/IEC 14598-1 (ISO/IEC, 1999)

This information has been extracted from ISO/IEC TR 9126-1 and ISO/IEC TR 14598-1 with the permission of Standards Council of Canada, in cooperation with IHS Canada, the official Canadian distributor of ISO publications, License #SCC 08/09 – 035. No further reproduction is permitted without prior written permission

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have produced two standards to deal with the evaluation of software product quality, namely ISO/IEC 9126 and ISO/IEC 14598. The

first standard bears the title ISO/IEC 9126 - Software engineering — Product quality, and consists of four parts:

- Part 1: Quality model (2001) - This part provides models for the evaluation of external quality (Fig. 8), internal quality (Fig. 8), and quality in use (Fig. 9). It builds on the model presented by McCall et al. ("ISO 9126," 2009), and the quality characteristics provided specify both functional and non-functional product attributes. Their definitions can be found in Appendix A.
- Part 2: External metrics (2003) - This part of the standard presents a list of metrics for the evaluation of external quality, which is defined as "the extent to which a product satisfies stated and implied needs when used under specified conditions". (ISO/IEC, 2001) According to the information on ISO 9126 in Wikipedia, external metrics are "applicable to running software". ("ISO 9126," 2009)
- Part 3: Internal metrics (2003) - This part of the standard presents a list of metrics for the evaluation of internal quality, which is defined as " the totality of attributes of a product that determine its ability to satisfy stated and implied needs when used under specified conditions". (ISO/IEC, 2001) Wikipedia describes internal metrics as "those which do not rely on software execution (static measures)". ("ISO 9126," 2009)
- Part 4: Quality in use metrics (2004) - This part of the standard presents a list of metrics for the evaluation of quality in use, which is defined as " the extent to which a product used by specified users meets their needs to achieve specified goals with effectiveness, productivity and satisfaction in specified contexts of use". (ISO/IEC, 2001) According to Wikipedia, quality in use metrics are "only available when the final product is used in real conditions". ("ISO 9126," 2009)

In addition to considering quality from three different perspectives, the standard provides in its first part an understanding of how the different perspectives are related. (Fig. 10) Ideally, internal quality determines external quality, and external quality in turn determines quality in use. This makes it possible to use the information obtained through one of the perspectives to infer a value for another perspective, and the predictive power is improved with a better understanding of the dependency links between them.

The second standard is titled ISO/IEC 14598 - Information technology — Software product evaluation, and its name reveals its purpose – to standardize the evaluation process by providing structure and describing deliverables. It consists of 6 parts:

- Part 1: General overview
- Part 2: Planning and management
- Part 3: Process for developers
- Part 4: Process for acquirers
- Part 5: Process for evaluators
- Part 6: Documentation of evaluation modules

and reveals some potential uses of the evaluation process of intermediate and final product quality. The following lists have been excerpted from (ISO/IEC, 1999):

The purpose of evaluation of intermediate product quality may be to:

- decide on the acceptance of an intermediate product from a subcontractor;
- decide on the completion of a process and when to send products to the next process;
- predict or estimate end product quality;
- collect information on intermediate products in order to control and manage the process.

The purpose of evaluation of end product quality may be to:

- decide on the acceptance of the product;
- decide when to release the product;
- compare the product with competitive products;
- select a product from among alternative products;
- assess both positive and negative effect of a product when it is used;
- decide when to enhance or replace the product.

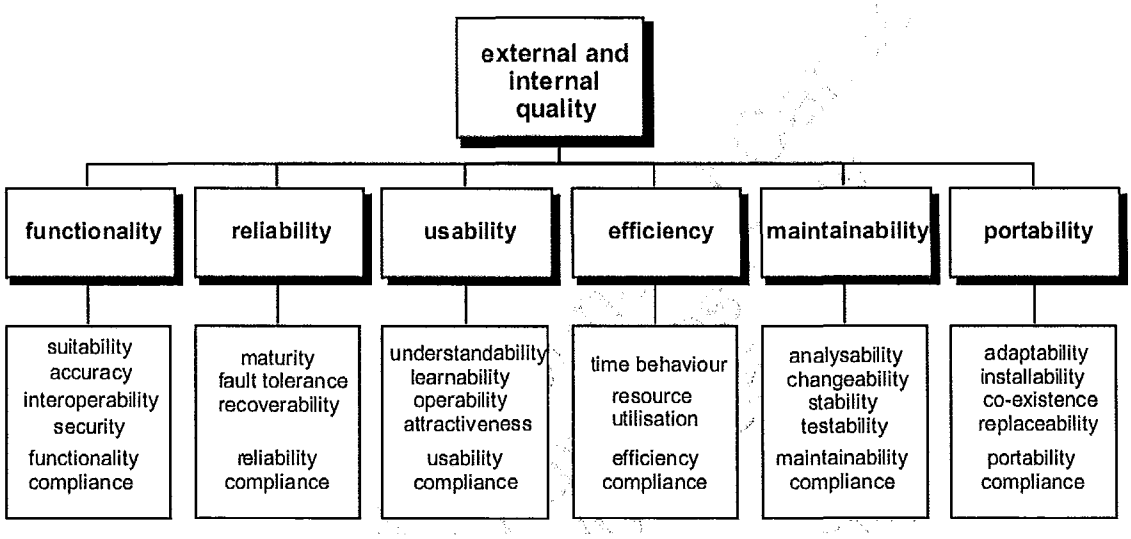


Figure 8. Quality model for external and internal quality according to ISO/IEC 9126-1 (ISO/IEC, 2001)

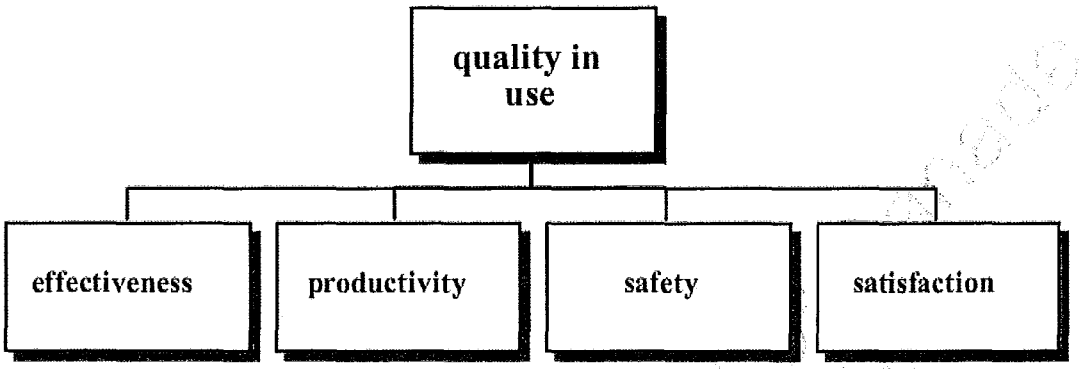


Figure 9. Quality model for quality in use according to ISO/IEC 9126-1 (ISO/IEC, 2001)

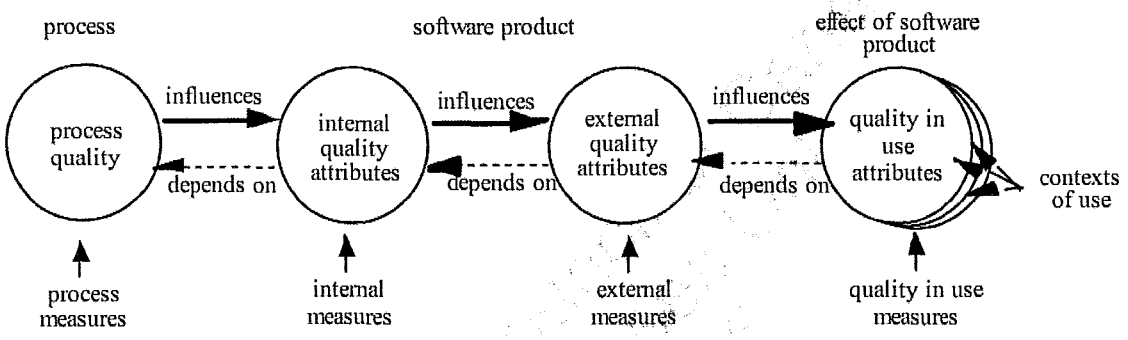


Figure 10. Quality in the lifecycle (ISO/IEC, 2001)

Other uses for the evaluation include the revision and enhancement of the production process and the deriving of links between the various metrics (internal, external and metrics in use), which can subsequently be used for quality prediction. Quality characteristics that need improvement can be better targeted as a result of the evaluation. This standard recognizes that the different perspectives that users, developers and managers have will be reflected in potentially different rating levels (their understanding of what values correspond to a "good" product might differ).

The ISO/IEC 14598 standard was created to facilitate the quality evaluation process by providing a framework for the use of ISO/IEC 9126. (Fig. 11)

The figure provides a systematic approach to the evaluation process that we have attempted to follow as much as possible in the case study presented in a subsequent chapter of this thesis. However, the ISO/IEC 14598 standard we use was released prior to the newer ISO/IEC 9126 metrics standards and does not match them. This was one of the

reasons why we consider it more of a general point of reference and observe only the relevant points it has to offer. For example, the standard suggests that the purpose of evaluation should be defined first. (Fig. 11) This is meaningful, because the purpose of evaluation will influence the decision about which quality characteristics to include and which to omit in the quality model. The type of product and its domain also play an important role in customizing the model, and are naturally considered as a second step of establishing the evaluation requirements. The next steps delineated in figure 11 are also meaningful regardless of the standards and models chosen and the metrics used.

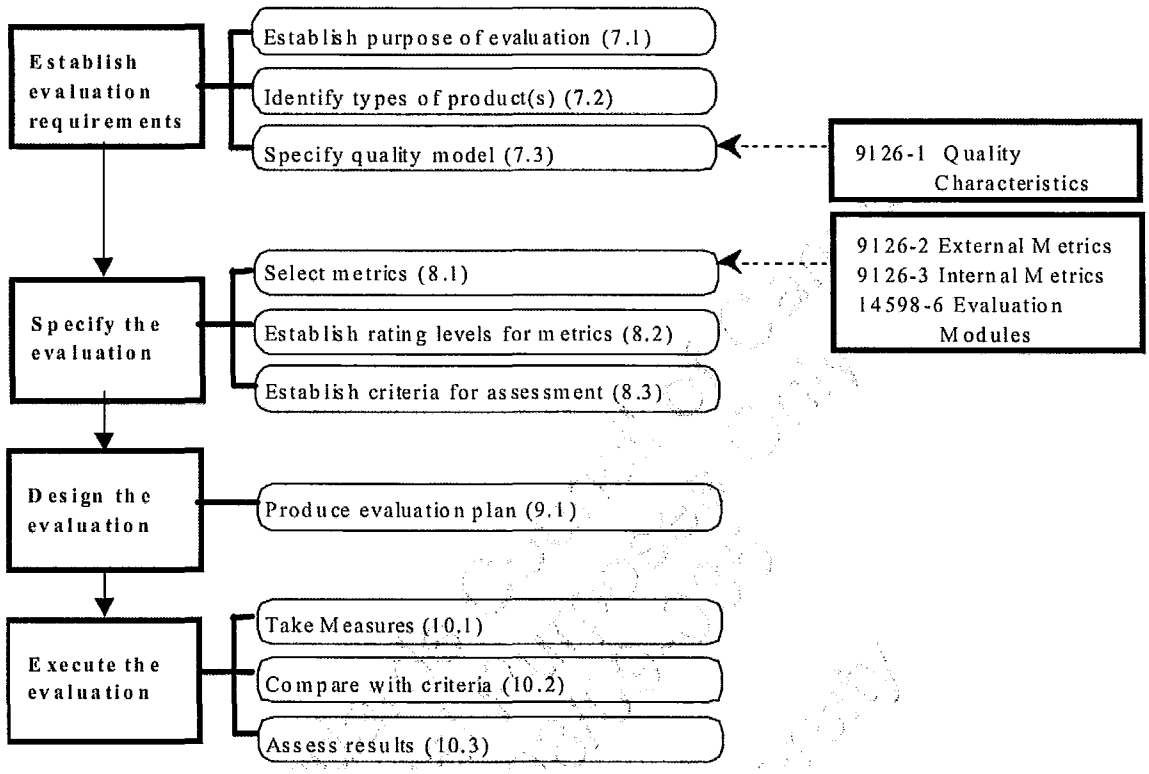


Figure 11. Software Quality Evaluation Process of ISO/IEC 14598-1 (ISO/IEC, 1999)

Some of the positive aspects of the ISO/IEC 9126 standard as discussed in (Burris, 2007) are:

- It is an international standard, therefore universally recognized and continuously improved.
- Because of its prominence, it provides a shared understanding of software product quality and thus facilitates comparison.
- It measures quality from multiple perspectives (internal, external, in use).
- The model is complete and comprehensive.
- The standard (model and metrics) is general enough to apply across software application domains and with regard to different implementation technologies,

and at the same time it provides customization options to make it specific enough for the tasks at hand.

- The standard does not claim to provide means of measuring absolute quality, as every quality characteristic in the model can vary in importance according to application domain, and therefore carry a different weight in the quality evaluation of different products. As an example, Reliability does not have the same importance in the quality evaluation of a pacemaker and a minesweeper game.
- The standard complements the ISO process quality standards.

Intended advantages mentioned in other sources, which we believe are present include:

- Quality evaluation becomes reproducible. (Punter, Solingen, & Trienekens, 1997)
- The standard facilitates the communication between software stakeholders by providing a common language. This in turn improves the quality of the requirements specification. (Al-Kilidar, Cox, & Kitchenham, 2005)

We believe that the best feature the standard possesses is its flexibility and the potential to be customized, as this is of particular importance when trying to arrive to a close approximation of something as elusive as quality. The standard provides structure for the evaluation process and gives many useful suggestions, without being overly prescriptive. The model itself is open for additions as well as for removing some of the recommended quality characteristics, as long as all the choices are well documented, "... giving the reasons for any exclusions, or describ[ing] its own categorisation of software product quality attributes and provid[ing] a mapping to the characteristics and subcharacteristics ... [of ISO/IEC 9126-1]". (ISO/IEC, 2001) Another example of the flexibility of the standard is the availability of choice of metrics for the evaluation. The list of metrics is not considered exhaustive, and evaluators are free to modify the suggested metrics or use different ones. (ISO/IEC, 2003) Some of the metrics have not been validated; however, there is an annex section in Annex A of ISO/IEC 9126-2 dedicated to metric validation, which is yet another useful aspect of the standard. (ISO/IEC, 2003)

Internal metrics refer to inherent product attributes and can be obtained earlier in the development life-cycle, measuring the quality of intermediate products. (ISO/IEC, 2001) Quality in use metrics are collected when the product is used in its intended environment by real users. This distinguishes them from the external metrics, which are collected in a simulated environment. (ISO/IEC, 2001) Quality in use metrics are suggested as a means of measuring the quality from the user's perspective and focus on the characteristics most important to the user (i.e., Effectiveness, Productivity, Safety and Satisfaction), excluding others which are less relevant. It seems as if the Satisfaction characteristic is included as a generalization corresponding to all quality characteristics which are not covered by the other three. However, since Satisfaction is defined in (ISO/IEC, 2001) as "[t]he capability of the software product to satisfy users in a specified context of use...", it overlaps with the other quality characteristics, as their prerequisites are necessarily prerequisites for user satisfaction. In this sense it appears closer to the

definition provided for quality in use, "[t]he capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in specified contexts of use". (ISO/IEC, 2001) The standard also specifies that quality in use "... may be influenced by any of the quality characteristics ..." thus supporting our understanding of the all-encompassing scope of quality in use. (ISO/IEC, 2001) In our understanding, Satisfaction is also potentially dependent on any and every quality characteristic, thus a disambiguation clause should be incorporated in the standard to avoid confusion. A good model minimizes the overlap of quality characteristics, and we believe that there needs to be an adjustment in the definitions supplied in the standard. A further discussion of this perceived discrepancy is outside the scope of this work.

In the discussion of the ISO/IEC 9126 standard in (Al-Kilidar, Cox, & Kitchenham, 2005) some disadvantages have been pointed out. Even though the work of Al-Kilidar et al. is concerned with the evaluation of the outputs of the design phase of the software life-cycle and uses internal metrics, we are going to briefly review some of the comments here, as they are general enough to apply to the standard as a whole. The authors were faced with two major problems: the students who participated in the experiment found the ISO/IEC 9126 terms unclear, and could not easily grasp what usability incorporates. The authors go on to add that the standard is lacking a glossary of terms, which is necessary in order to prevent alternative interpretation of concepts. The version of the standard that we consider here is newer, and it incorporates definitions of the important terms, so this can be seen as an example of the continuous improvement of the standard (the paper refers to Parts 2-4 of the standard from 2002, and we consider ISO/IEC TR 9126-2:2003, ISO/IEC TR 9126-3:2003, ISO/IEC TR 9126-4:2004). However, as we have seen above, some definitions need further work.

Another point made in (Al-Kilidar et al., 2005) is that the standard is open to subjective interpretation, and this makes it less of a standard. It is true that having the possibility to customize the model, interpret the suggested metrics in more than one way and use alternative metrics may hamper comparisons across companies, or even product lines within the same company, but this is a necessary feature of any standard which is to be useful. As the authors themselves point out in the words of M. A. Jackson, "the generality of a method is inversely proportional to its utility." (Al-Kilidar et al., 2005) The standard provides general guidelines and leaves room for adjustments to be made to meet project specifics. As long as the metrics used can be validated (and this means that they are reproducible and repeatable, among other things), the evaluator can choose the ones most suitable for the objectives of the product evaluation. (ISO/IEC, 2003) Al-Kilidar et al. state that the generality of the metrics definitions makes some of them overlap, depending on the interpretation. Since one of the goals of the standard is "[t]o describe the product quality with a minimum of overlap", we believe that this idea should be pursued throughout the evaluation process. (ISO/IEC, 2001) It is therefore the responsibility of the evaluator to choose metrics and interpretation of metric definitions in a way that minimizes overlapping. When the evaluation process has been appropriately documented, and all subjective decisions have been committed to paper and justified, there is sufficient

information to enable comparisons of products based on overall quality or on any given quality characteristic, as any subsequent evaluation could access this documentation. If the products are diverse enough to require different modeling and measuring of quality, then there is no better way to compare them to begin with, as it would be like comparing apples and oranges. In this case, comparisons can only be made of the matching parts of the quality model. Documenting the decisions also helps if there is a need to review and modify them after additional information comes to light. For example, some requirements which have not been written down, but were anticipated by users as part of an implicit agreement, might be added later on.

Another criticism is that the metrics suggested in the standard might not be the best ones. The authors quote Einstein's words that "not everything that can be counted counts, and not everything that counts can be counted". (Al-Kilidar et al., 2005) Counts were suggested in the standard in an attempt to decrease subjectivity and provide metrics applicable to all product domains. Their generality is not a serious hurdle, as the evaluator is given freedom in choosing which metrics to use in the evaluation, and is not limited to the suggested ones, which "are not intended to be an exhaustive set". (ISO/IEC, 2003)

3.4. Dromey's Model (Dromey, 1996)

Dromey describes quality as "experiential", determined by a person's needs, thus taking the user-based view of quality. He believes that a process approach alone cannot guarantee product quality and his model focuses on the quality of the products of the software life-cycle phases. There are in effect separate quality models for the requirements, design and implementation phase. Dromey attempts to express the relationship between product attributes and high-level quality characteristics, motivated by the axiom that "*a product's tangible internal characteristics or properties determine its external quality attributes*"[3]. (Dromey, 1996) The same idea is reflected in the relationship of internal and external quality in ISO/IEC 9126. In Dromey's opinion, establishing the links between the two can serve as a guideline for improving the process by means of providing a clear objective. He believes that product quality models are necessary for the efficient adjustment of process practices.

In Dromey's approach, the model is built from the bottom-up, starting with the measurable attributes of the product and proceeding to the higher-level quality characteristics that they influence. Since products are comprised of components, their quality depends on the quality-carrying attributes of the components they contain, and the quality-carrying attributes of the components' compositions. The model groups the quality-influencing component properties into Correctness properties, Internal properties, Contextual properties and Descriptive properties. The high-level quality characteristics of the model are chosen based on their relative importance for the evaluated product and the

[3] The excerpt was italicized in the original document.

current project. Because Dromey's work advocates the synergy of the process- and product-oriented approaches to software quality evaluation, "Process-mature" is included as one of the high-level quality characteristics in each of the proposed quality models. (Fig. 12-14) The dependencies between measurable properties and high-level quality attributes are inferred based on the relationships established by Dromey between the four component property groups and the latter. As an example, the Correctness properties have been linked to Functionality and Reliability. (Fig. 15)

IMPLEMENTATION: HIGH-LEVEL QUALITY ATTRIBUTES	
Attributes	Subattributes
Functionality	Suitability, accuracy, interoperability, compliance security
Reliability	Maturity, fault-tolerance, recoverability
Efficiency	Time behavior, resource behavior
Usability	Understandability, learnability, operability
Maintainability	Analyzability, changeability, stability, testability
Portability	Adaptability, installability, conformance, replaceability
Reusability	Machine-independent, separable, configurable
Process-mature	Client-oriented, well-defined, assured, effective

Figure 12. High-level quality attributes for the implementation model (Dromey, 1996)

REQUIREMENTS: HIGH-LEVEL QUALITY ATTRIBUTES	
Attributes	Subattributes
Accurate	Conformant, functional, valid, constrained
Understandable	Motivated, coherent, self-contained
Implementable	Achievable, viable, total solution
Adaptable	Modifiable, extensible, reusable
Process-mature	Client-oriented, well-defined, assured, effective

Figure 13. High-level quality attributes for the requirements model (Dromey, 1996)

Dromey proposes a framework for model construction with several steps:

- Choose the high-level quality characteristics with highest priority for the particular product;
- Make a list of the product components;
- Choose the most relevant measurable attributes of each component and assign them to one of the property groups;
- Establish dependencies between the property groups and quality objectives;
- Assess the model and improve it.

DESIGN: HIGH-LEVEL QUALITY ATTRIBUTES	
Attributes	Subattributes
Accurate	Conformant, functional, valid, constrained
Effective	Resource-efficient, rational
Understandable	Motivated, coherent, self-contained
Adaptable	Modifiable, extensible, reusable
Process-mature	Effective, well-defined, assured

Figure 14. High-level quality attributes for the design model (Dromey, 1996)

This is a generic approach that can be utilized for the quality models associated with each phase of the software life-cycle. Dromey follows the framework to describe the way he arrived at quality models for requirements, design and implementation. The model that corresponds to the product quality models we have reviewed above is the Implementation quality model. Its top level expands on the top level of ISO 9126 by adding Reusability and Process-mature. (Fig.12) The list of implementation components can be found in (Dromey, 1994), and in essence, it comprises the constructs of a programming language. They are outlined by the language's grammar and can be grouped into two categories, one describing computations (loops, if-statements, assignments, etc.) and one describing data (e.g. variables, constants). Figure 16 illustrates the impact that the properties of the component Expression have on quality. The dependencies between component properties and quality attributes are represented in figure 15. Dromey chose the links minimizing overlapping, and recognizes that the set is not complete.

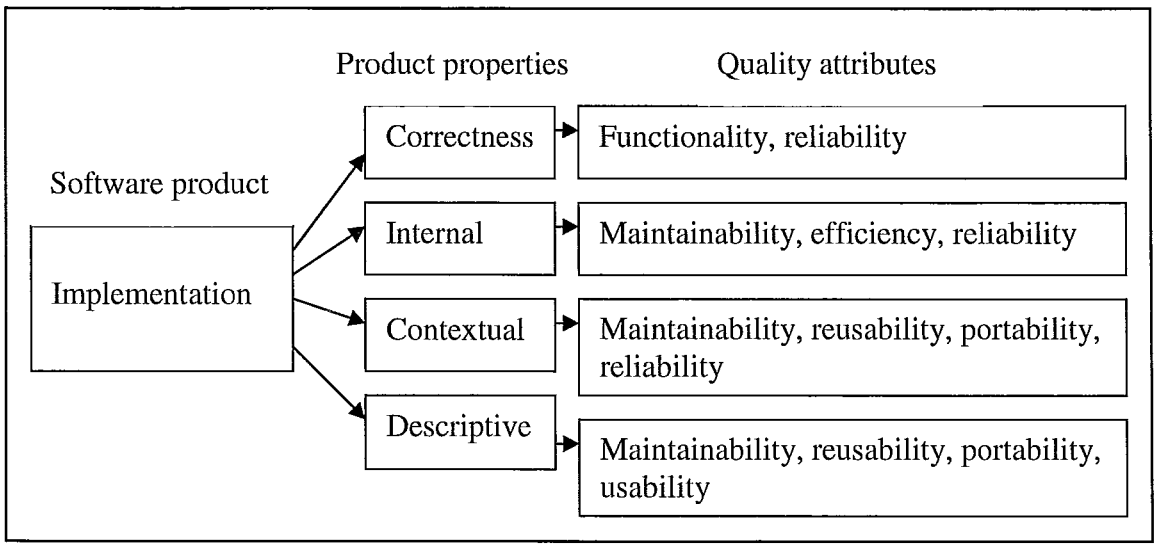


Figure 15. Links between product properties and quality attributes (adapted from Dromey, 1996)

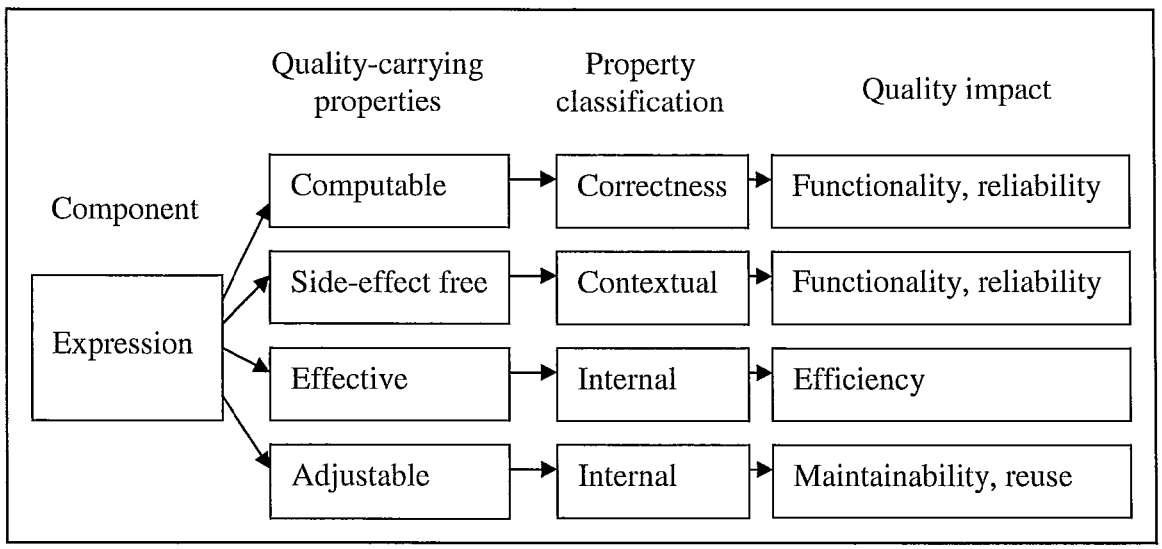


Figure 16. Impact of the Expression component on quality (adapted from Dromey, 1996)

A positive aspect to the model is the constant reevaluation and improvement that the framework implies. For example, additional quality-influencing attributes can be added as deemed necessary. The model is flexible and provides for customization. Another positive aspect is the linking of the process and product approach to quality evaluation. A negative aspect is the lack of metrics suggested for the measurable product attributes. The model is closer to suggesting a checklist of properties that should be present than a means for evaluating quality, and it does not discuss the varying importance of the quality characteristics included in the model depending on user preference. Another point that

should be duly noted is that the term "product" in the Implementation model setting refers to the software program, thus taking a more limited view of a software product, omitting attributes such as documentation. This is why the product components are represented by the constructs of the programming language used. On the other hand, ISO/IEC 9126 considers the software product in a broader sense, and recognizes its interaction with its environment and the user, facilitating more applications. (Fig. 17) This makes it better suited for use as a basis for our methodology.

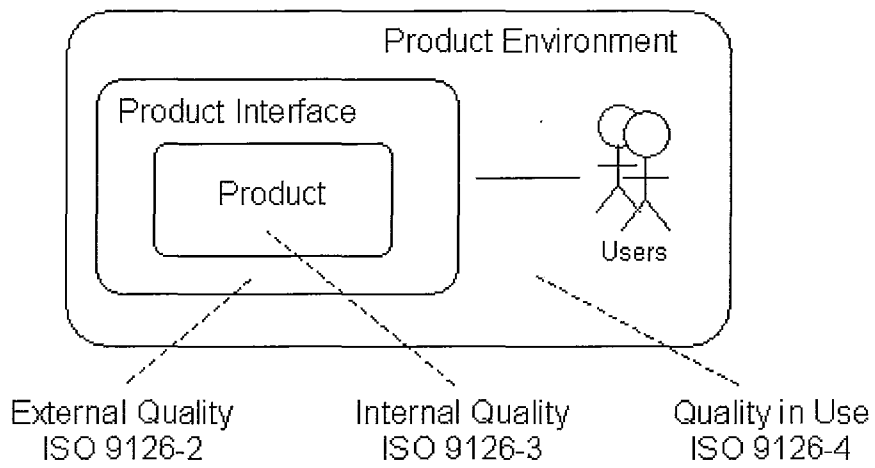


Figure 17. Quality perspectives of ISO/IEC 9126 (Burris, 2007)

3.5. FURPS (Grady & Caswell, 1987)

Another hierarchical quality model is Hewlett-Packard's FURPS, promoted by Robert Grady and Deborah Caswell. It focuses on both functional and non-functional quality characteristics. The acronym is expanded as follows:

F - Functionality
U - Usability
R - Reliability
P - Performance
S – Supportability

A disadvantage of this model is that no attention is given to Portability. Each characteristic is further decomposed into the following components:

- Functionality - Feature set, Capabilities, Generality, Security
- Usability - Human factors, Aesthetics, Consistency, Documentation
- Reliability - Frequency/severity of failure, Recoverability, Predictability, Accuracy, Mean time to failure

- Performance - Speed, Efficiency, Resource consumption, Thruput, Response time
- Supportability - Testability, Extensibility, Adaptability, Maintainability, Compatibility, Configurability, Serviceability, Installability, Localizability

Aesthetics, Documentation, Serviceability and Localizability are some of the quality characteristics introduced by the model which are not present in the models discussed above. This model takes explicitly into account the tradeoffs that exist for satisfying the quality characteristics. The priority of each component needs to be determined, and a way to measure it established. One of the advantages of the model is taking into account the specific objectives that a project might have (i.e., the emphasis can be on different quality characteristics depending on the product domain). After the priorities have been established, they help in selecting the goals for each component. (Fig. 18) In the figure, examples of measurable objectives for the respective phases of the software life cycle are presented. This gives structure to the evaluation process, by suggesting a metric to be collected during each phase. However, the exact objectives to be set can only be determined after consulting domain experts and available data.

The model has been extended to FURPS+ in order to accommodate design, implementation, interface and physical constraints. (Eeles, 2005)

3.6. Bayesian Belief Networks (BBNs)

Charles River Analytics (CRA) (2008) provide a concise description of BBNs in their short introduction titled "About Bayesian Belief Networks", summarized in the next three paragraphs. Bayesian Nets represent a modeling framework for causal relations that can be applied to numerous domains and problems. They take advantage of past information about the relationships between variables and encode it in conditional probability tables (CPTs), associated with each variable. BBNs are directed acyclic graphs (DAGs) (Nicholson & Korb, 2006) where variables are modeled as nodes, and causal relationships are modeled as edges connecting the nodes. An arrow shows the direction of influence. (Fig. 19) Each node has states, and can be either discrete or continuous (Nicholson & Korb, 2006). BBNs are especially useful when the information in the present is incomplete or vague. They can facilitate or even automate the decision-making process, thus acting as a decision support system.

Bayesian Belief Networks can be used for both inductive and deductive reasoning - inferring a cause from an effect and forecasting an effect given a cause. The probability of a node being in a given state depends on the states of other nodes, and is determined by causal relationships. CPTs hold this prior information. They can be represented in table form, where each column gives the probability of the node being in a certain state based on the combination of states of the influencing nodes. (Fig.20)

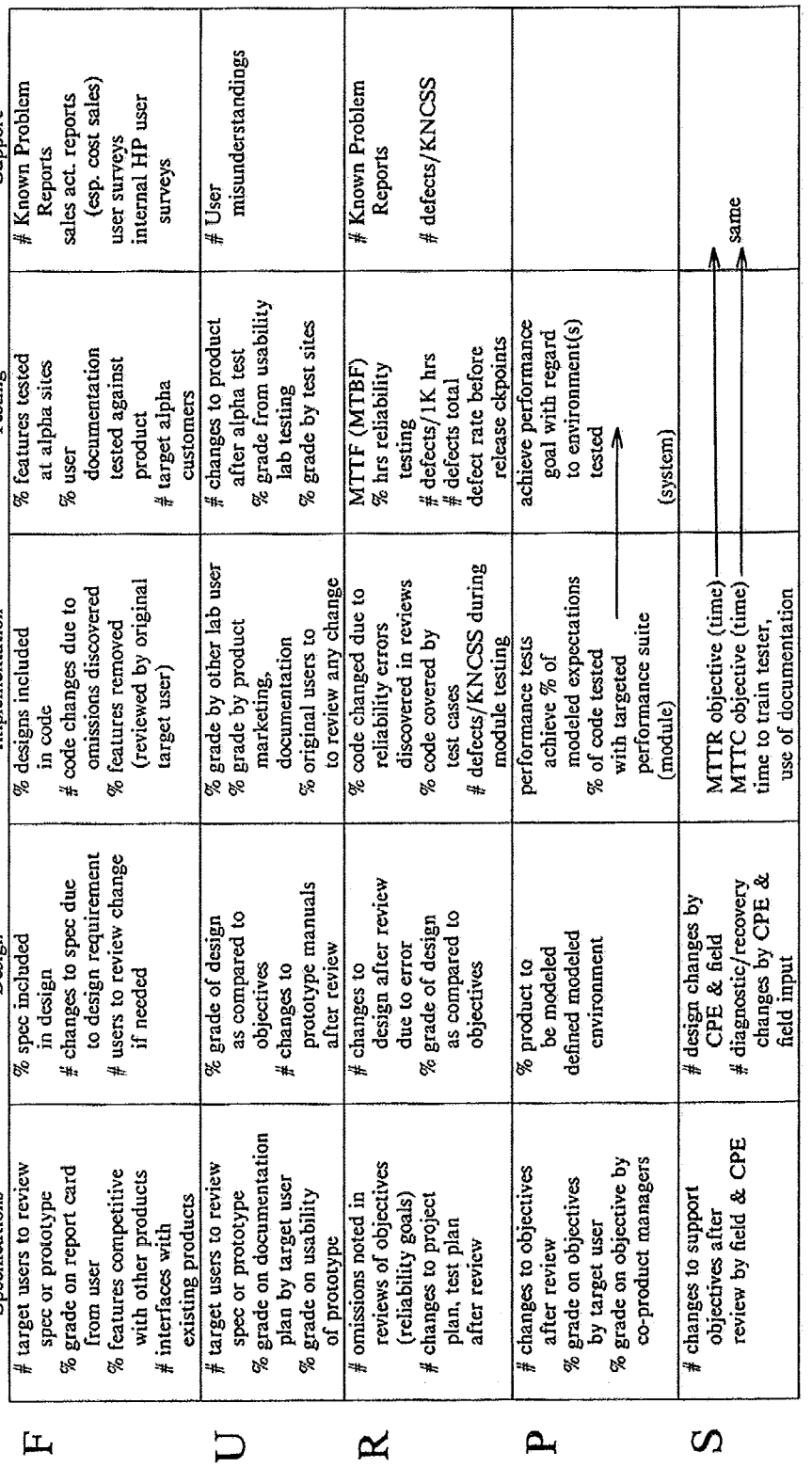


Figure 18. Measurable goals for the software life cycle phases (Grady & Caswell, 1987)

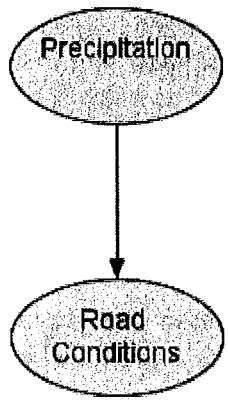


Figure 19. A simple Bayesian Network (CRA, 2008)

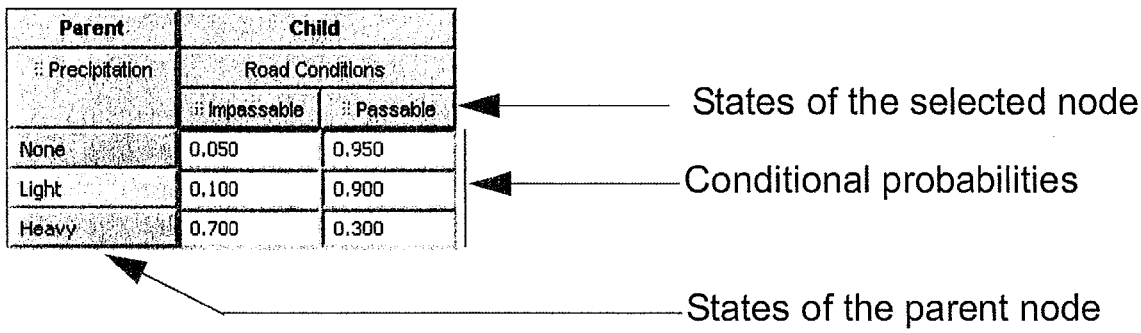


Figure 20. Road Conditions CPT (influenced directly only by precipitation) (CRA, 2008)

Two important concepts for BBNs are beliefs and evidence. Evidence is information about the present state of things, while beliefs are "the probability that a variable will be in a certain state based on the addition of evidence in a current situation". (CRA, 2008) Therefore, evidence modifies beliefs. There are two kinds of evidence - hard and soft. Hard evidence refers to information that a node is "100% in one state, and 0% in all other states". (CRA, 2008) The alternative is soft evidence, which allows for uncertain or even conflicting information to be entered in the model. This is accomplished by entering a probability for a variable being in each state. A-priori beliefs are dependent only on the data in the CPTs, reflecting the beliefs before any evidence is introduced in the model. (CRA, 2008)

Bayesian Belief Networks do not suggest a model, but rather present a way of modeling. The quality characteristics and overall quality can be modeled as nodes, and the relationships between them would be represented as edges. If enough data is available, the direction and strength of influences between the nodes can be inferred and encoded in the CPTs. Some of the available BBN software solutions provide structure and parameter learning. Structure learning refers to the elicitation of the model structure - the

positioning of edges and arrows is determined based on the available past data. Parameter learning refers to the elicitation of the strength of influences between the nodes, again inferred from past data. It is then encoded as a-priori beliefs in the CPTs. The characteristics of the different software solutions can be found in the list compiled and updated by Kevin Murphy (Murphy, 2009), and more information on parameter and structural learning can be found in (Goldenberg). A number of interesting links are also available on the Wikipedia page ("Bayesian Network", 2009).

Ann Nicholson and Kevin Korb point out in their Bayesian AI Tutorial that another important aspect of Bayesian Net models, which we believe is sometimes overlooked, is the so-called Markov property. (Nicholson & Korb, 2006) It states that all direct dependencies between nodes are explicitly modeled through edges. This is important for the correct computation of probabilities, and therefore for obtaining meaningful results. The task of determining which nodes are directly connected can be assisted by using Matilda, which is a tool for visual exploration of dependencies. (Nicholson & Korb, 2006)

The authors also discuss some extensions to Bayesian Nets, such as Bayesian Decision Networks, and Dynamic Bayesian Networks. The former are well-suited for assisting in the decision making process, through introducing decision nodes and utility nodes. They are also known as Influence diagrams. The latter assist in evaluating changes that occur with time. Bayesian Decision Networks can be used to facilitate decisions such as whether to release a product or not. If the utilities associated with releasing (or not) a product of any given quality (poor, satisfactory, average, good, excellent) are known, the software would provide the optimal decision given the available evidence. A wealth of information on Bayesian Nets, including a list of references related to learning Bayesian Networks, BN Knowledge Engineering, BN applications etc. can be found in (Nicholson & Korb, 2006).

Examples for the use of Bayesian Net models listed in the Bayesian AI Tutorial include weather forecasting (fogs, hailstorms, ...), medical diagnosis, judicial decisions, hiring decisions, etc. The following is a list of potential BBN uses excerpted from (Nicholson & Korb, 2006):

- **Decision making:** Which policy carries the least risk of failure?
- **Forward Prediction:** Hypothetical or factual. Who will win the election?
- **Retrodiction/Diagnosis:** Which illness do these symptoms indicate?
- **Monitoring/control:** Do containment rods needs [*sic*] to be inserted here at Chernobal [*sic*]?
- **Explanation:** Why did the patient die? Which cause exerts the greater influence?
- **Sensitivity Analysis:** What range of probs/utilities make no difference to X?
- **Information value:** What's the differential utility for changing precision of X to e?

This is accomplished through the inference process – probabilities for query variables are estimated based on the known values of evidence variables, using Bayes's rule. (Fig.21) When evidence for a given node becomes available, the beliefs of all other nodes are automatically adjusted to reflect this new information. The conditional independence implied by the Markov property supports efficient updating. As we have stated before, conditional independence is encoded in the following way: edges between the nodes express the qualitative dependency between the variables, while CPTs represent the quantitative dependencies. (Nicholson & Korb, 2006)

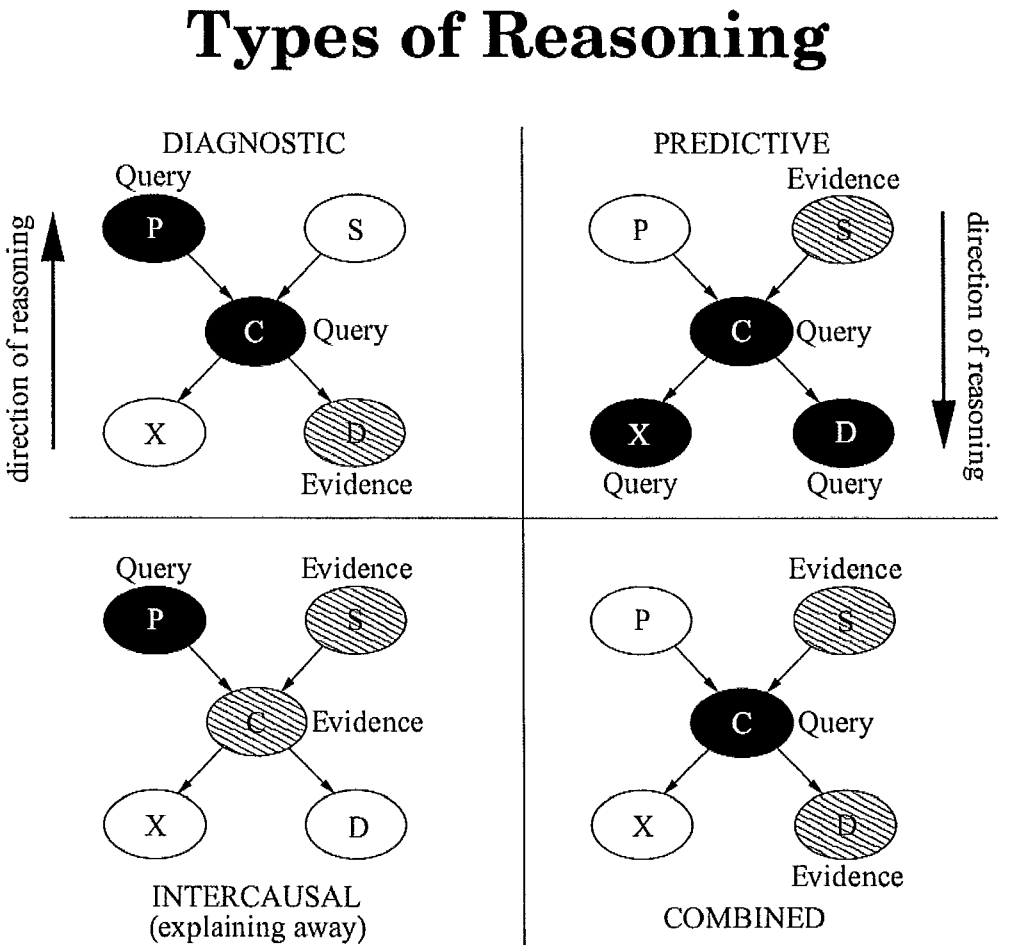


Figure 21. Bayesian inference (Nicholson & Korb, 2006)

Wu (2007) notes that Bayesian Belief Nets build on the Bayesian Probability concept, which does not depend on a physical property, but on the belief of a person that an event will occur, based on the person's prior knowledge. Thus, P (a|K) is the belief that event *a* will occur based on the knowledge of the person *K*. Bayesian probability has been around

for many years and has gained more popularity recently. One of the fundamental rules of Bayesian theory is Bayes's rule:

$$P(a|b) = P(b|a) * P(a) / P(b)$$

where the probability of *a* given evidence *b* (so-called posterior probability) equals the prior probability P (*a*) multiplied by the likelihood of observing evidence *b* given *a*, divided by the probability of *b*. In other words, Bayes' rule could be explained as:

Posterior = Likelihood x Prior / Probability of evidence, (Nicholson & Korb, 2006)

where the denominator is used for normalization. (Wu, 2007)

As noted above, a BBN model is a directed acyclic graph (DAG) where nodes represent variables, and edges represent conditional dependencies between variables. Wu points out that a missing edge denotes conditional independence between the nodes. The structure of the model is often explained as reflecting causal relationships between the variables (Pearl, 2000), and the model can be used both for inductive and deductive reasoning. The conditional probability tables (CPTs) associated with each node capture the influences exerted upon it by the nodes linked with it. (Fig. 22)

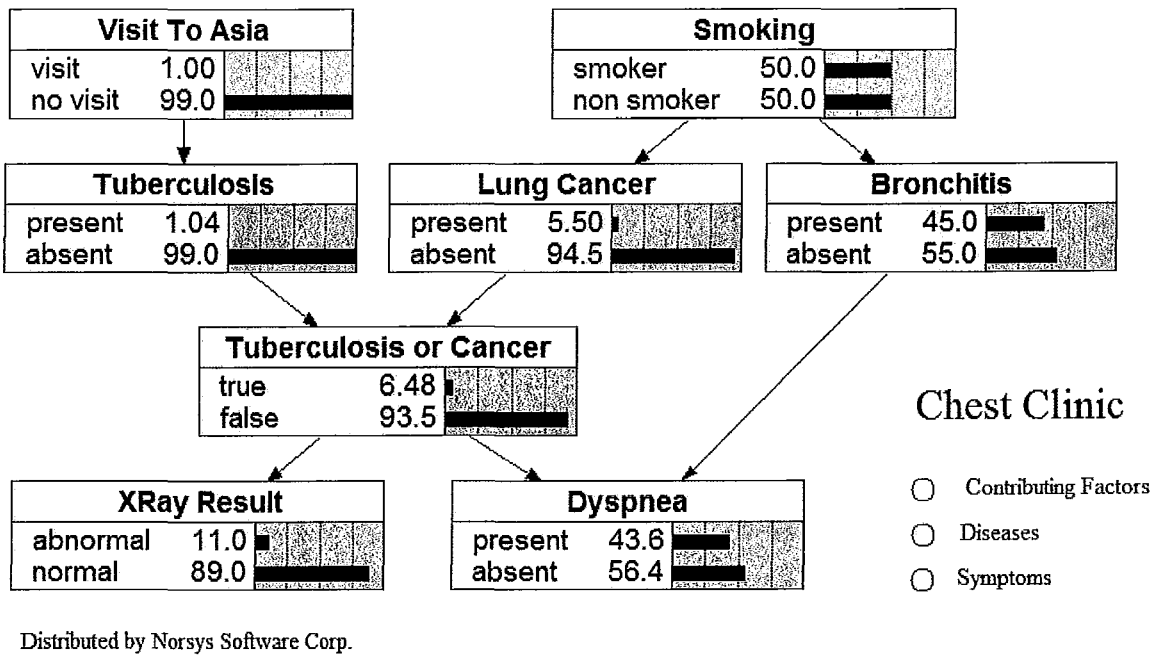


Figure 22. Netica BBN example (Norsys Software Corp., 2006)

One of the advantages of BBN models is their flexibility - they can be modified as new evidence becomes available. Every new set of data can be used to update the CPTs of the

model, taking advantage of the concept of parameter learning. This makes BBN models dynamic and constantly improving.

Wu points out that there has also been work to develop modular BBN models in works such as (Laskey & Mahoney, 1997) and (Koller & Pfeffer, 1997). This makes it possible to combine BBN models as nodes in a BBN framework and thus tackle more complicated problems. In the field of software engineering, BBN models have been used for safety risk evaluation by Fenton, Neil & Littlewood (1996); usability evaluation by Galliers, Sutcliffe & Minocha (1999); reliability and performance evaluation by Gregoriades & Sutcliffe (2005); software architecture evaluation by Gurp and Bosch (2000).

3.7. Summary

This chapter provided a brief overview of the McCall, Boehm, ISO/IEC 9126, FURPS+ and Dromey product quality models, as well as of Bayesian Belief Networks. After reviewing the product quality models presented above we came to the conclusion that ISO/IEC 9126-1 (2001) is the best basis for conducting a product quality evaluation. As its greatest advantage we consider the flexibility of the model, which allows the evaluator to customize it by adding desirable quality characteristics and omitting insignificant ones. Combining this product quality model with BBN use would enable the automation of some evaluation activities (e.g. parameter learning), and formalize others by explicitly considering expert opinion (e.g. determining the utility associated with different outcomes). The results obtained by using Bayesian Networks are only as good as the model used. In a perfect-case scenario there would be enough available data to verify the structure and parameters of the model, however most companies don't have access to such data. Herein lies the importance of making the greatest use of the expert knowledge available in the literature for inferring the model structure, and we have attempted to condense it in the generic model that we present in the next chapter. It combines the quality characteristics of ISO/IEC 9126-1 (2001) with characteristics discussed in other models, thus providing a convenient source of reference for choosing which characteristics to evaluate. In addition, we discuss a way of transforming the resulting model into a form suitable for BBN use. A way of eliciting model parameters is presented in Chapter 4: Customization of the Quality Model, Collecting Metrics and Reading the Results.

4. Towards a Generic Model

This chapter presents the creation of a generic model combining the ISO/IEC 9126-1 (2001) external quality model with characteristics from the McCall and Boehm quality models. It also discusses a way of transforming the model for use in BBN software. Section 4.2. presents the ideas of A. T. Morris and Peter Beling, who have developed a method for extracting acyclic dependency models from quality standards and have applied it to the ISO/IEC 9126-1 model. (Morris & Beling, 2004) A critique of their work is presented and several modifications to their approach are suggested, leading to a slightly different end result.

The quality characteristics of the McCall, Boehm and ISO/IEC 9126-1 models which have been identified in the preceding chapter have been compiled into a compendium in Appendix A. As a basis for the compendium we have used the glossary of non-functional requirements provided in (Colin et al., 2008), extending it with additional definitions, specification of the relative position of the quality characteristics in the quality models of which they are a part, and suggested metrics. In the appendix, characteristics are listed with a definition, parents[4] (these are the factors that the characteristic contributes to), children (characteristics that contribute to its higher quality) and siblings (other characteristics that influence its parents' quality). The information has been elicited from the aforementioned quality models in order to reconcile the different views contained in them in an attempt to construct a generic model that includes as many of the quality characteristics as possible, and reflects all interrelations.

The idea behind constructing a generic model is to simplify the task of the evaluator, who will not have to compare the models in order to choose the one with definitions closest to the evaluator's understanding, but instead will only need to review the definitions provided in the compendium and remove the insignificant characteristics from the generic model to obtain the desired model structure, also ensuring a better fit. An alternative approach is to start with a basic model, and add the characteristics that are considered significant after reviewing the definitions of the compendium. The first approach is better suited for automation as having a generic model means all the nodes have already been arranged to reflect the underlying dependencies, and the structure can then easily be modified to something more manageable. However, a generic model contains many quality criteria, and might discourage the evaluator. According to Miyoshi and Azuma (1993), "the number of key factors should be kept between three and eight". Therefore, a better approach might be to start with a simpler model and add characteristics as necessary.

[4] The terms parents and children used here and in Appendix A do not correspond to parents and children as defined in Bayesian Net models.

4.1. Constructing the Model

We recognize that there is more than one possible arrangement of the quality characteristics when combining the quality models, and that in order to determine which hierarchy best reflects the relationships between the quality criteria, more research is needed. Namely, data needs to be gathered which would support or refute the choices made.

We started the generic model construction with the ISO/IEC 9126-1 (2001) external quality model. Then, quality factors and criteria from McCall's model were added. The groupings of quality factors into Product Operation, Product Revision and Product Transition have been disregarded in the amalgamation of the ISO/IEC 9126-1 and McCall models. The quality factors listed in the McCall model, which are not already present in the generic model, have been added to the first tier of quality characteristics, directly influencing Overall Quality. The process of combining the models saw the creation of edges between quality characteristics which belong in the same tier, as well as edges that link criteria from the second tier directly to overall quality. Before the model is introduced for use in BBN software, all edges need to be evaluated to determine whether some of them might be omitted without affecting the precision of the model. Some of the quality criteria have similar, albeit slightly different definitions, and depending on the purpose of the evaluation and product domain, some of them might be omitted from the model. A third tier of quality criteria was introduced as a result of the combination of the two quality models. The last model we have incorporated in the generic model is Boehm's model. We have interpreted General Utility in Boehm's model as Overall Quality. Reviewing the definitions in Appendix A, we consider Modifiability in Boehm's model to correspond to Changeability in the ISO/IEC 9126-1 model. Moreover, we have accepted the assumption that Robustness and Integrity are sufficiently similar as suggested by Boehm and have modeled them as a single node labeled Robustness/Integrity. The resulting model, constructed using GeNIe 2.0 (Decision Systems Laboratory, University of Pittsburgh, 2008) is presented in figure 23. The addition of further criteria to the generic model is left for future work.

4.2. Transformation of the Generic Model for Use in BBN Software

As we have discussed above, transforming the model for use in BBN software tools will enable the evaluator to take advantage of their decision support capabilities. This section presents the ideas of A. T. Morris and Peter Beling, who have developed a method for extracting acyclic dependency models from quality standards and have applied it to the ISO/IEC 9126-1 model. (Morris & Beling, 2004) A critique of their work is presented and several modifications to their approach are suggested, leading to a slightly different end result.

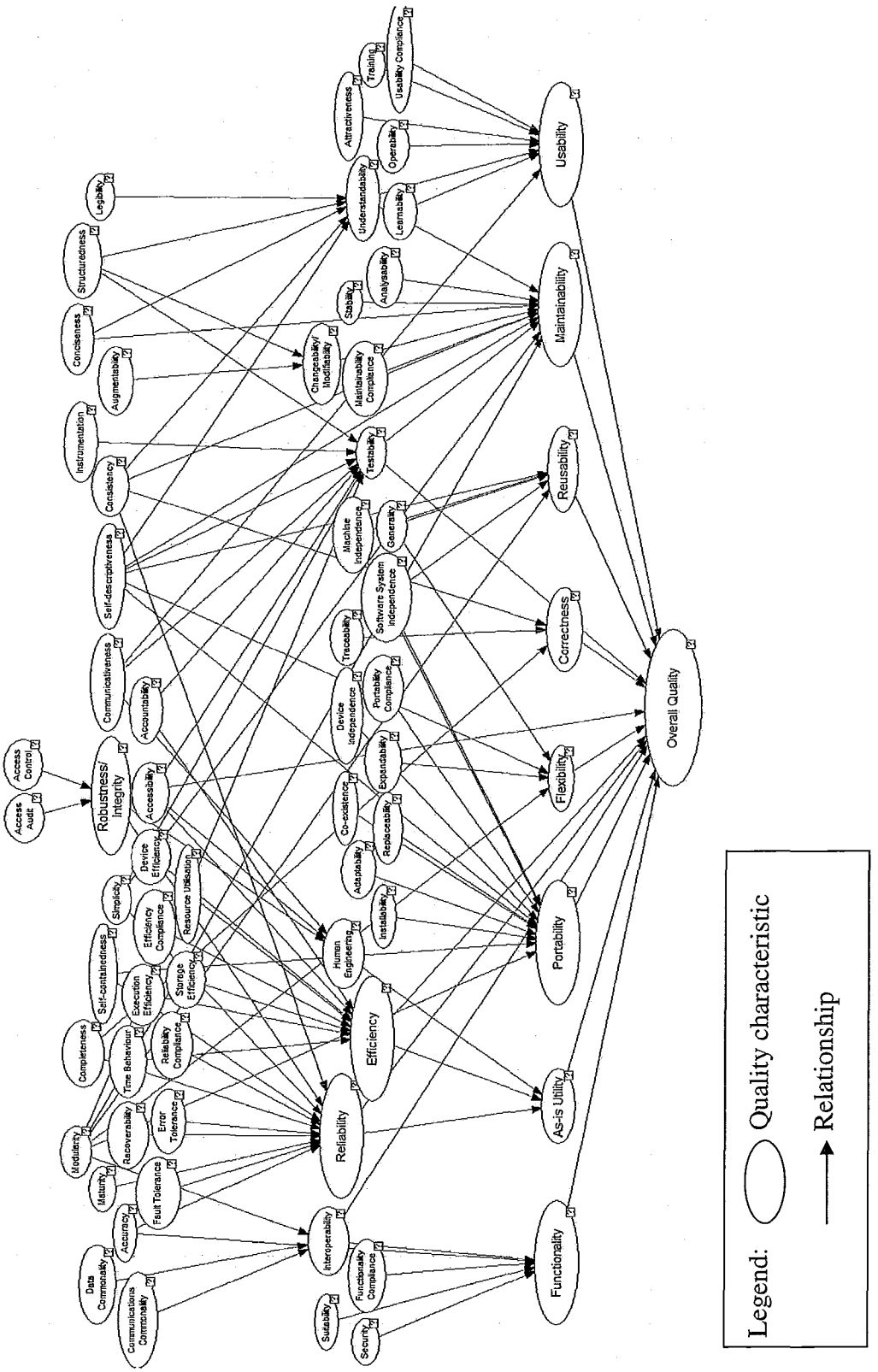


Figure 23. Generic model combining ISO/IEC 9126-1, McCall's and Boehm's models

Morris and Beling (2004) point out that at the time they conducted their research there were no well-established approaches to the evaluation of COTS software quality. The process was often flawed by subjectivity and assumptions regarding the independence of criteria that had no justification. This prompted the authors to review existing standards and attempt to model the dependencies incorporated in them in a graphical way, representing the relationships in a directed acyclic graph (DAG) where possible, thus facilitating product quality evaluation and shedding light on the reasoning behind the standards. The authors focused their work on standards because they "provide a criterion or an acknowledged measure of comparison for quantitative or qualitative value for software". (Morris & Beling, 2004) The transformation of the ISO/IEC 9126-1 model is provided as an example of the application of their approach. The standard is viewed as a knowledge base detailing the criteria that influence software product quality and the dependencies that link them.

The authors start by providing a definition of dependency models and note that both directed graphs (Bayesian networks) and undirected graphs (Markov networks) can be used as a graphical representation of a dependency model. A graphical representation is advantageous because of its ability to reproduce complexity and independence relationships in a more intuitive and compact way. The authors concentrated their research on BBNs, with some of the reasons provided being:

- "Bayesian networks ... result in a powerful knowledge representation formalism based on probability theory";
- "the Bayesian network's requirement of strict positivity allows it to serve as an inference instrument for logical and functional dependencies";
- "its ability to quantify the influences with local, conceptually meaningful parameters allows it to serve as a globally consistent knowledge base". (Morris & Beling, 2004)

For the elicitation of a Bayesian Belief Network when the structure is unknown, Morris and Beling (2004) list several approaches - using past knowledge, conditional independence statements (CIS), data, or a combination of the aforementioned. Past knowledge may refer to expert opinion or problem domain knowledge. The structure of the model is usually elicited with the help of domain experts, and the approach presented in the paper takes advantage of the expert knowledge incorporated in the ISO/IEC 9126-1 standard in order to do exactly that. The steps involved in extracting the model structure are detailed in figure 24.

The application of the approach to the ISO/IEC 9126-1 standard resulted in the BBN model presented in figure 25. The following abbreviations have been used in the figure:

Functionality (F)	Learnability (ln)	M compliance (mc)
Suitability (su)	Operability (op)	Portability (P)
Accuracy (acc)	Attractiveness (att)	Adaptability (ad)

Interoperability (int)
Security (sec)
F compliance (fc)
Reliability (R)
Maturity (mat)
Fault tolerance (ft)
Recoverability (rec)
R compliance (rc)
Usability (U)
Understandability (un)
U compliance (uc)
Efficiency (E)
Time behavior (tb)
Resource utilization (ru)
E compliance (ec)
Maintainability (M)
Analyzability (an)
Changeability (ch)
Stability (st)
Testability (te)
Installability (in)
Co-existence (co)
Replaceability (re)
P compliance (pc)
Effectiveness (Eff)
Productivity (Pro)
Safety (Safety)
Satisfaction (Sat)

Bayesian Network Structure Extraction

Input: A Software Quality Standard containing relational/ causal phrases that describe attribute relationships

Output: A qualitative DAG that encodes the attribute/ sub-attribute dependencies in the software standard

---

Step 1. Identify software quality standard relational/causal phrases that define attribute/sub-attribute relationships.
Step 2. Where possible, translate each causal relation into a conditional independence statement (CIS) and gather to form causal list.
Step 3. Reconcile the CISs to include missed causal relations located elsewhere in the software standard.
Step 4. Using the causal list, generate qualitative forms of local CPDs, then aggregate to form the qualitative JPD structure.
Step 5. Represent the JPD graphically allowing only one node for each attribute/sub-attribute to form the dependency model.
Step 6. Check for cycles in the dependency model; if cycles exist, model is not a DAG (thus, discard); if cycles do not exist, then model is a DAG (thus, DAG represents software standard dependencies).

Figure 24. BBN Structure Extraction Process (Morris & Beling, 2004)

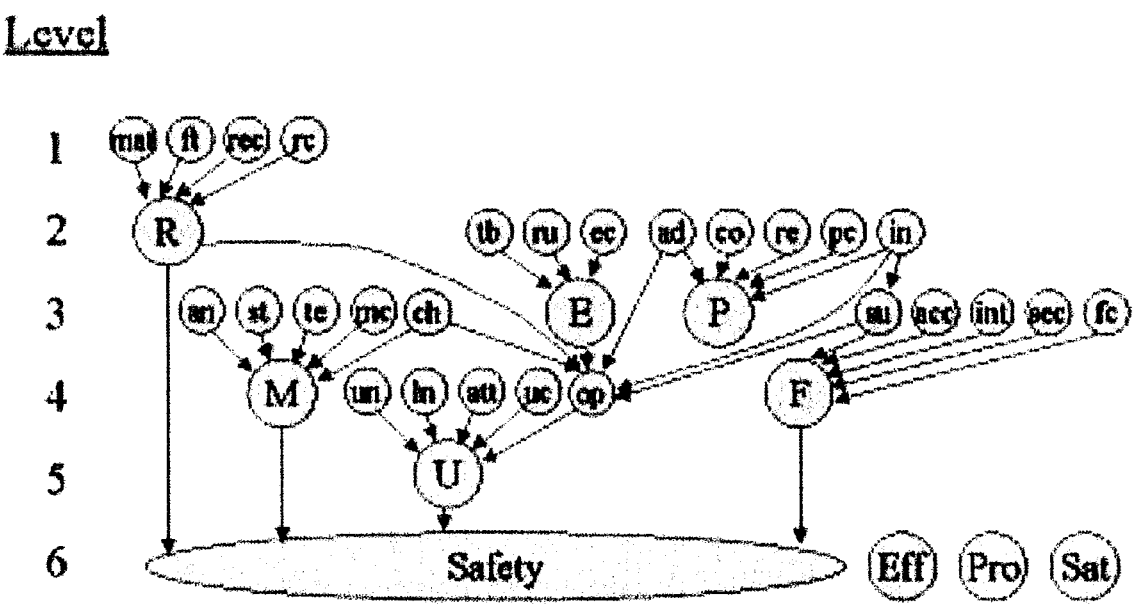


Figure 25. Multi-Level Representation of ISO/IEC 9126-1 Dependency Model (Morris & Beling, 2004)

There are several points in the process that we do not agree with. The first issue is that the authors consider the lack of mention of a causal relation between any two criteria to be equivalent to their conditional independence. We do not agree with this assumption, because the standard was not created with the idea of explicitly listing dependencies and independencies between the quality characteristics, and therefore the information regarding this topic might be incomplete. To correct this shortcoming, more information needs to be gathered, either as a result of expert interviews or literature reviews. Another weak point in our opinion is the mixing of different product quality perspectives in a single model. We believe that the external, internal and quality-in-use quality models should be kept distinct when modeled graphically, and therefore the quality-in-use characteristics should be discarded from the model in the figure above, resulting in the model in figure 26. In addition, Reliability can be introduced in the same level as Efficiency and Portability, reducing the number of levels by one, but we believe the authors chose this setting in order to improve the display of the model.

In our opinion for the sake of completeness, a node signifying Overall Quality needs to be added in the appropriate place, which we believe to be reflected in figure 27, based on the first of fifteen phrases referenced in the paper:

> "Clause 6 -> external/internal quality is characterized by 6 attributes (functionality, reliability, usability, efficiency, maintainability, and portability) that have minimal overlap (clause C.3)" (Morris & Beling, 2004)

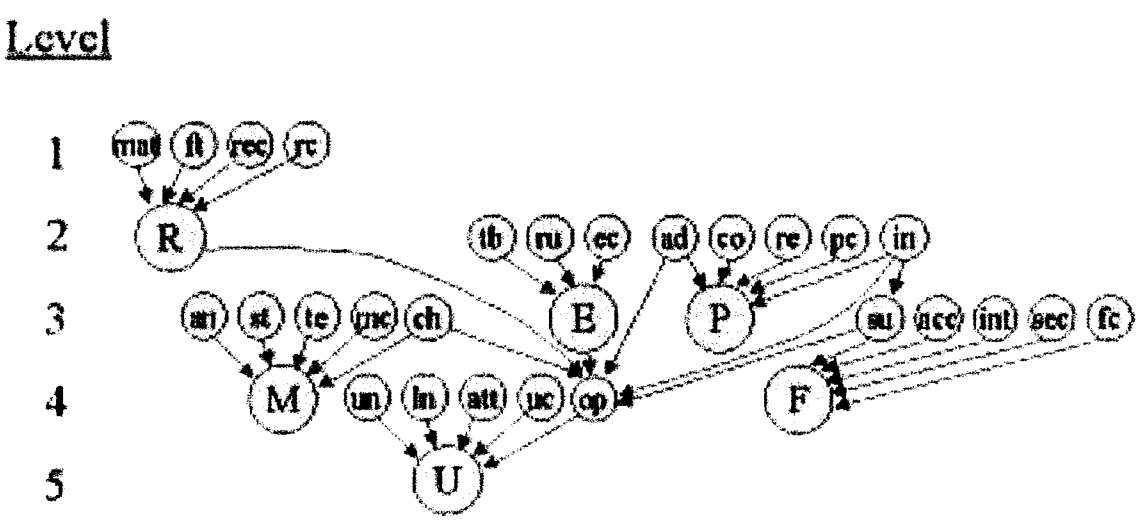


Figure 26. Modified ISO/IEC 9126-1 Dependency Model

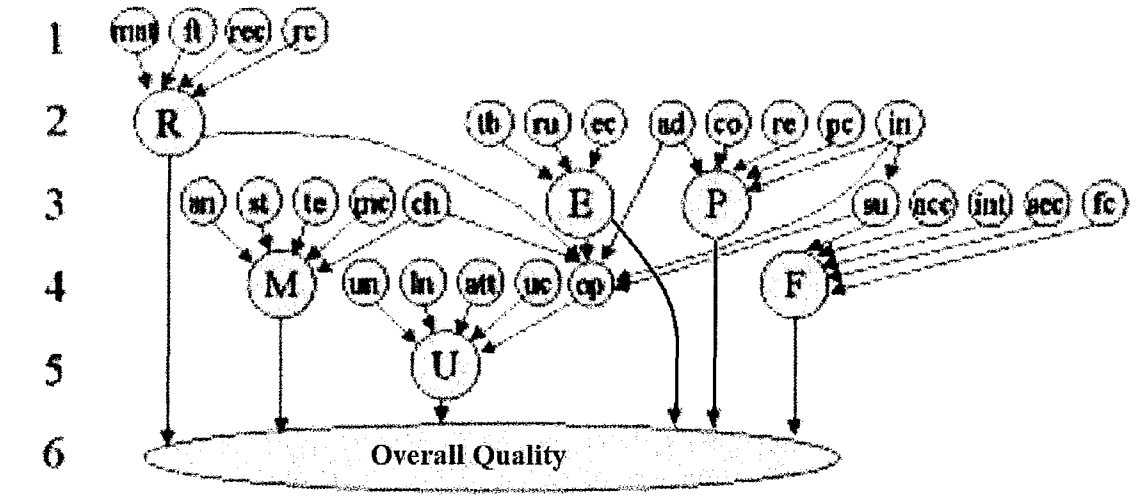


Figure 27. Modified ISO/IEC 9126-1 Dependency Model with Overall Quality

Another aspect that we believe needs improvement is the amount of clarification with regard to the principles guiding the omission of edges that can be inferred from the referenced phrases. For example, the sixth phrase states that "some aspects of functionality, reliability, and efficiency will also affect usability" (Morris & Beling, 2004), yet Functionality is linked to Usability through the direct link between Suitability

and Operability, while Reliability and Efficiency are linked to Usability through a direct link with Operability. There is no phrase that confirms that "some aspects" refers to only these aspects, even if this is in keeping with the authors' assumption that if a dependency link is not established explicitly, it does not exist. Another example is presented in the dependencies established between Installability, Suitability and Operability. The relevant phrases (third and thirteenth) state that "suitability also affects operability" and "installability can affect suitability and operability". (Morris & Beling, 2004) It can be seen in the figure above that all these relationships have been encoded through direct links. However, it might be the case that Installability only influences Operability through Suitability. Even though the logic the authors follow is consistent with their assumption, it does not necessarily lead to an accurate representation of the underlying criteria dependencies. To this end, case studies need to be performed to validate the relationships established, or past data and experts need to be consulted.

Despite some weaknesses of the presented approach, we consider it a step in the right direction, as BBN use is most advantageous when all conditional independencies have been taken into account, and the examples of BBN use in the literature that we are familiar with fail to take that into consideration, instead directly assuming a standard-proposed hierarchy as the model structure, e.g. (Stefani, Xenos, Stavrinoudis, 2003).

Another paper by Morris discusses how to efficiently modify the already defined BBN structure by adding or removing nodes. (Morris, 2007) Clique trees are used in an attempt to make it possible to amend the quality hierarchy without having to go through all the steps discussed in (Morris & Beling, 2004). The approach is developed as a means of using less time and rework when modifying an evaluation model, which might be prompted by a change of requirements. Secondary structures such as clique trees present us with a deeper understanding of the logic of the network structure. Morris elicits the clique tree corresponding to the ISO/IEC 9126-1 dependency model depicted above and discusses its potential uses. The clique tree transformation process is described step by step and its application to the ISO/IEC 9126-1 dependency model is detailed in the paper, resulting in figure 28. The clique tree definition is seen as a first step in an ongoing research for the efficient adding and deleting of sub-attributes existing within a specific clique associated with an attribute, adding and deleting of cliques associated with attribute clusters, and exchanging of cliques associated with attribute clusters. We believe that results from this future research would be very useful, as they would simplify the task of constructing or modifying a quality hierarchy for BBN use.

The generic model constructed in the previous chapter is in fact a DAG (directed graph with no cycles). Morris and Beling (2004) refer to the work of Castillo, Gutierrez & Hadi (1997) to establish that a dependency model which has a directed multi-level representation is necessarily a DAG. In a directed multi-level representation nodes in the same level are not linked, and all arrows point in the same direction. These rules make it clear that the generic model represented in fig. 23 is a DAG. This makes it possible to use it as a BBN model; however, it is not clear whether all the conditional dependencies have

been taken into account. Following the approach outlined by Morris and Beling (2004) would involve extracting phrases from the literature signifying relationships between quality attributes of the model. Because of the amount of literature that needs to be reviewed, this has been left for future work. As an alternative, the ISO/IEC 9126-1 model transformed for BBN use can be extended by adding only the quality characteristics that are considered of importance. The simplification of this process is the subject of ongoing research by Morris, expanding on the use of clique trees as mentioned in (Morris, 2007). It remains to be seen whether the models obtained by using this approach provide better results compared to the models directly reflecting the quality model structure presented in the literature, and we consider this as future work.

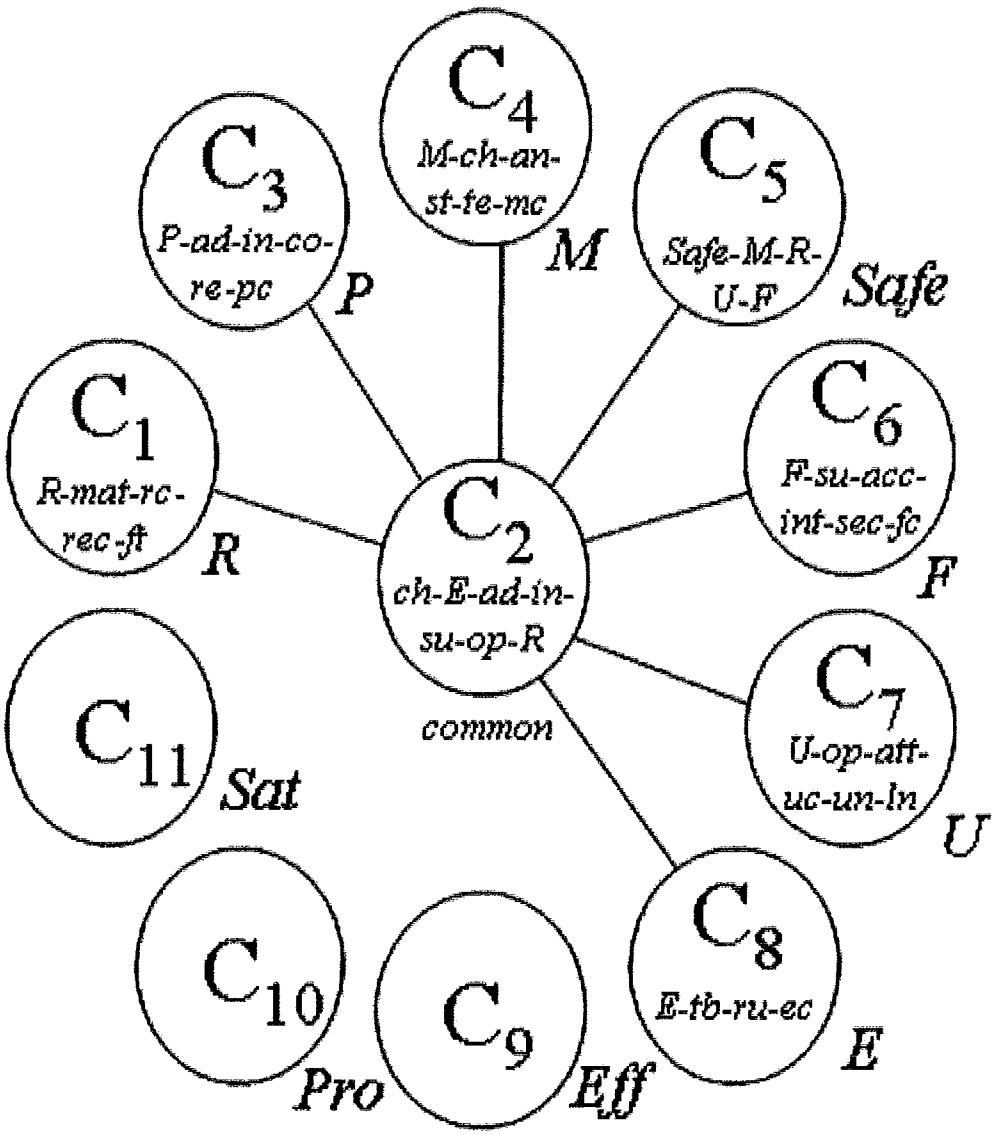


Figure 28. Clique Tree for the ISO/IEC 9126-1 DAG (Morris, 2007)

4.3. Summary

In this chapter we have presented a generic model combining the ISO/IEC 9126-1 (2001) external quality model with all characteristics from the McCall and Boehm quality models. The creation of the model was facilitated by the information collected in Appendix A, which was created as a convenient knowledge base assisting the evaluator in the choice of quality characteristics to include in the model. Being able to easily tailor the quality model to the evaluation needs of the company ensures that time and resources would be allocated to the activities that will provide the greatest return. Customizing the model by including significant characteristics and omitting the insignificant ones results in a better model, and therefore, more meaningful results. The generic model is a directed acyclic graph and can thus be used in a BBN setting; however, we cannot claim conditional independence of the nodes which have no direct links, and more work is needed in that regard. The ideas of Morris and Beling (2004) for establishing explicitly the conditional dependencies have been presented together with a critique of their work and several suggestions for modifications of their approach. The ISO/IEC 9126-1 dependency models obtained following their approach and the modified approach we have suggested are also presented. Conducting a case study comparing the two dependency models is left for future work.

5. Customization of the Quality Model, Collecting Metrics and Reading the Results

This chapter describes the essence of our methodology. It details the evaluation process from beginning to end. First, ways for eliciting the structure of the software product quality model are presented. Then, approaches for model parameter elicitation are discussed, which fall in the deterministic or probabilistic category. These approaches provide a means for establishing the importance of the quality characteristics included in the model and quantifying the relationships between them. Last but not least, two alternatives for metric collection are outlined together with ways of interpreting the obtained quantitative results.

5.1. Eliciting Model Structure

The generic quality model presented in the previous chapter is too broad and requires the collection of many metrics. Its purpose is to serve as an all-encompassing knowledge base, and not to be applied directly, due to the time and cost constraints that companies face. The importance of quality characteristics and factors varies across domains and products and the model needs to be customized to take that into account. Some of the characteristics which are considered insignificant can be dropped from the model so that there is no need to measure their value. Alternatively, they can remain part of the model, with their contribution to the parent's quality downgraded accordingly, and can subsequently be left with unknown value, as BBN software accommodates uncertainty. Evaluators that consider the ISO/IEC 9126-1 (2001) model to be a closer approximation to the tradeoffs they face can take the complementing approach of adding characteristics to the model structure after consulting the compendium in Appendix A. The compendium itself is not exhaustive, and is just one point of reference, designed to facilitate the choice of characteristics to be included in the model. It is created as a "living document", with the idea that other quality characteristics, metrics for their evaluation and other information will be added on a continuous basis, making it a well-informed source of reference. In choosing which characteristics to include, and which to omit, the expert is influenced by the product domain, as well as the purpose of the evaluation.

An alternative approach for the elicitation of model structure from expert opinion is to use the so-called card sorting, which is utilized for the grouping of domain concepts. (Duijnhoven, 2003) There are two types of card sorting - free format sorting and guided sorting. In both types, the expert is asked to review the definitions of concepts that are written on index cards, and then to group the cards into categories. With guided sorting, the categories into which concepts can be grouped are already known, while free format sorting involves the creation of piles with related concepts, and then the choice of a label for each pile, effectively introducing a new concept. In free format sorting categories can

be further grouped together and relabeled, thus creating a hierarchy and eliciting the relationships of the domain concepts. Card sorting is well complemented by the think aloud approach, which demonstrates the thought process of the expert and thus elucidates the way that concepts are interrelated. Another technique described in (Duijnhoven, 2003) is laddering, which is a structured interview to group domain concepts into types and subtypes, creating hierarchies. The groupings might be determined by definitional, causal, superset-subset, generalization-specialization relations etc. Additional methods for knowledge elicitation can be found in Janet Burge's work, complete with brief descriptions and references. (Burge, 2001)

The structure definition of large-scale BBN models can be facilitated by the techniques outlined in (Neil, Fenton & Nielsen, 2000), which expand upon object-oriented BBN ideas. These techniques are integrated in some BBN software tools, such as Hugin (Hugin Expert, 2007) and AgenaRisk (Agena Ltd., 2007). More techniques can be found in (Mahoney & Laskey, 1996).

5.2. Eliciting Model Parameters

The next step that needs to be taken is the quantification of the relationships between the quality characteristics and subcharacteristics. There are several ways to accomplish this. The relationships can either be modeled deterministically, in which case the contribution of each characteristic to the parent's value needs to be evaluated, or probabilistically, in which case the probability of the parent having a certain value needs to be determined based on the values of its children.[5] The deterministic relation between the nodes can be elicited using the AHP method (Han & Han, 2004) or the Mutual Comparison method (Behkamal, Kahani & Akbari, 2009). BBNs are not put to their best use in this case, they only serve as convenient visualization tool and automate the sensitivity analyses. The usefulness of BBN software is best demonstrated in the context of probabilistic modeling of the relations between the quality characteristics. If there is sufficient training data, several BBN software tools provide the so-called parameter learning - a way to infer the probabilistic relationships between nodes using the metrics collected for them in the past. The model will be most precise and useful if all underlying conditional independencies have been taken into account. The probabilistic dependencies can also be elicited using the help of experts, and ways to accomplish this have been presented in papers by (Fenton, Neil & Caballero, 2007), (Stefani & Xenos, 2001). The method proposed by Fenton et al. (2007) for the Conditional Probability Table elicitation has been automated in the AgenaRisk software (Agena Ltd., 2007).

[5] The terms parents and children used here and in Appendix A don't correspond to parents and children as defined in Bayesian Net models.

5.2.1. Deterministic Parameter Elicitation

A popular method for structuring multi-criteria decision making processes is the Analytic Hierarchy Process (AHP). As part of the process, the importance of criteria is determined quantitatively based on pairwise comparisons. In (Han & Han, 2004) D. Han and I. Han describe AHP in the following way:

- As a first step, a decision hierarchy needs to be formulated. It is elicited based on experts' domain knowledge and the goal set. All decision-influencing criteria need to be taken into account.
- As a second step, the criteria are compared in pairs and their relative importance for attaining the goal is specified. Alternatives are compared in pairs in order to specify their preference on each criterion.
- The third step involves transforming the collected data into relative weights for the criteria and alternatives.
- As a fourth step, a ranking of the alternatives is obtained, through computing the composite priorities of each decision element at each level. This is accomplished by using hierarchical composition for aggregating the available weights.

The authors go on to say that AHP assists in choosing the best alternative by synthesizing all the available data in a more manageable form, thus preventing the omission of details in the decision-making process.

An alternative to the AHP for the quantification of characteristic importance is the Mutual Comparison method, which we have utilized for our case study. A detailed description of this method, complemented by its application is provided in the chapter detailing the case study.

The product domain and the purpose of the evaluation influence not only the structure, but also the parameters of the model. With the deterministic approach, this information gets reflected in the weights given to the characteristics chosen for inclusion in the model. Two models that display the same structure, but different preferences with regard to the importance of the quality characteristics modeled will provide different results. This can be noted when considering product quality from the different perspectives of user, developer and manager. A comparison of the results acquired would reflect the importance that each group places on the varying quality characteristics and will help outline potential conflicts and misunderstandings. This has been done for a set of deterministically linked nodes, but we are not aware of a work that compares probabilistic results acquired from several perspectives using the same model, and consider this an avenue for future work.

5.2.2. Probabilistic Parameter Elicitation

The accuracy of the BBN model greatly depends on the node Conditional Probability Tables, or CPTs. Even if the underlying model structure is a correct representation of the interdependencies between quality characteristics, if the CPTs are not well defined, the outcome of the evaluation will be compromised. This is the reason why a great deal of attention should be focused on their elicitation. The process depends on the access to previously collected data and expert opinion.

In (Duijnhoven, 2003), the author comments that using lotteries, probability scales for marking assessments, etc. to elicit probabilities from experts "tend to be problematic and time-consuming". The author advocates a method proposed in (Van der Gaag, Renooij, Witteman, Aleman, & Taal 1999), which combines numerical and verbal aspects. A figure for each conditional probability that has to be determined is provided to the expert. The figure is supplemented with textual representation of the conditional probability in terms of likelihood and a vertical scale with numerical and verbal anchors. Examples of verbal anchors include "certain", "probable", "impossible". The way the figures are presented to the expert facilitates the concurrent review of probabilities from the same conditional distribution. The method was created with the objective of speeding up the elicitation process and making it more appealing.

In (Fenton, Neil & Caballero, 2007) the authors argue that the evaluators should attempt to reduce as much as possible the involvement of experts, as it is rarely justified to elicit the full set of probabilities, because of cost and time considerations. The approach presented in the paper concerns the elicitation of CPTs for ranked nodes. Ranked nodes are nodes that "represent qualitative variables that are abstractions of some underlying continuous quantities". (Fenton et al., 2007) The method hinges on the use of the doubly truncated Normal distribution with a central tendency, and is supported by case studies. In addition, the method has been incorporated in the Bayesian Belief Network software tool AgenaRisk (Agena Ltd., 2007). It is applicable in all domains and reduces the overhead involved with CPT elicitation. A further advantage is provided by the use of ranked nodes (vs. continuous ones), which simplifies the CPT definition task greatly, provided that some conditions that are described in the paper hold. For other elicitation techniques the authors refer us to presentations in (Díez, 1993), (Huang & Henrion, 1996), (Laskey & Mahoney, 1997), (Laskey & Mahoney, 2000), (Monti & Carenini, 2000), (Van der Gaag, Renooij, Witteman, Aleman, & Taal, 2001). Several methods, including probability wheels and verbal-numerical response scale are presented in (Renooij, 2000). Fenton et al. also point out that the use of verbal-numerical response scales was found to have "markedly improved the efficiency of elicitation and the accuracy of results" in (Van der Gaag, Renooij, Witteman, Aleman & Taal, 2002), and that it is beneficial when nodes are labeled. Fenton et al recognize that Noisy-OR is valuable when working with Boolean nodes and refer to (Zagorecki & Druzdzel, 2004) for a discussion of its use and effects. The method that the authors of (Fenton et al., 2007) propose is seen as a complement, rather than an alternative, to expert elicitation. Another

issue with expert elicitation that the paper discusses is the bias that can be introduced by the selection of questions and the way the problem is represented. For more information the readers are referred to (Kahneman, Slovic & Tversky, 1982). In our opinion, one of the most advantageous aspects of this work that the authors point out is the automation of the approach, which makes it possible for experts who are not familiar with statistics to more easily define CPTs and visually inspect the results of their input throughout the process. The authors have compiled a table comparing manual elicitation with the use of the new approach in two BBN models (for safety assessment and software defect prediction), and discovered that the percentage saving of effort on the manual approach had been 84% and a minimum of 93% respectively. In addition to that, Fenton et al report that their research partners found the predictions of the models constructed with the new approach demonstrably better compared to results of approaches utilized before.

5.3. Collecting Metrics and Reading the Results

Depending on the availability of data and experts willing to participate in the quality evaluation process there are two approaches to measuring the quality of the last tier of quality characteristics. One approach is to use the metrics suggested in the standards and other literature, some of which can be found in the compendium in Appendix A. The list is by no means exhaustive, and some of the metrics have not been validated. However, obtaining some of the metrics might be infeasible if there is no access to the development process data. An alternative approach is to relate the quality characteristics of the model to features of the software product under evaluation which can be either present, or absent. This approach has been utilized in (Stefani, Xenos, Stavrinoudis, 2003) and (Behkamal et al., 2009). Both works link product features to quality characteristics, and compile questionnaires, which have simple yes/no responses. The results of these questionnaires are then propagated up the quality model hierarchy to evaluate overall quality. An illustrative example taken from (Stefani et al., 2003) is the measuring of Accuracy for an E-commerce system, which is linked to the presence or absence of a Search Engine, Shopping Cart, Shopping List, Comparative Presentation and Alternative Presentation. These product features were chosen because the authors are evaluating E-commerce systems' quality; for other product domains a different choice of product features would be appropriate. The whole model can be seen on the website of the Hellenic Open University Software Quality Research Group (http://quality.eap.gr/ecom_en.htm). This approach is similar to the operationalization of soft goals in the goal-oriented requirements frameworks - the presence of certain product features and the absence of others can be set as goals in the requirements definition. Attaining these goals would lead to the satisfaction of the soft goals that underlie product quality. A note worth making is that the metrics associated with the goal-oriented paradigm need not be binary. As an example, we refer the reader to (Basili, Caldeira & Rombach, 1994) where the goal-question-metric (GQM) approach is described. It links a goal to a set of questions that answer how the goal is to be attained and a metric that assesses how well that has been done. The approach is applied top-down, breaking down

quality into satisfiable goals. Then the values at the bottom can be acquired from the metrics associated with goal satisfaction.

Once the values for the lowest model layer have been elicited, they need to be propagated up in the hierarchy based on a certain set of rules. One approach is the deterministic one, in which each quality subcharacteristic is characterized by a weight, signifying its importance for the quality characteristic it influences. Ways to determine these weights have been discussed above. Another way is to use conditional probability tables, relating the value of the parent quality characteristic to the values of its children (note that the use of the terms parent and child are reversed in the definition of a BBN model), as elaborated above. Once the metrics data is collected and propagated to the top of the tree, another question arises - how should we interpret the results? One approach is to create a ranking which matches values from the quantitative scale with qualitative descriptions of quality such as Excellent, Good, Average, Satisfactory and Poor. A way to establish such a ranking by eliciting Scale Calibration Tables (SCTs) from past data is presented in (Stefani, Stavrinoudis & Xenos, 2004). The same work notes that the ranking can be alternatively obtained by querying experts or combining both approaches.

Stefani et al. (2004) provide a way of calibrating a model to make inferences based on the numerical results obtained for the quality of its characteristics. The model presented in their work was developed for the evaluation of quality of E-commerce systems. It has Boolean leaf nodes with values elicited from a user survey, intermediate nodes that have states of "good", "average" and "poor" and a central Quality node that has states of "good" and "poor". The ranked values correspond to a numerical value from the interval 0 to 1. The introduction of ranked nodes was prompted by the desire to be able to compare different E-commerce systems based on quality characteristics, as well as to be able to pinpoint problem areas.

The intervals corresponding to the different labels of the ranked nodes were elicited as a result of experimental measurements performed on 120 E-commerce systems. The values for the leaf nodes were entered as evidence, and the probabilities of the other nodes were calculated based on that information. As a means of validation, the normal distribution of each quality characteristic was considered. The data acquired in the case studies was grouped in clusters corresponding to good or bad quality. The authors distinguish three methods for grouping the data into clusters:

- Setting boundary values for the clusters before conducting the case studies;
- Setting the boundary values using percentages of the data results;
- Setting the boundary values as a result of the natural clustering of data observed in the measurement results.

The paper details the third method, which was chosen because of its more representative rates. The data available supported that approach, as it was "clearly distributed in

different clusters", providing precise results, as can be seen in figure 29. (Stefani et al., 2004)

After the ranking has been established, the model can be used in two ways - forward and backward. An example of forward use is the evaluation of quality based on the values supplied for the leaf nodes and providing probabilities for observing good versus poor quality. The backward use of the model could be applied to infer what the values of the leaf nodes need to be for a specified value of overall quality, thus effectively suggesting a set of requirements necessary for building in the specified quality. The model is made available at http://quality.eap.gr/default_en.asp.

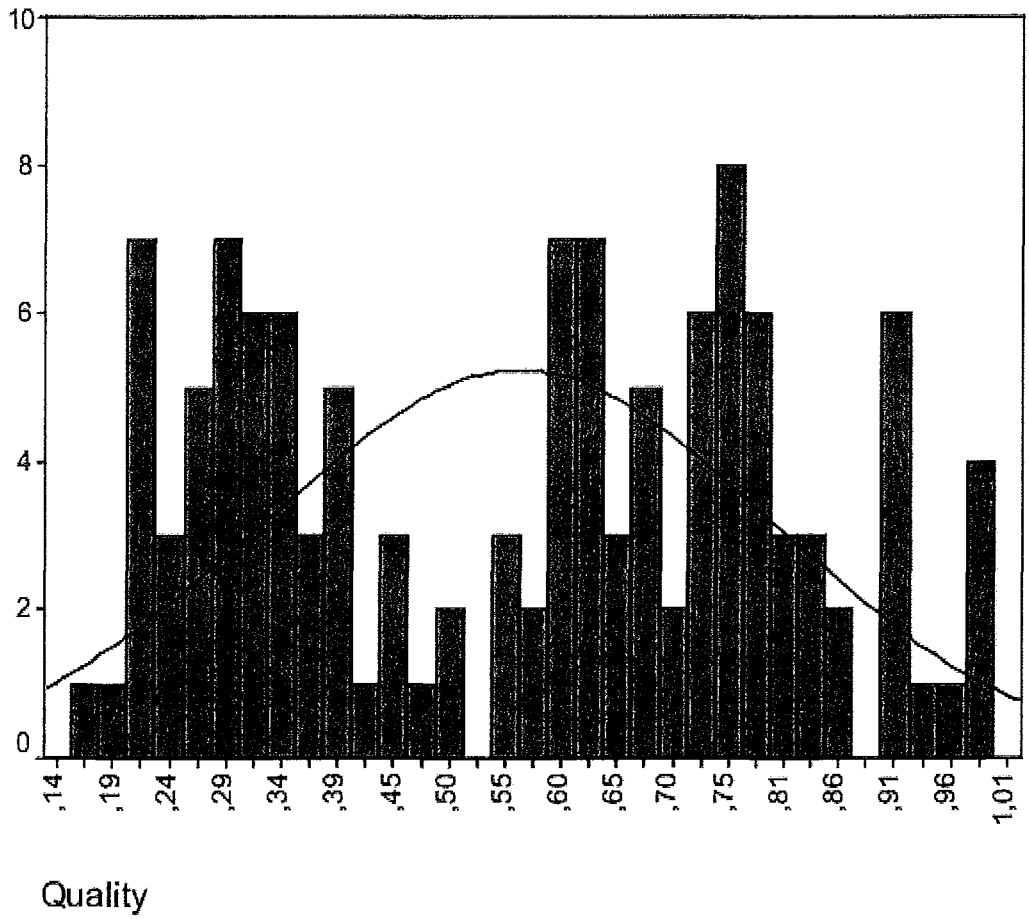


Figure 29. Histogram for overall quality (Stefani et al., 2004)

The results for overall quality obtained from the experimental data evaluated in the paper are normally distributed and grouped into three different clusters. Category A groups the E-commerce systems with high overall quality, meeting the strictest criteria for the underlying quality attributes; category B includes systems with average overall quality,

which have a number of issues that need improvement; and category C groups the systems with low overall quality. The systems were grouped in three categories because of the natural clustering displayed by the overall quality data, as can be seen in the histogram in figure 29. The boundaries of the categories were determined using the minimal and maximal values of each cluster. The same approach was used for defining the categories of the quality characteristics and subcharacteristics, and it was observed that all data was distributed normally. The results were gathered in a Scale Calibration Table. (Table 2)

Scale Calibration			
Category	**A**	**B**	**C**
Quality	x>0,88	0,88>x>0,53	x<0,53
Functionality	x>0,82	0,82>x>0,55	x<0,55
Security	x>0,82	0,82>x>0,55	x<0,55
Interoperability	x>0,93	0,93>x>0,80	x<0,80
Suitability	x>0,83	0,83>x>0,46	x<0,46
Accuracy	x>0,83	0,83>x>0,61	x<0,61
Reliability	x>0,84	0,84>x>0,62	x<0,62
Fault Tolerance	x>0,80	0,80>x>0,57	x<0,57
Maturity	x>0,80	0,80>x>0,62	x<0,62
Recoverability	x>0,84	0,84>x>0,62	x<0,62
Usability	x>0,87	0,87>x>0,63	x<0,63
Attractiveness	x>0,89	0,89>x>0,72	x<0,72
Learnability	x>0,90	0,90>x>0,60	x<0,60
Understandability	x>0,82	0,82>x>0,57	x<0,57
Efficiency	x>0,90	0,90>x>0,39	x<0,39
Resource Behavior	x>0,87	0,87>x>0,53	x<0,53
Time Behavior	x>0,86	0,86>x>0,44	x<0,44

Table 2. Scale Calibration Table (Stefani et al., 2004)

Once the Scale Calibration Table has been created, a system's evaluation results can be used to determine which category it falls into with regard to overall quality, or with

regard to a given quality characteristic, and then inferences can be made concerning any need for improvement, and what are the specific areas which need it most. This helps to focus resources on the problematic areas. The proposed approach facilitates not only quality assessment, but also design activities, as it helps to concentrate on the attributes necessary for the product in order to display quality, and thus helps explicitly consider the quality attributes and how to build quality into the product. Other uses for the model include performing sensitivity analyses which can demonstrate what the optimal solution is when we need to increase overall quality or the quality of a certain characteristic, for example Usability. Quality characteristics which have been proven insignificant can be dropped from the model, taking into account any further modifications that need to be made to preserve the conditional independence claim. Evaluating software becomes more objective using this approach in the sense that the ranking is obtained by evaluating the available data, and not based on subjective judgment. This would be true of models using deterministic as well as probabilistic parameter elicitation.

5.4. Summary

We have suggested a methodology for evaluating software quality which builds on the state of the art that we are familiar with. We have attempted to collect the diverse information available in the literature in one place and link the fragmented views available through different sources into a comprehensive methodology addressing all the important aspects of software quality evaluation. In this chapter we have provided alternatives for eliciting the software product quality model structure and parameters, for collecting metrics and interpreting the obtained results. Understanding the importance of model customization is paramount to appreciating the value of our work. The possibility to customize, to adjust the quality model and evaluation process to best utilize the available data and expert knowledge promotes more accurate evaluation results, and therefore, better company project performance. We offer some insights for the evaluation process below:

- If there is data from previous evaluations, we advise the evaluator to use the structure and parameter learning options of BBN software, and subsequently verify with experts the obtained results.
- When there is no data, but there is expert intuition with regard to the relationships of the quality characteristics, the probabilistic approach to parameter elicitation can be used, complemented when possible with the use of software tools facilitating the elicitation.
- When no evaluator expert opinion is available, the deterministic approach to parameter elicitation is more suitable.
- When the evaluator does not have access to testing and development data, it might be more suitable to take a goal-oriented approach to metric collection.

- Having access to past data accommodates the mapping of quantitative results to qualitative statements using the clustering of values observed. When data is unavailable, the evaluator can use expert opinion.

The next chapter details the application of the proposed methodology to a case study.

6. Case Study

This chapter presents a real-world case study following the outlined methodology. The case study that we conducted evaluated the quality of a product monitoring network health. The company that developed the product undertook the evaluation with the objective to start a formal assessment of software quality which would facilitate the comparison of company products developed under the old methodology, and under the newly adopted AGILE methodology. The approach taken was to use the slightly modified ISO/IEC 9126-1 (2001) model, which was considered representative enough by the company experts. The compliance characteristics were the only ones removed and no other characteristics have been added. The model was quantified using the mutual comparison method from the perspective of user, developer and manager. Since there were no added characteristics, all the metrics data gathered followed the suggestions of the ISO/IEC 9126-2 (2003) standard. Values for the metrics were collected by the company quality analysts. Because of the lack of previous data and expert knowledge on the subject, it was not possible to construct Scale Calibration Tables and infer qualitative rankings. Several applications of the obtained results are discussed at the end of the chapter.

6.1. Eliciting Model Structure

The company experts were presented with the ISO/IEC 9126-1 (2001) quality model, which we suggested as a basis for the evaluation, as well as with the definitions of additional quality characteristics (gathered in Appendix A). After a review of the model and definitions, the experts considered the criteria represented in the ISO model sufficient for the purpose of the evaluation. Nothing was added to the ISO quality model; however, the functionality compliance, reliability compliance, usability compliance, efficiency compliance, maintainability compliance and portability compliance subcharacteristics were omitted, as they were deemed not applicable. This decision was made because there were no "standards, conventions or regulations in laws and similar prescriptions" that needed to be followed by the company in developing the product. (ISO/IEC, 2001)

6.2. Eliciting Model Parameters

Once the model was finalized, the next step was to determine the weights of the quality characteristics. There was no available data on similar projects in the past and therefore a deterministic approach to quantifying the relationships in the model was chosen. In other work (Behkamal et al., 2009) a combination of the mutual comparison and the AHP methods has been used, we consider applying only the mutual comparison method less prone to random answers, and that is especially important since the number of experts

who filled out the questionnaire from each viewpoint is relatively small and any inconsistencies in their answers would be difficult to resolve.

6.2.1. The Mutual Comparison Method

The method chosen for eliciting the weights was the mutual comparison method, as described in (Behkamal et al., 2009). As the name suggests, the method is based on the mutual comparisons of the quality criteria, which are collected in an n-by-n (square) matrix, where n is the number of the characteristics being compared. The rows of the matrix display the preference of the quality characteristic in the row title over the quality characteristics listed in the column titles. The preferences are encoded using the integers 1-5 and their reciprocals, as follows: 1 denotes equal importance of the quality attributes, 2-5 denote that the first characteristic is more important than the second, the strength of importance growing with the magnitude of the numbers. For example, 5 signifies that the first characteristic (row title) is extremely more important than the second (column title). The reciprocals 1/2-1/5 denote that the first characteristic is less important, 1/5 signifying it's significantly less important than the second. Data was collected through a questionnaire, a sample of which can be seen in Appendix B. The questionnaires were filled in by four users, two developers and two managers. The sample is representative enough, as the company is relatively small. Separate calculations were made for the different stakeholder perspectives, in order to analyze the difference in perceived importance of the quality characteristics for overall quality. An example of the calculations made can be seen in Appendix B.

The questionnaires collect information about the importance of quality characteristics for overall quality, as well as the importance of quality subcharacteristics for their parent quality characteristic. Since comparing a characteristic with itself should yield equal importance, the main diagonal of the matrix is filled with '1'-s, and this comparison is not included in the questionnaire. (Fig.30) This reduces the number of necessary questions to n(n-1). Moreover, the method stipulates that if when comparing characteristic A with characteristic B we obtain x, than comparing B with A should yield 1/x. This further reduces the number of necessary comparisons to n(n-1)/2. All of these questions are necessary, because the method does not assume transitivity. In this it is similar to the AHP, which "operates under the axiom of non-transitivity of preference ratios as well as under the assumption of reciprocal judgments." (Bardis, 2009) This makes it prone to inconsistencies, as we shall see below. However, we agree with Bardis (2009) that "a valid preference structure does not necessarily require or imply transitivity."

	A	B	C
A	1	x	
B	1/x	1	
C			1

Figure 30. Mutual comparison matrix

6.2.2. Evaluation of Respondent Judgment Consistency

The questionnaires were designed to elicit expert opinion in a way that minimizes random and inconsistent answers; however, because of the nature of the human mind, there are always inconsistencies, be it because of conflicting, potentially irrational opinions, or the choice of a scale that is not fine-tuned enough to reflect all nuances of preference. For example, even if the quality characteristics A and B are considered equal in importance, corresponding to a 1 in the comparison matrix, comparing A with C might yield 3, and comparing B with C might yield 4. This does not necessarily signify a mistake on the part of the respondent, as the actual preference in both cases might be somewhere in between 3 and 4, and A and B might not be entirely equal in importance, but sufficiently equal so as not to clearly prefer one over the other. The problem then becomes how to determine what level of inconsistency is acceptable, and to establish whether a minor modification to the matrix could lead to a significant increase in consistency. An example where the latter might be very beneficial is catching an involuntary mistake the respondents made. It is our belief that any change to the matrix has to be performed with the understanding and approval of the person who filled in the questionnaire. For example, looking at a matrix that displays A equal in importance to B, and A extremely more important than C (signified by 5 in the matrix), one would expect that B would also be more important than C. If instead we notice that the corresponding matrix component is equal to 1/3, we might want to further verify with the respondent whether one of the numbers entered was a result of a mistake, and which one exactly. Even if the expert confirms that the data is correct, or there is no way to contact him/her (in particular if the questionnaire was anonymous) we might still find the data useful, as long as it is considered sufficiently consistent. In addition to visual inspection, one might use the method described by Saaty (2003) to determine which component modification would lead to the highest increase in consistency.

There are several methods for evaluating pairwise comparison matrix consistency, for example, if using the AHP method, one might use Saaty's consistency ratio (CR) (Saaty, 1980), which has been discussed and critiqued in (Xu, Dong & Xiao, 2008). Unfortunately, this ratio is dependent on the numerical scale applied in the Conventional-AHP, and therefore does not facilitate consistency testing when other scales are applied. The method has the following logic - a consistency index is defined, which is dependent on the order of the matrix, and therefore the consistency measure is given as a normalization of this index by dividing it by an index that captures the expected value of the consistency index for this particular order. Because the normalization index has been calculated through applying the Conventional-AHP prioritization method to a large number of simulated matrices using the {1/9,...,1,...9} scale, the data provided as a result of the simulation cannot be directly applied to this case study, which uses the {1/5,...,1,...5} scale. One of the areas proposed by the authors of (Xu et al., 2008) for future work is to develop a method that would not be contingent upon the numerical scale used for prioritization. Another consistency testing method is suggested by Aguarón and Moreno-Jimenez (2003), but again is dependent on the scale. Aguarón and Moreno-

Jimenez give a definition for the Geometric Consistency Index (GCI) and suggest thresholds for it, relating it to Saaty's work. This consistency test is applicable when the Row Geometric Mean Method (RGMM) has been used, which is one of the two methods utilized in this study. Several other consistency evaluation methods have been referenced in (Xu et al., 2008).

Barzilai (1998) suggests two measures for the evaluation of consistency of pairwise comparison matrices, the relative consistency (RC) and the relative error (RE) measures. He compares them with other popular consistency measures, such as Saaty's consistency index. The RC and RE measures are mapped onto the closed interval [0,1] and their sum equals 1. One of the advantages of these measures pointed out by the author is the fact that they are not dependent on the order of the matrix, which makes it possible to come up with a single consistency threshold, which would signify what values are considered sufficiently consistent. It is also possible to compare the consistency of matrices of different order. Another advantage of the proposed measures is their independence of the scale used for preference encoding.

Before we can proceed to the method for calculating RC and RE, let us provide the definitions of a pairwise additive and multiplicative matrix and related terms, excerpted from (Barzilai, 1998):

1. $A = (a_{ij})$ is a pairwise multiplicative matrix if $0 < a_{ij} = 1/a_{ji}$.
2. $w = (w_k)$ is a *multiplicative weight vector* if $w_k > 0$ and $\Pi_{k=1}^{n} w_k = 1$.
3. $A = (a_{ij})$ is a *multiplicative consistent matrix* if $a_{ij} = w_i / w_j$ for some multiplicative weight vector w.
4. $A^{\times}$, $w^{\times}$ and $C^{\times}$ are the sets of all pairwise multiplicative matrices, multiplicative weight vectors and multiplicative consistent matrices respectively.
5. $f^{\times}$ is the set of all mappings from $A^{\times}$ to $w^{\times}$.
6. $A = (a_{ij})$ is a *pairwise additive matrix* if $a_{ij} = - a_{ji}$.
7. $w = (w_k)$ is an *additive weight vector* if $\Sigma_{k=1}^{n} w_k = 0$.
8. $A = (a_{ij})$ is an *additive consistent matrix* if $a_{ij} = w_i - w_j$ for some additive weight vector w.
9. A^{+}, w^{+} and C^{+} are the sets of all pairwise additive matrices, additive weight vectors and additive consistent matrices respectively.
10. f^{+} is the set of all mappings from A^{+} to w^{+}.

The RC measure for a pairwise multiplicative matrix, which is the type of matrix that we construct from our questionnaire, can be obtained by the following method suggested by Barzilai:

- As a first step, the matrix M needs to be transformed into a pairwise additive matrix A by substituting each component with the logarithm of the component to a fixed base. Any number can be used for the base, as the sets of all pairwise

multiplicative matrices and all pairwise additive matrices are related by a logarithmic isomorphism. In our study we have chosen 2 for the base.

- As a next step, we need to calculate the row arithmetic mean vector of A, w.
- The third step involves calculating the consistent component C_A of A. Its definition is as follows:

  $C_A = (c_{ij}) = (w_i - w_j)$
- Next, we need to calculate the sum of the squared components of C_A and the sum of the squared components of A.
- RC is obtained as the quotient of the sums from the previous step (when A is not equal to 0), i.e.

  $RC(M) = RC(A) = \Sigma_{ij} c_{ij}^2 / \Sigma_{ij} a_{ij}^2$

To complete the definition of relative consistency we need to add that RC (0) = 1. (Barzilai, 1998) This means that if we obtain a zero matrix when we transform M to A, then M is a totally consistent matrix. We omit the definition and method for calculation of relative error for brevity, suffice it to say that it can be obtained from the equation

RC(A) + RE(A) = 1. (Barzilai, 1998)

We have computed the relative consistency values for all the matrices derived from the questionnaires that were filled in. The results were grouped to obtain the statistics in tables 3 and 4.

	USER	DEVELOPER	MANAGER
Overall Quality Breakdown consistency	0.88	0.89	0.92
Functionality Breakdown consistency	0.84	0.90	0.87
Reliability Breakdown consistency	0.80	0.99	1.00
Usability Breakdown consistency	0.92	0.87	0.95
Efficiency Breakdown consistency	1.00	1.00	1.00
Maintainability Breakdown consistency	0.89	0.83	0.98
Portability Breakdown consistency	0.83	0.84	0.97

Table 3. Consistency according to Quality Characteristic Breakdown and Perspective

USER 1	USER 2	USER 3	USER 4	DEVELOPER 1	DEVELOPER 2	MANAGER 1	MANAGER 2
0.85	0.87	0.89	0.91	0.91	0.87	0.92	0.92
0.62	1.00	0.75	1.00	0.85	0.94	1.00	0.74
0.85	1.00	0.33	1.00	1.00	0.99	1.00	1.00
0.93	0.91	0.85	0.98	0.84	0.89	1.00	0.91
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.92	0.77	0.88	1.00	0.80	0.85	1.00	0.97
0.85	0.71	0.82	0.95	0.88	0.80	0.99	0.94
0.86	**0.89**	**0.79**	**0.98**	**0.90**	**0.91**	**0.99**	**0.92**

Table 4. Consistency according to respondent

It can be seen from the tables that the average relative consistency of any given perspective for any given breakdown is greater than or equal to 0.80. The average relative consistency grouped by respondent is greater than or equal to 0.79. Granted that taking the average is not the most appropriate measure for adequacy of the responses, it provides a general feeling. Out of the 56 matrices evaluated, there were only 6 with RC values less than 0.80. We have not set a consistency threshold because the data available to us at this time is not sufficient to infer what would be a meaningful cut-off line. However, we have determined that this is not necessarily a weak point, as Barzilai (1998) claims that inconsistency alone cannot warrant the updating of judgments in the comparison matrix. One reason for this is that the projected weights will remain the same if we increase consistency without affecting the underlying judgments, and if the respondent is comfortable with the inconsistency that the answer displays, there is no sense to distort it just in order to increase consistency. As Barzilai clarifies:

> Forcing the values of judgements to improve consistency distorts the answer regardless of the level of consistency of A. In our opinion the projected weights should be presented to the decision maker as feedback from the analysis. If the decision maker can confirm that the matrix C is indeed an acceptable reflection of his/her preferences, no revision of judgements is necessary regardless of his/her level of consistency. Otherwise, a revision of the judgements is justified. (Barzilai, 1998)

Since the questionnaires were anonymous, it is not possible for us to consult the respondents to see whether they accept the obtained results. We trust that they have dedicated sufficient time in providing their views. An alternative approach would have been to exclude the matrices with lowest RC from the calculation of weights for the quality characteristics, as we do not justify judgment modification without the involvement of the respective respondent.

As can be seen from the data, the manager perspective displayed the highest consistency in all but the Functionality Breakdown. It would be interesting to infer whether that arises as a result of a better understanding of the underlying issue, or because of spending more time contemplating the problem. Unfortunately, we cannot make any hypotheses as the questionnaires were filled in anonymously and we have not had a chance to observe the respondents' behavior.

Barzilai (1998) discusses a way to measure the consistency of an entire hierarchy, but we omit this discussion and refer the interested parties to his work.

6.2.3. Deriving the Weights of the Quality Characteristics

After checking the consistency of the expert judgments we proceeded by deriving the weights of the quality characteristics. To this end, two alternative approaches were used - the eigenvector approach (Behkamal et al., 2009) and the geometric mean approach

(Barzilai, 1997). The eigenvector approach as described by Behkamal (2009) involves several steps:

- As a first step, a matrix is constructed based on the questionnaire responses, as described above.
- As a second step, the eigenvalues and eigenvectors corresponding to this square matrix are calculated. We used a free online calculator, available at http://www.bluebit.gr/matrix-calculator/ .
- The eigenvector corresponding to the largest eigenvalue (λ_{max}) is then set as the weight vector.
- After normalization of the weight vector, we have obtained the desired weights.

These steps were followed for all collected questionnaires. After applying the eigenvector method for finding the weights of the different quality characteristics, the results in tables 5-11 were obtained.

The geometric mean approach was discussed by Barzilai (1997). Again we start by constructing the square matrix encoding the respondents' preferences. The weight vector is then obtained by calculating the geometric mean for each row of the matrix. We have proceeded to normalize the vector in order to compare the results of the two approaches. Barzilai claims that the geometric mean is "*the only method* for deriving weights from multiplicative pairwise comparisons which satisfies fundamental consistency requirements." (Barzilai, 1997) Among the advantages discussed by the author are the lack of scale-inversion rank reversal and the fact that the result is not contingent on the formulation of the problem or the order of operations. Using the geometric mean to derive weights from the pairwise comparison matrices we obtain the results in tables 12-18. Additional methods for deriving weights have been discussed in (Blankmeyer, 1987).

Comparing the results of the two approaches, we notice that they are quite similar, the only differences are centered and bolded in table 19 below. The first column in each perspective contains the results of the eigenvector approach (labeled A), while the second contains the results of the geometric mean approach (labeled B). The Reliability, Usability and Efficiency breakdowns display the same values regardless of the approach taken. Using precision of 2 decimal places shows 3 rank changes for the importance of the quality characteristics, even though the difference in values is relatively minor. According to the geometric mean approach Suitability and Security contribute equally to Functionality, while the eigenvector approach determines Suitability as slightly more important (Manager perspective). The eigenvector approach sees Changeability and Stability as equally important for Maintainability, while the geometric mean approach favors Stability slightly (User perspective). The biggest change, leading to rank reversal, concerns Adaptability and Replaceability, which are subcharacteristics of Portability. The eigenvector approach favors the latter, while the geometric mean approach favors the former (Developer perspective). For the purposes of this study we are going to proceed using the results obtained using the geometric mean method because of the advantages pointed out by Barzilai (1997).

	Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
USER	0.2665	0.2500	0.2176	0.0780	0.1254	0.0625
DEVELOPER	0.3447	0.2217	0.1648	0.0921	0.1130	0.0636
MANAGER	0.1590	0.2906	0.1691	0.0522	0.2386	0.0905

Table 5. Overall Quality using the eigenvector method

	Suitability	Accuracy	Interoperability	Security
USER	0.3932	0.2507	0.1915	0.1646
DEVELOPER	0.3467	0.2782	0.1373	0.2377
MANAGER	0.2459	0.2088	0.3072	0.2382

Table 6. Functionality Breakdown using the eigenvector method

	Maturity	Fault Tolerance	Recoverability
USER	0.3308	0.3920	0.2772
DEVELOPER	0.3698	0.3485	0.2817
MANAGER	0.2250	0.4500	0.3250

Table 7. Reliability Breakdown using the eigenvector method

	Understandability	Learnability	Operability	Attractiveness
USER	0.2636	0.2900	0.3643	0.0821
DEVELOPER	0.2001	0.3229	0.3711	0.1059
MANAGER	0.2056	0.2289	0.3796	0.1859

Table 8. Usability Breakdown using the eigenvector method

	Time Behavior	Resource Utilization
USER	0.7083	0.2917
DEVELOPER	0.7750	0.2250
MANAGER	0.3500	0.6500

Table 9. Efficiency Breakdown using the eigenvector method

	Analysability	Changeability	Stability	Testability
USER	0.2042	0.2455	0.2529	0.2975
DEVELOPER	0.3590	0.2240	0.2918	0.1252
MANAGER	0.1781	0.1796	0.3994	0.2428

Table 10. Maintainability Breakdown using the eigenvector method

	Adaptability	Installability	Co-existence	Replaceability
USER	0.2904	0.2948	0.3247	0.0901
DEVELOPER	0.2732	0.3307	0.1211	0.2750
MANAGER	0.1732	0.3201	0.2623	0.2445

Table 11. Portability Breakdown using the eigenvector method

	Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
USER	0.2670	0.2524	0.2173	0.0771	0.1229	0.0634
DEVELOPER	0.3422	0.2253	0.1644	0.0942	0.1116	0.0623
MANAGER	0.1576	0.2803	0.1749	0.0526	0.2440	0.0905

Table 12. Overall Quality using the geometric mean method

	Suitability	Accuracy	Interoperability	Security
USER	0.3926	0.2460	0.1959	0.1655
DEVELOPER	0.3471	0.2789	0.1371	0.2369
MANAGER	0.2439	0.2065	0.3097	0.2398

Table 13. Functionality Breakdown using the geometric mean method

	Maturity	Fault Tolerance	Recoverability
USER	0.3308	0.3920	0.2772
DEVELOPER	0.3698	0.3485	0.2817
MANAGER	0.2250	0.4500	0.3250

Table 14. Reliability Breakdown using the geometric mean method

	Understandability	Learnability	Operability	Attractiveness
USER	0.2637	0.2925	0.3619	0.0818
DEVELOPER	0.1996	0.3198	0.3726	0.1080
MANAGER	0.2059	0.2313	0.3757	0.1871

Table 15. Usability Breakdown using the geometric mean method

	Time Behavior	Resource Utilization
USER	0.7083	0.2917
DEVELOPER	0.7750	0.2250
MANAGER	0.3500	0.6500

Table 16. Efficiency Breakdown using the geometric mean method

	Analysability	Changeability	Stability	Testability
USER	0.2058	0.2444	0.2526	0.2972
DEVELOPER	0.3567	0.2267	0.2948	0.1218
MANAGER	0.1780	0.1804	0.3994	0.2422

Table 17. Maintainability Breakdown using the geometric mean method

	Adaptability	Installability	Co-existence	Replaceability
USER	0.2901	0.2910	0.3252	0.0937
DEVELOPER	0.2762	0.3340	0.1210	0.2688
MANAGER	0.1730	0.3205	0.2614	0.2450

Table 18. Portability Breakdown using the geometric mean method

	USER		DEVELOPER		MANAGER	
	A	B	A	B	A	B
Overall Quality						
Functionality	0.27	0.27	0.34	0.34	0.16	0.16
Reliability	0.25	0.25	**0.22**	**0.23**	**0.29**	**0.28**
Usability	0.22	0.22	0.16	0.16	0.17	0.17
Efficiency	0.08	0.08	0.09	0.09	0.05	0.05
Maintainability	**0.13**	**0.12**	0.11	0.11	0.24	0.24
Portability	0.06	0.06	0.06	0.06	0.09	0.09
Functionality Breakdown						
Suitability	0.39	0.39	0.35	0.35	**0.25**	**0.24**
Accuracy	0.25	0.25	0.28	0.28	0.21	0.21
Interoperability	**0.19**	**0.20**	0.14	0.14	0.31	0.31
Security	**0.16**	**0.17**	0.24	0.24	0.24	0.24
Reliability Breakdown						
Maturity	0.33	0.33	0.37	0.37	0.23	0.23
Fault Tolerance	0.39	0.39	0.35	0.35	0.45	0.45
Recoverability	0.28	0.28	0.28	0.28	0.33	0.33
Usability Breakdown						
Understandability	0.26	0.26	0.20	0.20	0.21	0.21
Learnability	0.29	0.29	0.32	0.32	0.23	0.23
Operability	0.36	0.36	0.37	0.37	0.38	0.38
Attractiveness	0.08	0.08	0.11	0.11	0.19	0.19
Efficiency Breakdown						
Time Behavior	0.71	0.71	0.78	0.78	0.35	0.35
Resource Utilization	0.29	0.29	0.23	0.23	0.65	0.65
Maintainability Breakdown						
Analysability	**0.20**	**0.21**	0.36	0.36	0.18	0.18
Changeability	**0.25**	**0.24**	**0.22**	**0.23**	0.18	0.18
Stability	0.25	0.25	0.29	0.29	0.40	0.40
Testability	0.30	0.30	**0.13**	**0.12**	0.24	0.24
Portability Breakdown						
Adaptability	0.29	0.29	**0.27**	**0.28**	0.17	0.17
Installability	0.29	0.29	0.33	0.33	0.32	0.32
Co-existence	**0.32**	**0.33**	0.12	0.12	0.26	0.26
Replaceability	0.09	0.09	**0.28**	**0.27**	**0.24**	**0.25**

Table 19. Comparison of the results obtained using the eigenvector method (A) and the geometric mean method (B)

It is interesting to note that as expected, developers attribute highest importance to Functionality, and users are the group that cares most about Usability. (table 20) Efficiency and Portability are stressed least by all groups, which seems to be a reflection of the product domain, and managers put the greatest emphasis on Maintainability, estimating it as roughly twice as important as the other groups perceive it. Managers are also the group that esteems Portability most, which is again expected, as it creates more

market opportunities. Some of the obtained results provide new intuition, e.g. managers perceive product Attractiveness as far more important than users do, more than two times as much. (table 20) In addition, users and developers both consider Time Behavior as more important than Resource Utilization, while managers hold the opposing view. (table 20) This might lead to misunderstandings in the development process and we consider outlining such differences as very important for the smooth operation of the company. It is also unexpected to see Testability get its least importance value from developers, even though this could be explained as being the result of attributing more weight to Analysability, which is expected. (table 20) This demonstrates the point that we should not consider the numbers as absolutes, but interpret them in the overall setting in order to get the best understanding.

	USER	DEVELOPER	MANAGER
Functionality	0.27	0.34	0.16
Reliability	0.25	0.23	0.28
Usability	0.22	0.16	0.17
Efficiency	0.08	0.09	0.05
Maintainability	0.12	0.11	0.24
Portability	0.06	0.06	0.09
Suitability	0.39	0.35	0.24
Accuracy	0.25	0.28	0.21
Interoperability	0.20	0.14	0.31
Security	0.17	0.24	0.24
Maturity	0.33	0.37	0.23
Fault Tolerance	0.39	0.35	0.45
Recoverability	0.28	0.28	0.33
Understandability	0.26	0.20	0.21
Learnability	0.29	0.32	0.23
Operability	0.36	0.37	0.38
Attractiveness	0.08	0.11	0.19
Time Behavior	0.71	0.78	0.35
Resource Utilization	0.29	0.23	0.65
Analysability	0.21	0.36	0.18
Changeability	0.24	0.23	0.18
Stability	0.25	0.29	0.40
Testability	0.30	0.12	0.24
Adaptability	0.29	0.28	0.17
Installability	0.29	0.33	0.32
Co-existence	0.33	0.12	0.26
Replaceability	0.09	0.27	0.25

Table 20. Comparison of characteristic importance across perspectives

6.3. Collecting Metrics and Reading the Results

The structure of the model chosen for this evaluation corresponds closely to ISO/IEC 9126-1 (2001), and we chose metrics suggested in the ISO/IEC 9126-2 (2003) standard for the product quality evaluation. The metrics' values were provided by the company

quality analysts. The chosen metrics, their definitions and the results obtained from the evaluation are presented in table 21.[6]

In table 21 there are four metrics which have a pair of submetrics associated with them:

- Data exchangeability (User's success attempt based)
- Operational consistency in use
- Error correction in use
- Undoability (User error correction)

After a discussion with company experts it was concluded that the pair of submetrics for the first metric represent alternatives, while the pairs of submetrics associated with the other metrics are complements, and therefore two values have been entered in our table for these metrics. Moreover, it was deemed that the submetrics have equal importance, and are represented with a weight of 0.5 each in our quality model.

Below we present some notes on the collected metrics:

- The results obtained for *Physical accessibility* reflect the accessibility of the software product for a user who has an inability to use a mouse.
- The value of the *Interface appearance customizability* metric reflects the fact that there was no explicit requirement to make the product customizable. The quality subcharacteristic was left in the model in order to determine whether users, managers and developers agree that it is of low significance, and make the appropriate conclusions and amendments if the results of the questionnaires show otherwise.
- The *Hardware environmental adaptability* and *System software environmental adaptability* metrics have values of 1 reflecting adaptability to the environments supported in the requirements definition. If other possible environments are considered, the result would be different.
- The value of the *Ease of installation* metric was set to 0 because there was no available data and the user can not change install operations for his/her convenience.
- The value for the *Continued use of data* metric shows that there is no support for data migration to a previous version. Again, software system migration was not part of the requirements definition for the product. Our model has outlined a feature that can be improved upon, if its importance justifies the spending of additional time and resources.

[6] *The information in the table (with the exception of the Result column) has been extracted from ISO/IEC TR 9126-2 with the permission of Standards Council of Canada, in cooperation with IHS Canada, the official Canadian distributor of ISO publications, License #SCC 08/09 – 035. No further reproduction is permitted without prior written permission*

In addition to the submetrics issue discussed above, there was another issue which required consultation with the company experts. As can be seen in table 21, several quality subcharacteristics have more than one metric associated with them, and this made it necessary to determine the weights of each metric. The company experts provided the results in table 22.

The numbers in the *Result* column of table 21 are given as provided by the company evaluation and have not been truncated or rounded by us.

Looking at the *Interpretation of measured value* column in table 21 we note that most of the metrics fall within the closed interval [0;1]; however, there are unbounded exceptions, which make the result of combining the metrics unbounded as well. This makes it impossible to determine a maximal value for quality. Moreover, a transformation of some of the metrics is necessary in order to be able to compare results obtained for different software products. For example, if we get a small number for Accuracy, this signifies better quality, while in the case of Suitability bigger numbers are better. Holding everything else equal, we could get a higher value for the Functionality metric by either inputting an increased value for Accuracy, or for Suitability. This makes it impossible to compare the Functionality of two alternative products based on the value of the Functionality metric. In order to overcome this and arrive at meaningful results usable for comparisons we have transformed the metrics that provide values which are better the smaller they are by taking the result of subtracting them from 3.4 (the maximal metric value obtained from the evaluation that needs transforming), and then dividing by 3.4 to normalize the value and avoid skewing of results because of the different magnitudes of metrics.

Metrics which have the interpretation "the higher, the better" have also been modified when they are unbounded from above (e.g. the first submetric associated with *Error correction in use*), by dividing them by 1.2 (the maximal occurring value). With regard to metric manipulation it would have been easiest to collect metrics which have consistent interpretation; however choosing a metric which is more meaningful for the product domain and easier to collect in the company setting take precedence over this concern. The list of metrics associated with the quality subcharacteristics after this transformation is presented in table 23. The transformed metrics are rounded and listed with precision of 3 decimal digits. The transformation process might need to be repeated when comparing this product with another one, to take into account new maximal values to use in the transformations.

Figure 31 displays the quality model reflecting the user perspective, implemented in WinBUGS14 (Imperial College of Science, Technology and Medicine & Medical Research Council, 2003).

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Suitability (Functionality)	Functional implementation completeness	How complete is the implementation according to requirement specifications?	0<=X<=1 The closer to 1.0 is the better.	0.95
Accuracy (Functionality)	Accuracy to expectation	Are differences between the actual and reasonable expected results acceptable?	0<=X The closer to 0 is the better.	0.04
Interoperability (Functionality)	Data exchangeability (User's success attempt based)	How often does the end user fail to exchange data between target software and other software?	0<=X<= 1 The closer to 1.0 is the better.	0.999975
Security (Functionality)	Access controllability	How controllable is access to the system?	0<=X<=1 The closer to 1.0 is the better.	0.78
Maturity (Reliability)	Failure resolution	How many failure conditions are resolved?	0<=X<= 1 The closer to 1.0 is better as more failures are resolved.	0.9875
Fault tolerance (Reliability)	Breakdown avoidance	How often the software product causes the break down of the total production environment?	0<=X<= 1 The closer to 1.0 is the better.	0.9875

Table 21. Results of metric collection

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Recoverability (Reliability)	Restorability	How capable is the product in restoring itself after abnormal event or at request?	0<=X<=1 The larger and closer to 1.0 is better, as he [*sic*] product is more capable to restore in defined cases.	0.5
Understandability (Usability)	Evident functions	What proportion of functions (or types of function) can be identified by the user based upon start up conditions?	0<=X<= 1 The closer to 1.0 is the better.	1.0
Learnability (Usability)	Effectiveness of the user documentation and/or help system	What proportion of tasks can be completed correctly after using the user documentation and/or help system?	0<=X<=1 The closer to 1.0 is the better.	0.958
Operability (Usability)	Operational consistency in use	How consistent are the component of the user interface?	0<=X<=1 The closer to 1.0 is the better. 0<=Y The smaller and closer to 0.0 is the better.	a) 0.85 b) 1

Table 21. Results of metric collection (Continued)

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Operability (Usability)	Error correction in use	Can user easily recover his/her error or retry tasks?	0<=X The higher is the better.	a) 1.2
		Can user easily recover his/her input?	0<=X<=1 The closer to 1.0 is the better.	b) 1
Operability (Usability)	Default value availability in use	Can user easily select parameter values for his/her convenient operation?	0<=X<= 1 The closer to 1.0 is the better.	0.74
Operability (Usability)	Self-explanatory error messages	In what proportion of error conditions does the user propose the correct recovery action?	0 <= X <= 1 The closer to 1.0 is the better.	0.857
Operability (Usability)	Undoability (User error correction)	How frequently does the user successfully correct input errors?	0<=X<=1 The closer to 1.0 is the better.	a) 1
		How frequently does the user correctly undo errors?	0 <= Y <= 1 The closer to 1.0 is the better.	b) 0.857
Operability (Usability)	Physical accessibility	What proportion of functions can be accessed by users with physical handicaps?	0 <= X <= 1 The closer to 1.0 is the better.	0

Table 21. Results of metric collection (Continued)

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Attractiveness (Usability)	Interface appearance customizability	What proportion of interface elements can be customised in appearance to the user's satisfaction?	0 <= X <= 1 The closer to 1.0 is the better.	0
Time behaviour (Efficiency)	Response time (Mean time to response)	What is the average wait time the user experiences after issuing a request until the request is completed within a specified system load in terms of concurrent tasks and system utilisation?	0 <= X The nearer to 1.0 and less than 1.0 is the better.	1.14
Time behaviour (Efficiency)	Throughput (Mean amount of throughput)	What is the average number of concurrent tasks the system can handle over a set unit of time?	0 < X The larger is the better.	0.82
Time behaviour (Efficiency)	Turnaround time (Worst case turnaround time ratio)	What is the absolute limit on time required in fulfilling a job task?	0 < X The nearer to 1.0 and less than 1.0 is the better.	1.76
Resource utilisation (Efficiency)	Mean I/O fulfilment ratio	What is the average number of I/O related error messages and failures over a specified length of time and specified utilisation?	0<= X The smaller is the better.	0

Table 21. Results of metric collection (Continued)

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Resource utilisation (Efficiency)	Maximum memory utilization	What is the absolute limit on memory required in fulfilling a function?	0<= X The smaller is the better.	0
Resource utilisation (Efficiency)	Mean of transmission error per time	How many transmission-related error messages were experienced over a set period of time and specified resource utilisation?	0 <= X The smaller is the better.	0
Analysability (Maintainability)	Failure analysis capability	Can user identify specific operation which caused failure? Can maintainer easily find cause of failure?	0<=X<= 1 The closer to 1.0 is the better.	0.9875
Changeability (Maintainability)	Change cycle efficiency	Can the user's problem be solved to his satisfaction within an acceptable time scale?	0<Tav The shorter is the better., except of the number of revised versions was large.	3.4 days
Stability (Maintainability)	Modification impact localisation (Emerging failure after change)	Can user operate software system without failures after maintenance? Can maintainer easily mitigate failures caused by maintenance side effects?	0<=X The smaller and closer to 0 is the better	0.0125

Table 21. Results of metric collection (Continued)

Quality subcharacteristic (parent)	*Metric name*	*Clarification of metric intent*	*Interpretation of measured value*	*Result*
Testability (Maintainability)	Re-test efficiency	Can user and maintainer easily perform operational testing and determine whether the software is ready for operation or not?	0<X The smaller is the better.	0.167
Adaptability (Portability)	Hardware environmental adaptability	Can user or maintainer easily adapt software to environment? Is software system capable enough to adapt itself to operation environment?	0<=X<=1 The larger is the better.	1
Adaptability (Portability)	System software environmental adaptability	Can user or maintainer easily adapt software to environment? Is software system capable enough to adapt itself to operation environment?	0<=X<=1 The larger is the better.	1
Installability (Portability)	Ease of installation	Can user or maintainer easily install software to operation environment?	0<=X<= 1 The closer to 1.0 is the better.	0
Co-existence (Portability)	Available co-existence	How often user encounters any constraints or unexpected failures when operating concurrently with other software?	0<=X The closer to 0 is the better.	0
Replaceability (Portability)	Continued use of data	Can user or maintainer easily continue to use the same data after replacing this software to previous one? Is software system migration going on successfully?	0<= X <=1 The larger is the better.	0

Table 21. Results of metric collection (Continued)

Quality subcharacteristic	*Metric name*	*%*
Operability	Operational consistency in use	21
Operability	Error correction in use	21
Operability	Default value availability in use	21
Operability	Self-explanatory error messages	21
Operability	Undoability (User error correction)	11
Operability	Physical accessibility	5
Time behaviour	Response time (Mean time to response)	28
Time behaviour	Throughput (Mean amount of throughput)	36
Time behaviour	Turnaround time (Worst case turnaround time ratio)	36
Resource utilisation	Mean I/O fulfilment ratio	36
Resource utilisation	Maximum memory utilization	28
Resource utilisation	Mean of transmission error per time	36
Adaptability	Hardware environmental adaptability	50
Adaptability	System software environmental adaptability	50

Table 22. Weights of multiple metrics associated with a single subcharacteristic

name: Quality type: logical link: identity
value: .27*Functionality+.25*Reliability+.22*Usability+.08*Efficiency+.12*Maintainability+.06*Portability

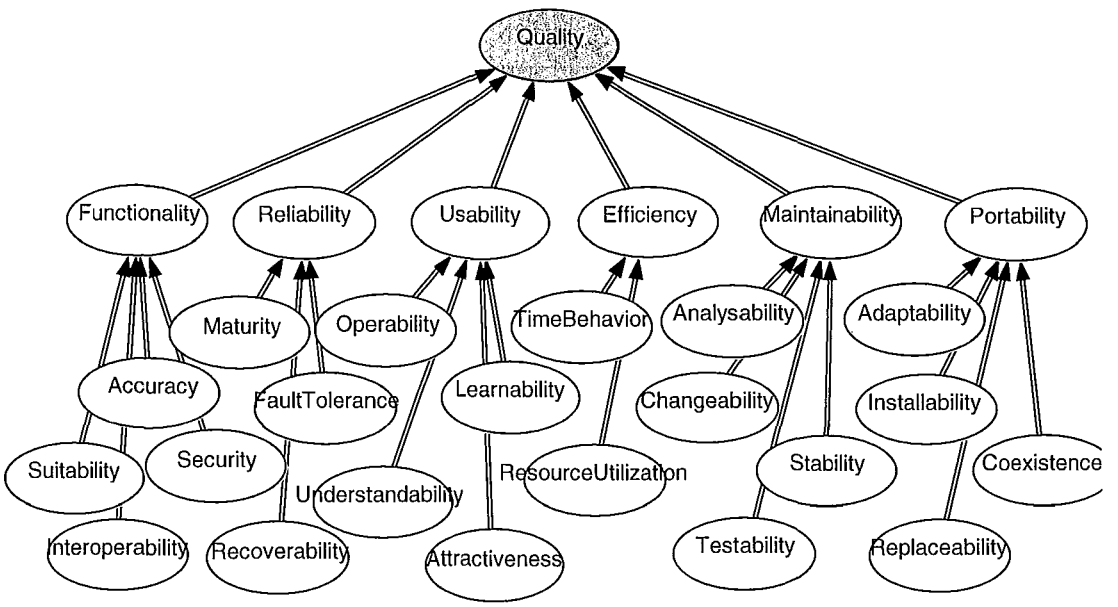


Figure 31. Quality model reflecting the user perspective

Quality subcharacteristic	*Metric calculation*	*Result*
Suitability	Kept the same.	0.95
Accuracy	(3.4 – 0.04)/3.4 = 0.988	0.988
Interoperability	Kept the same.	0.999975
Security	Kept the same.	0.78
Maturity	Kept the same.	0.9875
Fault tolerance	Kept the same.	0.9875
Recoverability	Kept the same.	0.5
Understandability	Kept the same.	1.0
Learnability	Kept the same.	0.958
Operability	Kept the same. (3.4 – 1)/3.4 = 0.706	a) 0.85 b) 0.706
Operability	1.2/1.2 = 1 Kept the same.	a) 1 b) 1
Operability	Kept the same.	0.74
Operability	Kept the same.	0.857
Operability	Kept the same.	a) 1 b) 0.857
Operability	Kept the same.	0
Attractiveness	Kept the same.	0
Time behaviour	(3.4 – 1.14)/3.4 = 0.665	0.665
Time behaviour	0.82/1.2 = 0.683	0.683
Time behaviour	(3.4 – 1.76)/3.4 = 0.482	0.482
Resource utilisation	(3.4 – 0)/3.4 = 1	1
Resource utilisation	(3.4 – 0)/3.4 = 1	1
Resource utilisation	(3.4 – 0)/3.4 = 1	1
Analysability	Kept the same.	0.9875
Changeability	(3.4 – 3.4)/3.4 = 0	0
Stability	(3.4 – 0.0125)/3.4 = 0.996	0.996
Testability	(3.4 – 0.167)/3.4 = 0.951	0.951
Adaptability	Kept the same.	1
Adaptability	Kept the same.	1
Installability	Kept the same.	0
Co-existence	(3.4 – 0)/3.4 = 1	1
Replaceability	Kept the same.	0

Table 23. Transformed quality metrics

In addition, after obtaining the weights for the metrics associated with *Operability*, *Time behaviour*, *Resource utilisation* and *Adaptability* from consultation with company experts we were able to calculate the corresponding metrics. (Table 24)

Quality subcharacteristic	*Metric calculation*	*Result*
Operability	0.21*(0.5*0.85+0.5*0.706) +0.21*(0.5*1+0.5*1) + 0.21*0.74 +0.21*0.857 +0.11*(0.5*1+0.5*0.857) +0.5*0 = 0.810885	0.810885
Time behaviour	0.28*0.665 + 0.36*0.683 + 0.36*0.482 =0.6056	0.6056
Resource utilisation	0.36*1 + 0.28*1 + 0.36*1=1	1
Adaptability	0.5*1 + 0.5*1 = 1	1

Table 24. Metric aggregation

Using the calculated metrics as input in our quality models reflecting the perspectives of *USER*, *DEVELOPER* and *MANAGER* we obtained the following results from using WinBUGS14 (Imperial College & MRC, 2003):

- *USER*
  - Quality 0.835617222
  - Functionality 0.950095
  - Reliability 0.8510000000000001
  - Usability 0.8297386
  - Efficiency 0.7199759999999999
  - Maintainability 0.741675
  - Portability 0.62
- *DEVELOPER*
  - Quality 0.8137821220000001
  - Functionality 0.9363365
  - Reliability 0.8510000000000001
  - Usability 0.80658745
  - Efficiency 0.702368
  - Maintainability 0.7584599999999999
  - Portability 0.4
- *MANAGER*
  - Quality 0.7838401309999998
  - Functionality 0.93267225
  - Reliability 0.8365
  - Usability 0.7384763
  - Efficiency 0.8619600000000001
  - Maintainability 0.80439
  - Portability 0.43

Table 25 presents a side-by-side comparison of the rounded results.

	USER	DEVELOPER	MANAGER
Overall Quality	0.84	0.81	0.78
Functionality	0.95	0.94	0.93
Reliability	0.85	0.85	0.84
Usability	0.83	0.81	0.74
Efficiency	0.72	0.70	0.86
Maintainability	0.74	0.76	0.80
Portability	0.62	0.40	0.43

Table 25. Comparison of evaluation results according to perspective

Looking at table 25 we notice several things with regard to the evaluation of the company's product:

- Reliability and Functionality are the quality characteristics that can garner a consensus from users, developers and managers.
- Developers thought the least of Efficiency and Portability compared with users and managers.
- Users gave the highest scores to Functionality, Reliability and Usability, followed closely by developers.
- Managers gave the highest score of any group to Efficiency and Maintainability.
- Portability got its highest score from users, and lowest from developers, and here we observe a big quantitative difference.
- Users have the highest score for overall product quality, while managers have the lowest; however the numbers are relatively close across the perspectives, which would facilitate communication among them.

Because of the lack of previous quality evaluation data and experience in this product domain it was not possible to obtain qualitative rankings for the quality characteristics. It is therefore not possible to determine whether the calculated values correspond to Excellent, Good or Poor quality. However, it is possible to compare the results from the different perspectives and see which group attributes more quality to a certain quality factor. It would also be possible to compare the results acquired from two similar projects, as well as to perform sensitivity analyses to see which characteristics need to be changed to make the perception of quality consistent across perspectives, or to improve it significantly for a certain group. The results of the case study presented in this chapter have been collected in a knowledge base that would enable parameter learning and SCT elicitation for future evaluation processes.

Performing sensitivity analyses is important for validating the obtained results. We have not performed sensitivity analyses for this data because it would be difficult to interpret the meaning of the observed quantitative changes without the qualitative rankings.

6.4. Summary

This chapter saw the proposed methodology applied to a case study. We used the slightly modified ISO/IEC 9126-1 (2001) model, omitting only the compliance characteristics from the model. Parameter elicitation was performed using the mutual comparison method from the perspective of user, developer and manager, and a discussion of the differences in perception was presented. Metrics following the ISO/IEC 9126-2 (2003) standard were collected and transformed in order to make possible the comparison of quality characteristic values across perspectives. It was not possible to construct Scale Calibration Tables because of the lack of previous data and expert knowledge on the subject; however we have outlined some applications for the obtained results.

7. Conclusion

The methodology outlined in this work has several key advantages that distinguish it from other general approaches to software quality evaluation. Most of the quality models presented in the literature do not accommodate structural change, while our methodology allows model customization. In addition, it indicates how to take advantage of newly available information and data in order to fine-tune the model. In essence, it consists of a set of hierarchies, with overall quality always being at the root of the tree, and the particular hierarchy is determined by the intention of the evaluation. Morris (2007) points out that the general approaches to software evaluation are not suitable for the discussion of tradeoffs in the requirements acquisition process. Even though he made this point with regard to commercial off-the-shelf software, we believe that it is relevant for the evaluation of any type of software. The approach we suggest accommodates tradeoff consideration, and further promotes decision-making by considering quality from different perspectives.

Depending on the product domain and the perspective of the evaluator (developer, manager, user) the importance of quality characteristics and subcharacteristics and their contribution to overall quality is different. Therefore, when we want to evaluate the quality of a specific product, we need to customize the model taking into account the relative weights that the quality characteristics have. These weights can influence the decision where to focus limited resources when trying to improve product quality. For example, the manager might have to decide whether to add functionality, or to devote more time to increasing the reliability of the product. In the case where both activities will lead to an equal increase in overall quality from the manager's perspective, he might take into consideration the developer's perspective or the user's perspective to finalize his decision. Having separate models for the different viewpoints will also facilitate discussions in the company as to why people view the importance of contributing factors differently and are intuitively lead to focus more time and energy on certain activities. If the perspectives can be reconciled, this will lead to increased levels of overall contentedness with the finished product.

7.1. Summary

This work provides a comprehensive view of the steps necessary to evaluate a software product. It compares and combines several alternative approaches to evaluation. We began with an overview of software quality concepts and a demonstration of the importance of software quality evaluation through the project statistics supplied. Then we established the relevance of the product-oriented approach to software quality evaluation, as the quality of a finished software product cannot be guaranteed by following best process practices. After presenting an overview of several existing quality models that

have gained prominence and discussing their advantages and weaknesses we came to the conclusion that ISO/IEC 9126-1 (2001) is the best basis for conducting a product quality evaluation. We consider the flexibility of the model as its greatest advantage, allowing the evaluator to customize it by adding desirable quality characteristics and omitting insignificant ones. Combining this product quality model with BBN use makes possible the automation of some evaluation activities (e.g. parameter learning), and formalizes others by explicitly considering expert opinion (e.g. determining the utility associated with different outcomes). We have attempted to condense the expert knowledge available in the literature regarding the model structure in the generic model that we presented in Chapter 4. It combines the quality characteristics of the ISO/IEC 9126-1 (2001) external quality model with all characteristics from the McCall and Boehm quality models, thus providing a convenient source of reference for choosing which characteristics to evaluate. In addition, we discuss a way of transforming the resulting model into a form suitable for BBN use.

The methodology for evaluating software quality that we have suggested provides alternatives for eliciting the software product quality model structure and parameters, for collecting metrics and interpreting the obtained results. Understanding the importance of model customization is paramount for appreciating the value of our work. The possibility to customize, to adjust the quality model and evaluation process to best utilize the available data and expert knowledge, promotes more accurate evaluation results, and therefore, better company project performance. The application of the proposed methodology to a case study is detailed in chapter 6. Parameter elicitation was performed from the perspectives of user, developer and manager, and the differences in perception were outlined. Understanding these differences facilitates dialogue between the parties, and contributes to a better work environment and more efficient company operation.

Working with the collected metrics presented a problem, as they had different interpretations which necessitated metric transformation in order to be able to make valid inferences. The comparison of quality characteristic values across perspectives exhibited some interesting results. Among the potential applications for the results obtained in the case study are:

- Comparing the results from the different perspectives to see which group values a certain quality factor more;
- Comparing the results acquired from the case study with results from a similar project;
- Performing sensitivity analyses to see whether the results can be considered valid even if the obtained measures for the quality characteristics are somewhat imprecise;
- Performing sensitivity analyses to see which characteristics need to be changed in order to make the perception of quality consistent across perspectives;
- Performing sensitivity analyses to see which characteristics need to be changed in order to improve quality significantly for a certain group;

- Adding the results collected in a knowledge base that would enable parameter learning and SCT elicitation for future evaluation processes.

Appendix A was created as a convenient knowledge base assisting the evaluator in the choice of quality characteristics to include in the model, and in addition provides suggestions for metrics.

7.2. Related Work

There are a number of works that have been referenced and discuss similar ideas, yet they are taking a more customized approach or focusing on just one aspect of the evaluation process. The works that come close to our views are (Stefani et al., 2003) and (Behkamal et al., 2009), both of which evaluate the quality of E-commerce products using ISO/IEC 9126-1 as a basis and utilizing BBN tools. Both are customized to reflect domain specifics and can be seen as examples of domain specific application of the more general approach presented here. Stefani et al. (2003) used an earlier version of the ISO/IEC 9126-1 standard and this led to some perceived discrepancies. For example, the authors model quality from the user's perspective, with questionnaires that are filled in to document use of the product in a real setting versus a simulated environment, and therefore we believe that it is more appropriate to employ the quality in use model and not the external quality model. It would be interesting to see whether the two models would provide different insights or will be sufficiently similar. The creation of an all-encompassing generic model and its transformation for BBN use are among the novel ideas presented in this work, and their usefulness will have to be confirmed by future case studies.

7.3. Reasoning behind BBN Use

In this methodology it was suggested that the quality model is used in conjunction with a BBN software tool. There are many potential rewards in this arrangement, and some of them are described in this section. One possibility is the creation of an add-on module to the software specifically dedicated to product-oriented software quality evaluation. The add-on module will present the evaluator with either the generic model proposed above with the potential to remove nodes, or with the ISO/IEC 9126-1 (2001) model with the potential to add new nodes automatically in their correct place. The definitions and other relevant information regarding the quality characteristics (as contained in Appendix A) can be made available as a reference point. The dependencies between the nodes can be quantified either deterministically or probabilistically using questionnaires which are also part of the module and the necessary calculations can be performed automatically after the experts fill in their opinions. The next step would be to suggest operationalization of the bottom layer of the model according to the product domain, or to suggest metrics to

be used for the quantification of the bottom layer. There has been work on automating the collection of internal metrics for use with ISO/IEC 9126 (Lincke & Löwe, 2006) and this might enable the establishment of mappings between internal and external metrics, which will be useful in predicting external quality before external metrics are available. Many BBN software tools provide the option of performing sensitivity analysis and this makes it possible to evaluate tradeoffs. Some tools additionally offer the capability to extend the model to a decision model, taking into account the utility associated with different outcomes, and can be used for further facilitation of the decision-making process. As we have already pointed out, if the evaluators have sufficient data at their disposal, there are BBN software tools that provide as features structure learning and parameter learning. The use of Bayesian Net tools can therefore significantly decrease the time that company experts need to dedicate to the evaluation process.

7.4. Directions for Future Work

As directions for future work we consider the following:

- Addition of further criteria to the generic model and Appendix A;
- Providing more suggested metrics in Appendix A;
- Verifying the conditional independence property for the generic model we constructed using the ideas of Morris and Beling (2004);
- Comparing the results acquired by using models obtained by the Morris and Beling transformation with results from models directly reflecting the hierarchies presented in the literature;
- Comparing probabilistic model results acquired from several perspectives for the same product;
- Creating a BBN module dedicated to software product quality evaluation, as outlined in section 7.3.

Appendix A - Compendium of Quality Characteristics

In Appendix A we present a glossary of non-functional requirements excerpted from (Colin et al., 2008) and extended with additional definitions, suggested metrics for the evaluation of the listed quality characteristics, as well as a specification of their relative position in a quality model. The positioning is determined by their parents[7] (these are the factors that the characteristic contributes to), children (characteristics that contribute to its higher quality) and siblings (other characteristics that influence its parents' quality). This document is intended as a living document, which should be updated regularly to include new definitions and insights.

The 151 candidate metrics proposed by Boehm (Selby, 2007) are not listed here as suggested metrics, because they are established for the quality evaluation of FORTRAN code, and are therefore not generally applicable. We have attempted to include only generally applicable metrics in this appendix, and therefore some quality characteristics do not have suggested metrics at this time.

The information added to the original glossary is italicized. A reference is provided after the last quality characteristic of a list coming from the same source. Whenever the quality characteristic is already listed with a different source, it is not repeated.

Abstractness
The degree to which a system or component performs only the necessary functions relevant to a particular purpose.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Access control
Access control is the ability to permit or deny the use of particular resource to particular user based on the user credentials.
Related terms: Accessibility, Auditing, Integrity
Example: Driver and user authentication
Parent: Integrity (McCall et al., 1977)
Siblings: Access audit (McCall et al., 1977)
Children:

[7] The terms parents and children used here do not correspond to parents and children as defined in Bayesian Net models.

Similar:
Suggested metric: consider (McCall et al., 1977)

Accessibility
The degree to which the software system protects system functions or service from being denied to the user.
Related terms: Access control, Efficiency
Example: Additional services for handicap persons
Parent: Efficiency, Human Engineering, Testability (Selby, 2007)
Siblings: Accountability, Device Efficiency, Robustness/Integrity, Communicativeness, Self-Descriptiveness, Structuredness (Selby, 2007)
Children:
Similar:
Suggested metric:

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, and so on. Code used for probes is preferably invoked by conditional assembly techniques to eliminate the additional instruction words or added execution times when the measurements are not needed. (Selby, 2007)
Related terms:
Example:
Parent: Efficiency, Testability (Selby, 2007)
Siblings: Accessibility, Device Efficiency, Communicativeness, Structuredness, Self-descriptiveness (Selby, 2007)
Children:
Similar:
Suggested metric:

Accuracy
A quantitative measure of the magnitude of error [IEE90].
Related terms: Reliability
Example: The distance between predecessor and successor cycabs
Parent: Reliability (McCall et al., 1977), (Selby, 2007), Functionality (ISO/IEC, 2001)
Siblings: Consistency, Error tolerance, Simplicity (McCall et al., 1977), Suitability, Interoperability, Security, Functionality Compliance (ISO/IEC, 2001), Self-containedness, Completeness, Robustness/Integrity (Selby, 2007)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (ISO/IEC, 2003)

Adaptability
The ease with which software satisfies differing system constraints and user needs. An adaptable software system can tolerate changes in its environment without external intervention [EM87].
Related terms:
Example: Using cycabs at different routes
Parent: Portability (ISO/IEC, 2001)
Siblings: Installability, Co-existence, Replaceability, Portability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003), (Khosravi & Gueheneuc, 2004)

Analyzability
Attributes of software that relate to the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified [ISO01].
Related terms:
Example: Sensor changing
Parent: Maintainability (ISO/IEC, 2001)
Siblings: Changeability, Stability, Testability, Maintainability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Anonymity
The degree to which a software system or component allows for or supports anonymous transactions.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

As-is utility
How well (easily, reliably, efficiently) can I use it as is? (Selby, 2007)
Related terms:
Example:
Parent: Product quality (Selby, 2007)
Siblings: Portability, Maintainability (Selby, 2007)
Children: Reliability, Human engineering, Efficiency (Selby, 2007)
Similar:

Suggested metric: derived from children's (Selby, 2007)

Attractiveness
The capability of the software product to be attractive to the user. (ISO/IEC, 2001)
Related terms:
Example: the use of colour, the nature of the graphical design (ISO/IEC, 2001)
Parent: Usability (ISO/IEC, 2001)
Siblings: Understandability, Learnability, Operability, Usability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Auditing
The degree to which a software system records information concerning transactions performed within the system.
Related terms: Access control, Integrity
Example: Black box
Parent: Integrity (McCall et al., 1977)
Siblings: Access control (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977) (Access audit)

Augmentability
Code possesses the characteristic of augmentability to the extent that it can easily accommodate expansion in component computational functions or data storage requirements. This is a necessary characteristic for modifiability. (Selby, 2007)
Related terms: Maintainability (Selby, 2007)
Example:
Parent: Modifiability (Selby, 2007)
Siblings: Structuredness (Selby, 2007)
Children:
Similar: Changeability (Selby, 2007)
Suggested metric:

Availability
The ability of a system to be operable and in a committable state when required to use. Availability is the proportion of time a system is in a functioning condition and is ready to provide correct services.
Related terms:
Example: Brakes
Parent:
Siblings:

Children:
Similar:
Suggested metric:

Capacity
A measure of the amount of work a system can perform [BKLW95].
Related terms:
Example: Seating capacity for persons, stamina of battery
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Changeability
Attributes of software that relate to the effort needed for modification, fault removal or for environmental change [ISO01].
Related terms: Maintainability
Example:
Parent: Maintainability (ISO/IEC, 2001)
Siblings: Analysability, Stability, Testability, Maintainability Compliance (ISO/IEC, 2001)
Children:
Similar: Modifiability
Suggested metric: consider (ISO/IEC, 2003)

Co-existence
The capability of the software product to co-exist with other independent software in a common environment sharing common resources. (ISO/IEC, 2001)
Related terms:
Example:
Parent: Portability (ISO/IEC, 2001)
Siblings: Adaptability, Installability, Replaceability, Portability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Communicativeness
Code possesses the characteristic of communicativeness to the extent that it facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and useful. Communicativeness is necessary for testability and human engineering. (Selby, 2007)
Related terms: As-is Utility, Maintainability (Selby, 2007)

Example:
Parent: Human Engineering, Testability (Selby, 2007), Usability (McCall et al., 1977)
Siblings: Integrity, Accessibility, Accountability, Structuredness, Self_descriptivness (Selby, 2007), Operability, Training (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Communications commonality
Those attributes of the software that provide the use of standard protocols and interface routines. (McCall et al., 1977)
Related terms:
Example:
Parent: Interoperability (McCall et al., 1977)
Siblings: Modularity, Data commonality (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Compactness
The degree to which a system or component makes efficient use of its data storage space i.e. occupies a small volume.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Compatibility
The ability of two or more systems or components to perform their required functions while having the same hardware or software environment [IEE90].
Related terms:
Example: Sensors
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Completeness
The degree to which all the parts of a software system or component are present and each of its parts is fully specified and developed. This means that if the code calls a subroutine

from an external library, the software package must provide reference to that library and all required parameters must be passed and all required input data must be available [BBK+78].
Related terms: Correctness, Reliability
Example:
Parent: Correctness (McCall et al., 1977), Reliability (Selby, 2007)
Siblings: Traceability, Consistency (McCall et al., 1977), Self-containedness, Accuracy, Robustness/Integrity (Selby, 2007)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

Complexity
The degree to which a system or component has a design or implementation that is difficult to understand and verify [IEE90].
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric: consider (Khosravi & Gueheneuc, 2004)

Compliance
Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions [ISO01].
Related terms:
Example: Adherence to specific rules
Parent: Functionality/Reliability/Usability/Efficiency/Maintainability/Portability (ISO/IEC, 2001)
Siblings:
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Conciseness
The degree to which a software system or component has no excessive information present [MRW77].
Related terms: Maintainability, Understandability
Example: Unavailability of unused methods and dummy variables
Parent: Maintainability (McCall et al., 1977), Understandability (Selby, 2007)
Siblings: Simplicity, Self-descriptiveness, Modularity, Consistency (McCall et al., 1977), Structuredness, Legibility (Selby, 2007)
Children:

Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

Confidentiality
The nonoccurrence of the unauthorized disclosure of information [BKLW95].
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Consistency
The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component [IEE90].
Related terms: Correctness, Reliability
Example: Naming conventions
Parent: Correctness, Reliability, Maintainability (McCall et al., 1977), Understandability (Selby, 2007)
Siblings: Traceability, Completeness, Accuracy, Error Tolerance, Simplicity, Conciseness, Modularity, Self-Descriptiveness (McCall et al., 1977), Self-containedness, Robustness/Integrity, Structuredness, Legibility (Selby, 2007)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Correctness
The extent to which a program satisfies its specifications and fulfills the customer's mission objectives [MRW77].
Related terms: Traceability, Completeness, Consistency
Example:
Parent:
Siblings:
Children: Traceability, Consistency, Completeness (McCall et al., 1977)
Similar:
Suggested metric: derivable from children, consider (Khosravi & Gueheneuc, 2004)

Data commonality
Those attributes of the software that provide the use of standard data representations. (McCall et al., 1977)
Related terms:
Example:
Parent: Interoperability (McCall et al., 1977)

Siblings: Modularity, Communications commonality (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Denial of service
The degree to which a software system or component prevents the interference or disruption of system services to the user.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Dependability
The property of a system such that reliance can justifiably be placed on the service it delivers.
Related terms:
Example: Brake, stamina of battery
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Device independence
Code possesses the characteristic of device independence to the extent it can be executed on computer hardware configurations other than its current one. Clearly, this characteristic is a necessary condition for portability. (Selby, 2007)
Related terms:
Example:
Parent: Portability (Selby, 2007)
Siblings: Self-containedness (Selby, 2007)
Children:
Similar:
Suggested metric:

Effectiveness
The degree to which a system's features and capabilities meet the user's needs.
Related terms:
Example:
Parent:

Siblings:
Children:
Similar:
Suggested metric:

Efficiency
The degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks) [IEE90].
Related terms: Accessibility
Example: Power consumption
Parent: Product Quality, As-is Utility (Selby, 2007)
Siblings: Functionality, Reliability, Usability, Maintainability, Portability (ISO/IEC, 2001), Human Engineering (Selby, 2007)
Children: Execution Efficiency, Storage Efficiency (McCall et al., 1977), Time Behaviour, Resource Utilisation, Efficiency Compliance (ISO/IEC, 2001), Accountability, Device Efficiency, Accessibility (Selby, 2007)
Similar:
Suggested metric: derivable from children (McCall et al., 1977), (ISO/IEC, 2003), consider (Khosravi & Gueheneuc, 2004)

Evolvability/Upgradability
The ease with which a system or component can be modified to take advantage of new software or hardware technologies.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Exception Handling / Error Tolerance
The capability of system to handle the occurrence of some condition that changes the normal flow of execution. For example, incorrect input, communication problems, page fault and etc [IEE90].
Related terms: Reliability
Example: Packet collision
Parent: Reliability (McCall et al., 1977)
Siblings: Consistency, Accuracy, Simplicity (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Execution Efficiency
Those attributes of the software that provide for minimum processing time. (McCall et al., 1977)
Related terms:
Example:
Parent: Efficiency (McCall et al., 1977)
Siblings: Storage Efficiency (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Extendibility/Expandability
The ease with which a system or component can be modified to increase its storage or functional capacity [IEE90].
Related terms: Flexibility
Example:
Parent: Flexibility (McCall et al., 1977)
Siblings: Modularity, Generality, Self-descriptiveness (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Fault-Tolerance
The capability to continue normal operation after the occurrence of an error, with as minimal human intervention as possible. Fault-tolerant describes a computer system or component designed so that, in the event that a component fails, a backup component or procedure can immediately take its place with no loss of service.
Related terms:
Example:
Parent: Reliability (ISO/IEC, 2001)
Siblings: Maturity, Recoverability, Reliability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Fidelity
The degree of similarity between a model and the system properties being modeled [IEE90].
Related terms:
Example: Usage of system on multiple type of vehicles
Parent:
Siblings:
Children:
Similar:

Suggested metric:

Flexibility
The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEE90].
Related terms: Expandability, Generality
Example:
Parent:
Siblings:
Children: Modularity, Generality, Expandability, Self-Descriptiveness (McCall et al., 1977)
Similar:
Suggested metric: derivable from children

Functionality
A set of attributes that relate to the existence of a set of functions and their specified properties [ISO01].
Related terms: Accuracy, Compliance, Interoperability, Security, Suitability
Example:
Parent: Product Quality
Siblings: Reliability, Usability, Efficiency, Maintainability, Portability (ISO/IEC, 2001)
Children: Suitability, Accuracy, Interoperability, Security, Functionality Compliance (ISO/IEC, 2001)
Similar:
Suggested metric: derivable from children

Generality
The degree to which a system or component performs a broad range of functions [IEE90].
Related terms: Flexibility, Reusability
Example: Movement of vehicle in both directions
Relevant:
Parent: Reusability, Flexibility (McCall et al., 1977)
Siblings: Modularity, Software System Independence, Machine Independence, Self-Descriptiveness, Expandability (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

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. This characteristic implies accessibility, robustness, and communicativeness. (Selby, 2007)
Related terms:

Example:
Parent: As-is Utility (Selby, 2007)
Siblings: Reliability, Efficiency (Selby, 2007)
Children: Integrity, Accessibility, Communicativeness (Selby, 2007)
Similar:
Suggested metric:

Incompleteness
The degree to which all the parts of a software system or component are not present and each of its parts is not fully specified or developed.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Install-ability
Attributes of software that relate to the effort needed to install the software in a specified environment [ISO01].
Related terms: Portability
Example:
Parent: Portability (ISO/IEC, 2001)
Siblings: Adaptability, Co-existence, Replaceability, Portability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Instrumentation
Those attributes of the software that provide for the measurement of usage or identification of errors. (McCall et al., 1977)
Related terms:
Example:
Parent: Testability (McCall et al., 1977)
Siblings: Simplicity, Modularity, Self-Descriptiveness (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Integrity
Absence of improper system alterations.
Related terms: Access control, Auditing, Reliability

Example:
Parent: Reliability, Human Engineering (Selby, 2007)
Siblings: Self-containedness, Accuracy, Completeness, Consistency, Accessibility, Communicativeness (Selby, 2007)
Children: Access Control, Access Audit (McCall et al., 1977)
Similar: Robustness (Selby, 2007)
Suggested metric: derivable from children

Interoperability
The ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEE90].
Related terms: Commonality, Modularity
Example: Interaction among heterogeneous vehicles
Parent: Functionality (ISO/IEC, 2001)
Siblings: Suitability, Accuracy, Security, Functionality Compliance (ISO/IEC, 2001)
Children: Modularity, Communications Commonality, Data Commonality (McCall et al., 1977)
Similar:
Suggested metric: derivable from children, consider (ISO/IEC, 2003)

Latency
The length of time it takes to respond to an event [BKLW95].
Related terms:
Example: Time for brake
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Legibility
Code possesses the characteristic of legibility to the extent that its function is easily discerned by reading the code. Legibility is necessary for understandability. (Selby, 2007)
Related terms: Maintainability (Selby, 2007)
Example: complex expressions have mnemonic variable names and parentheses even if unnecessary (Selby, 2007)
Parent: Understandability (Selby, 2007)
Siblings: Consistency, Conciseness, Structuredness, Self-descriptiveness (Selby, 2007)
Children:
Similar:
Suggested metric:

Logging
The capability to register events of a given component, for traceability and/or debugging purposes.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Machine Independence
Those attributes of the software that determine its dependency on the hardware system. (McCall et al., 1977)
Related terms:
Example:
Parent: Portability, Reusability (McCall et al., 1977)
Siblings: Self-Descriptiveness, Software System Independence, Generality, Modularity (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Maintainability
The ease with which a software system or component can be modified to correct faults and improve performance [MRW77].
Related terms: Conciseness, Modifiability, Modularity, Simplicity, Testability, Understandability
Example: Software/Hardware drivers
Parent: Product Quality
Siblings: Functionality, Reliability, Usability, Efficiency, Portability (ISO/IEC, 2001), As-is Utility (Selby, 2007)
Children: Consistency, Simplicity, Conciseness, Modularity, Self-Descriptiveness (McCall et al., 1977), Analysability, Changeability, Stability, Testability, Maintainability Compliance (ISO/IEC, 2001), Understandability, Modifiability (Selby, 2007)
Similar:
Suggested metric: derivable from children (McCall et al., 1977), (ISO/IEC, 2003)

Maturity
Attributes of software that relate to the frequency of failure by faults in the software [ISO01].
Related terms:
Example:
Parent: Reliability (ISO/IEC, 2001)

Siblings: Fault Tolerance, Recoverability, Reliability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003), (Khosravi & Gueheneuc, 2004)

Modifiability
The degree to which a system or component facilitates the incorporation of changes, once the nature of the desired change has been determined [BBK+78].
Related terms: Maintainability, Structuredeness,
Example:
Parent: Maintainability (Selby, 2007)
Siblings: Testability, Understandability (Selby, 2007)
Children: Structuredness, Augmentability (Selby, 2007)
Similar: Changeability
Suggested metric:

Modularity
The capability to divide a system into local modules for understandability and reusability purposes.
Related terms:
Example: Classes, Components, Modules
Parent: Maintainability, Testability, Reusability, Interoperability, Flexibility, Portability (McCall et al., 1977)
Siblings: Consistency, Simplicity, Conciseness, Self-Descriptiveness, Instrumentation, Generality, Software System Independence, Machine Independence, Communications Commonality, Data Commonality, Expandability (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), consider (Khosravi & Gueheneuc, 2004)

Multi Access
The capability of having several users using the system at the same time.
Related terms:
Example: Supervisors intervention
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Openness
The degree to which a system or component complies with standards.
Related terms:

Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Operability
The ease of operating the software [DW88].
Related terms: Usability
Example: Joysticks
Parent: Usability (McCall et al., 1977), (ISO/IEC, 2001)
Siblings: Training, Communicativeness (McCall et al., 1977), Understandability, Learnability, Attractiveness, Usability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (ISO/IEC, 2003), (Khosravi & Gueheneuc, 2004)

Portability
The ease with which a system or component can be transferred from one hardware or software environment to another [IEE90].
Related terms:
Example: Components usability from one vehicle to another
Parent: Product Quality
Siblings: Functionality, Reliability, Usability, Efficiency, Maintainability (ISO/IEC, 2001), As-is Utility (Selby, 2007)
Children: Modularity, Self-Descriptiveness, Machine Independence, Software System Independence (McCall et al., 1977), Adaptability, Installability, Co-existence, Replaceability, Portability Compliance (ISO/IEC, 2001), Device Independence, Self-containedness (Selby, 2007)
Similar:
Suggested metric: derivable from children (McCall et al., 1977), (ISO/IEC, 2003), consider (Khosravi & Gueheneuc, 2004)

Quantifiability
The quality of being measurable.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Readability
The degree to which a system's functions and those of its component statements can be easily discerned by reading the associated source code.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric: consider (Khosravi & Gueheneuc, 2004)

Recovery
The restoration of a system, program, database, or other system resource to a prior state following a failure or externally caused disaster; for example, the restoration of a database to a point at which processing can be resumed following a system failure [IEE90].
Related terms:
Example: Cycab's direction
Parent: Reliability (ISO/IEC, 2001)
Siblings: Maturity, Fault Tolerance, Reliability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Reliability
The ability of a system to perform and maintain its functions in routine circumstances as well as in hostile or unexpected circumstances with required precision [MRW77].
Related terms: Accuracy, Completeness, Consistency, Error Tolerance, Integrity, Robustness
Example:
Parent: Product Quality, As-is Utility (Selby, 2007)
Siblings: Functionality, Usability, Efficiency, Maintainability, Portability (ISO/IEC, 2001), Human Engineering (Selby, 2007)
Children: Error Tolerance, Consistency, Accuracy, Simplicity (McCall et al., 1977), Maturity, Fault Tolerance, Recoverability, Reliability Compliance (ISO/IEC, 2001), Self-containedness, Completeness, Robustness/Integrity (Selby, 2007)
Similar:
Suggested metric: derivable from children (McCall et al., 1977), (ISO/IEC, 2003), consider (Khosravi & Gueheneuc, 2004)

Replaceability
Attributes of software that relate to the opportunity and effort of using it in the place and environment of other software [ISO01].

Related terms: Portability
Example:
Parent: Portability (ISO/IEC, 2001)
Siblings: Adaptability, Installability, Co-existence, Portability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Requirement
A requirement is a singular documented need of what a particular product or service should be or do. It is most commonly used in a formal sense in systems engineering or software engineering. It is a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Resource Behavior
Attributes of software that relate to the amount of resources used and the duration of such use in performing its function [ISO01].
Related terms: Efficiency
Example:
Parent: Efficiency (ISO/IEC, 2001)
Siblings: Time Behaviour, Efficiency Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Response Time
The capability to react promptly to external stimuli confronting the processing time with performance requirements. The response time may be calculated between the instant at which an operator at a terminal enters a request for a response from a computer and the instant at which the first character of the response is received at a terminal.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:

Suggested metric:

Responsiveness
The degree to which a software system or component has incorporated the user's requirements.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Reusability
The extent to which a program (or parts of a program) can be reused in other applications [MRW77].
Related terms: Generality, Modularity
Example:
Parent:
Siblings:
Children: Generality, Modularity, Software System Independence, Machine Independence, Self-Descriptiveness (McCall et al., 1977)
Similar:
Suggested metric: derivable from children

Robustness
The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [IEE90].
Related terms: Reliability
Example:
Parent: Reliability, Human Engineering (Selby, 2007)
Siblings: Self-containedness, Accuracy, Completeness, Consistency, Accessibility, Communicativeness (Selby, 2007)
Children:
Similar: Integrity (Selby, 2007)
Suggested metric: consider (Khosravi & Gueheneuc, 2004)

Safety
The ability of the system to operate without (internal) catastrophic failure [BKLW95].
Related terms:
Example:
Parent:
Siblings:
Children:

Similar:
Suggested metric:

Scalability
It is a desirable property of a system which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric: consider (Khosravi & Gueheneuc, 2004)

Security
The capability of the system to protect itself against (external) accidental or deliberate intrusion.
Related terms:
Example:
Parent: Functionality (ISO/IEC, 2001)
Siblings: Suitability, Accuracy, Interoperability, Functionality Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Self-containedness
Code possesses the characteristic of self-containedness to the extent that it performs all its explicit and implicit functions within itself. Examples of implicit functions are initialization, input checking, and diagnostics. (Selby, 2007)
Related terms:
Example:
Parent: Portability, Reliability (Selby, 2007)
Siblings: Device independence, Completeness, Accuracy, Integrity, Consistency (Selby, 2007)
Children:
Similar:
Suggested metric:

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. Commentary and traceability of previous changes by transforming previous versions of code into nonexecutable but present (or available by macro calls) code are some of the ways of providing this characteristic. Self-descriptiveness is necessary for both testability and understandability. (Selby, 2007)
Related terms: Maintainability (Selby, 2007)
Example:
Parent: Testability, Understandability (Selby, 2007), Maintainability, Flexibility, Portability, Reusability (McCall et al., 1977)
Siblings: Communicativeness, Accessibility, Accountability, Structuredness, Consistency, Conciseness, Legibility (Selby, 2007), Modularity, Simplicity, Expandability, Software System Independence, Machine Independence, Generality, Instrumentation (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Simplicity
The degree to which a system or component has a design and implementation that is straightforward and easy to understand [IEE90].
Related terms: Maintainability
Example:
Parent: Testability, Maintainability, Reliability (McCall et al., 1977)
Siblings: Consistency, Conciseness, Modularity, Self-Descriptiveness, Instrumentation, Accuracy, Error Tolerance (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

Software System Independence
Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.). (McCall et al., 1977)
Related terms:
Example:
Parent: Portability, Reusability (McCall et al., 1977)
Siblings: Self-Descriptiveness, Machine Independence, Generality, Modularity (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

Speed
The rate at which a software system or component performs its functions.

Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Stability
Attributes of software that relate to the risk of unexpected effect of modifications.
Related terms: Maintainability
Example:
Parent: Maintainability (ISO/IEC, 2001)
Siblings: Analysability, Changeability, Testability, Maintainability Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Storage Efficiency
Those attributes of the software that provide for minimum storage requirements during operation. (McCall et al., 1977)
Related terms:
Example:
Parent: Efficiency (McCall et al., 1977)
Siblings: Execution Efficiency (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Structuredness
The degree to which a system or component possesses a definite pattern of organization of its interdependent parts [BBK+78].
Related terms: Modifiability, Testability, Understandability
Example:
Parent: Modifiability, Testability, Understandability (Selby, 2007)
Siblings: Augmentability, Accountability, Accessibility, Communicativeness, Self-descriptiveness, Consistency, Conciseness, Legibility (Selby, 2007)
Children:
Similar:
Suggested metric: consider (Khosravi & Gueheneuc, 2004)

Suitability
Attributes of software that bare on the provision of right or agreed results or effects [ISO01].
Related terms:
Example:
Parent: Functionality (ISO/IEC, 2001)
Siblings: Accuracy, Interoperability, Security, Functionality Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Survivability
The degree to which essential functions are still available even though some part of the system is down [DW88].
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Testability
The effort required to test a program to ensure that it performs its intended function [MRW77].
Related terms: Maintainability, Modularity, Simplicity, Structuredness
Example:
Parent: Maintainability (ISO/IEC, 2001)
Siblings: Analysability, Changeability, Stability, Maintainability Compliance (ISO/IEC, 2001), Understandability, Modifiability (Selby, 2007)
Children: Simplicity, Modularity, Instrumentation, Self-Descriptiveness (McCall et al., 1977), Accountability, Accessibility, Communicativeness, Structuredness (Selby, 2007)
Similar:
Suggested metric: derivable from children, consider (ISO/IEC, 2003)

Throughput
Throughput is the amount of work that an application can do in a given time period.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:

Suggested metric:

Time behavior
Attributes of software that relate to response and processing times and on throughput rates in performing its function [ISO01].
Related terms: Efficiency, Response time, Throughput
Example:
Parent: Efficiency (ISO/IEC, 2001)
Siblings: Resource Utilisation, Efficiency Compliance (ISO/IEC, 2001)
Children:
Similar:
Suggested metric: consider (ISO/IEC, 2003)

Traceability
The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [IEE90].
Related terms: Correctness
Example:
Parent: Correctness (McCall et al., 1977)
Siblings: Consistency, Completeness (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977), (Khosravi & Gueheneuc, 2004)

Training
Those attributes of the software that provide transition from current operation or initial familiarization. (McCall et al., 1977)
Related terms:
Example:
Parent: Usability (McCall et al., 1977)
Siblings: Operability, Communicativeness (McCall et al., 1977)
Children:
Similar:
Suggested metric: consider (McCall et al., 1977)

Trustworthiness
The degree to which a system or component avoids compromising, corrupting, or delaying sensitive information.
Related terms:
Example:
Parent:
Siblings:
Children:

Similar:
Suggested metric:

Understandability
The degree to which the purpose of the system or component is clear to the user. All of the design and user documentation must be clearly written so that it is easily understandable. This is obviously subjective that the user context must be taken into account, i.e. if the software product is to be used by software engineers it is not required to be understandable to the layman. [BBK+78]
Related terms: Conciseness, Maintainability, Structuredness
Example:
Parent: Usability (ISO/IEC, 2001), Maintainability (Selby, 2007)
Siblings: Learnability, Operability, Attractiveness, Usability Compliance (ISO/IEC, 2001), Testability, Modifiability (Selby, 2007)
Children: Consistency, Self-descriptiveness, Structuredness, Conciseness, Legibility (Selby, 2007)
Similar:
Suggested metric: consider (ISO/IEC, 2003), (Khosravi & Gueheneuc, 2004)

Usability/Learnability
The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component [IEE90].
Related terms: Operability
Example:
Parent: Product Quality
Siblings: Functionality, Reliability, Efficiency, Maintainability, Portability (ISO/IEC, 2001)
Children: Training, Communicativeness, Operability (McCall et al., 1977), Understandability, Learnability, Operability, Attractiveness, Usability Compliance (ISO/IEC, 2001)
Similar:
Suggested metric: derivable from children (McCall et al., 1977), (ISO/IEC, 2003), consider (Khosravi & Gueheneuc, 2004)

Verifiability
The relative effort to verify the specified software operation and performance [EM87].
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Vulnerability
The degree to which a software system or component is open to unauthorized access, change, or disclosure of information and is susceptible to interference or disruption of system services.
Related terms:
Example:
Parent:
Siblings:
Children:
Similar:
Suggested metric:

Appendix B - Questionnaire

Appendix B provides a sample questionnaire for eliciting model parameters, as well as an example of the elicitation technique used in this work.

The next three pages represent a sample questionnaire filled from the Manager perspective.

Table 26 presents the mutual comparison matrix constructed for the breakdown of Usability reflected in the sample questionnaire.

The eigenvector approach is presented in figure 32 and table 27, while the geometric mean approach is presented in table 28.

Dear Sir/Madam,

This questionnaire is targeted at you as a MANAGER. The following tables contain pairs of quality factors and characteristics. I rely on your experience to determine the relative importance that these factors/characteristics have. In the third column, fill in a number in the range 1-5 if the first factor/characteristic is more important than the second (1 signifies equal importance, 5 signifies that the first factor/characteristic is significantly more important). Use reciprocals to signify that the first factor is less important (1/5 signifies it is significantly less important). Based on this information, I will calculate the relative weight of contribution these characteristics have in the ISO 9126 model used for the quality evaluation (Fig. 1). As you may notice, the compliance characteristics have been omitted, as they were considered not applicable in this case. The software products that are being evaluated are *xxxxxx*. Thank you!

Best,
Silviya Grigorova

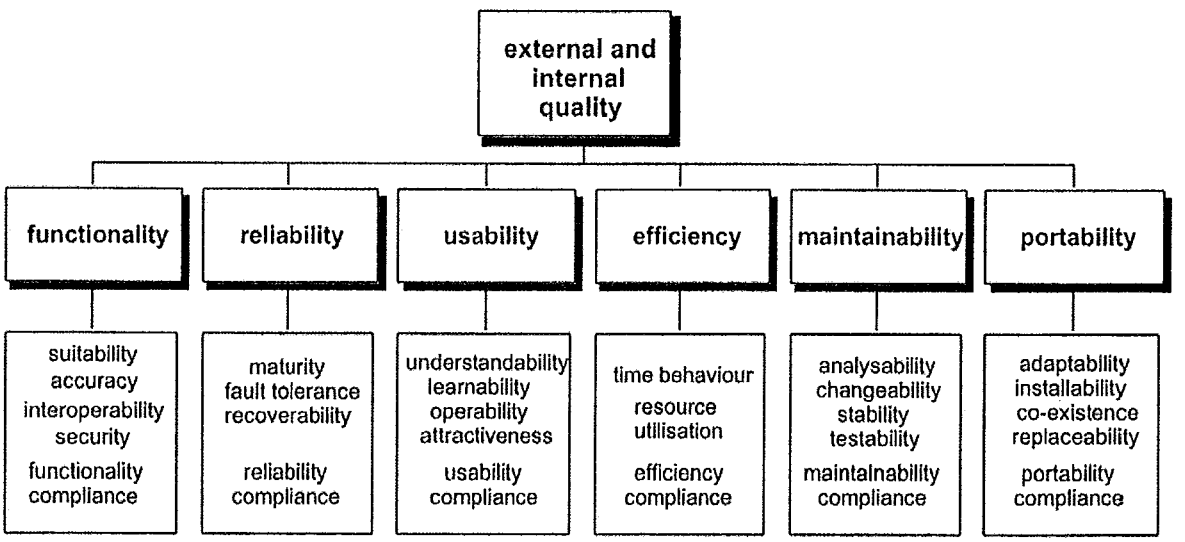


Fig. 1 Quality Model for External and Internal Quality According to ISO/IEC 9126 *

** This information has been extracted from ISO/IEC TR 9126-1 with the permission of Standards Council of Canada, in cooperation with IHS Canada, the official Canadian distributor of ISO publications, License #SCC 08/09 – 035. No further reproduction is permitted without prior written permission*

Comparison of the Relative Importance of Quality Factors Influencing **Overall Quality**

First Quality Factor	Second Quality Factor	Strength of Preference
Functionality	Reliability	½
Reliability	Usability	2
Usability	Efficiency	2
Efficiency	Maintainability	1/3
Maintainability	Portability	2
Portability	Functionality	½
Functionality	Usability	1
Reliability	Efficiency	4
Usability	Maintainability	1/3
Efficiency	Portability	1/3
Maintainability	Functionality	2
Portability	Reliability	½
Functionality	Efficiency	2
Reliability	Maintainability	1
Usability	Portability	1

Relative Importance of the Characteristics Influencing **Functionality**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Suitability	Accuracy	2
Accuracy	Interoperability	½
Interoperability	Security	4
Security	Suitability	¼
Suitability	Interoperability	1
Accuracy	Security	2

Relative Importance of the Characteristics Influencing **Reliability**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Maturity	Fault Tolerance	½
Fault Tolerance	Recoverability	2
Recoverability	Maturity	1

Relative Importance of the Characteristics Influencing **Usability**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Understandability	Learnability	½
Learnability	Operability	1
Operability	Attractiveness	1
Attractiveness	Understandability	2
Understandability	Operability	½
Learnability	Attractiveness	1

Relative Importance of the Characteristics Influencing **Efficiency**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Time Behavior	Resource Utilization	¼

Relative Importance of the Characteristics Influencing **Maintainability**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Analyzability	Changeability	1
Changeability	Stability	1
Stability	Testability	1
Testability	Analyzability	1
Analyzability	Stability	1
Changeability	Testability	1

Relative Importance of the Characteristics Influencing **Portability**

First Quality Characteristic	Second Quality Characteristic	Strength of Preference
Adaptability	Installability	½
Installability	Co-existence	4
Co-existence	Replaceability	1/3
Replaceability	Adaptability	2
Adaptability	Co-existence	2
Installability	Replaceability	1

	Understandability	Learnability	Operability	Attractiveness
Understandability	1.0000	0.5000	0.5000	0.5000
Learnability	2.0000	1.0000	1.0000	1.0000
Operability	2.0000	1.0000	1.0000	1.0000
Attractiveness	2.0000	1.0000	1.0000	1.0000

Table 26. Mutual comparison matrix constructed for the breakdown of Usability

Input matrix:

```
1.0000 0.5000 0.5000 0.5000
2.0000 1.0000 1.0000 1.0000
2.0000 1.0000 1.0000 1.0000
2.0000 1.0000 1.0000 1.0000
```

Eigenvalues Eigenvectors:

Eigenvalues:

```
(0.0000,0.0000i)
(4.0000,0.0000i)
(0.0000,0.0000i)
(0.0000,0.0000i)
```

Eigenvectors:

```
(-0.6547, 0.0000i) ( 0.2774, 0.0000i) ( 0.0000, 0.0000i) ( 0.0000, 0.0000i)
( 0.4364, 0.0000i) ( 0.5547, 0.0000i) (-0.5774, 0.0000i) (-0.5774, 0.0000i)
( 0.4364, 0.0000i) ( 0.5547, 0.0000i) ( 0.7887, 0.0000i) (-0.2113, 0.0000i)
( 0.4364, 0.0000i) ( 0.5547, 0.0000i) (-0.2113, 0.0000i) ( 0.7887, 0.0000i)
```

Figure 32. Results obtained for the matrix in table 26 by using the online calculator available at http://www.bluebit.gr/matrix-calculator/

	Eigenvector	Normalized
Understandability	0.2774	0.1429
Learnability	0.5547	0.2857
Operability	0.5547	0.2857
Attractiveness	0.5547	0.2857
sum	1.9415	1.0000

Table 27. Excel table normalizing the eigenvector values

	Geometric Mean	Normalized
Understandability	0.5946	0.1429
Learnability	1.1892	0.2857
Operability	1.1892	0.2857
Attractiveness	1.1892	0.2857
sum	4.1622	1.0000

Table 28. Excel table with geometric mean computations

REFERENCES

Agena Ltd.. (2007). AgenaRisk Software, www.agenarisk.com.

Aguarón, J., & Moreno-Jiménez, J. M. (2003). The geometric consistency index. Approximated thresholds, *European Journal of Operational Research*, Vol. 147, No.1, pp. 137-145.

Al-Kilidar, H., Cox, K., & Kitchenham, B. (2005). The Use and Usefulness of the ISO/IEC 9126 Quality Standard, *Proceedings of the 2005 International Symposium on Empirical Software Engineering (ISESE 05)*, IEEE CS Press, pp. 126-132.

Azuma, M. (1996). Software products evaluation system: quality models, metrics and processes - International Standards and Japanese Practice, *Information and Software Technology*, 38 (3), pp.145-154.

Bardis, G. (2009). Intelligent Personalization in a Scene Modeling Environment, *Intelligent Scene Modelling Information Systems*, SCI 181, pp. 89–119.

Barzilai, J. (1997). Deriving Weights from Pairwise Comparison Matrices, *Journal of the Operational Research Society*, Vol. 48, No. 12, pp. 1226-1232.

Barzilai, J. (1998). Consistency measures for pairwise comparison matrices. *Multi-Criteria Decision Analysis*, Vol. 7, No. 3, pp. 123-132.

Basili, V. R., Caldeira, G., & Rombach, H. D. (1994). The Goal Question Metric Approach. In J. Marciniak. Encyclopedia of Software Engineering. John Wiley & Sons. New York, USA.

Bayesian network. (2009). In *Wikipedia*. Retrieved March 15, 2009, from http://en.wikipedia.org/wiki/Bayesian_network

Behkamal, B., Kahani, M., & Akbari, M. K. (2009). Customizing ISO 9126 quality model for evaluation of B2B applications, *Information and Software Technology*, Vol. 51, No. 3, pp. 599-609. doi:10.1016/j.infsof.2008.08.001

Blankmeyer, E. (1987). Approaches to Consistency Adjustment, *Journal of Optimization Theory and Applications*, Vol. 54, No. 3, pp. 479-488.

Boehm, B. W. (2000). Project Termination Doesn't Equal Project Failure, *Computer*, Vol. 33, No. 9, pp. 94-96. doi:10.1109/2.868706

Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantiative Evaluation of Software Quality, *Proceedings of the 2nd International Conference on Software Engineering*. IEEE Computer Society Press, Los Alamitos, CA, pp. 592-605.

Boehm, B. W., & Valerdi, R. (2008). Achievements and Challenges in Cocomo-Based Software Resource Estimation. *IEEE Software*, Vol. 25, No. 5, pp. 74-83.

Burge, J. E. (2001). *Knowledge Elicitation Tool Classification.* Artificial Intelligence Research Group, Worcester Polytechnic Institute. Online at http://web.cs.wpi.edu/~jburge/thesis/kematrix.html

Burris, E. (2007). Software Quality Management. *Programming Large*. Retrieved June 8, 2009, from http://programminglarge.com/software_quality_management/

Castillo, E., Gutierrez, J. M., & Hadi, A. S. (1997). *Expert Systems and Probabilistic Network Models*, Springer-Verlag, New York.

Chapman, J. R. (2007). Software Development Methodology a.k.a. System Development Life Cycle. Retrieved April 23, 2009, from http://www.hyperthot.com/pm_sdm.htm

Capability Maturity Model Integration. (2009). In *Wikipedia*. Retrieved May 2, 2009, from http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration

Carnegie Mellon Software Engineering Institute. (2006). CMMI for Development, Version 1.2. Pittsburgh. Retrieved May 2, 2009, from http://www.sei.cmu.edu/reports/06tr008.pdf

Charles River Analytics. (2008). About Bayesian Belief Networks. Retrieved April 25, 2009, from http://www.cra.com/pdf/BNetBuilderBackground.pdf

Colin, S., Maskoor, A., Lanoix, A., Souquières, J., Hammad, A., Dormoy, J., Chouali, S., Hufflen, J.-M., Kouchnarenko, O., Mountassir, H., Lecomte, S., Petit, D., & Poirriez, V. (2008). A synthesis of existing approaches to specify non-functional properties, Livrable TACOS L2-1.1. Retrieved March 19, 2009, from http://tacos.loria.fr/drupal/?q=system/files/Livrable2-1.1-2Fev08.pdf

Decision Systems Laboratory, University of Pittsburgh. (2008). GeNIe Version 2.0, http://genie.sis.pitt.edu/.

DeMarco, T. (1986). *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall PTR.

Díez, F. J. (1993). Parameter adjustment in Bayes networks: the generalized noisy or-gate. *Proceedings of Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 99-105.

Dromey, R. G. (1994). A Model for Software Product Quality. Griffith University, Australia.

Dromey, R. G. (1996). Cornering the Chimera. *IEEE Software*, Vol. 13, No. 1, pp. 33-43. doi:10.1109/52.476284

Duijnhoven, J. v. (2003). Knowledge Assessment using Bayesian Networks. A case study in the domain of algebraic expectation. Utrecht University.

Eeles, P. (2005). Capturing Architectural Requirements. Retrieved July 5, 2009, from http://www.ibm.com/developerworks/rational/library/4706.html

Fenton, N. E., Neil, M., & Caballero, J. G. (2007). Using Ranked Nodes to Model Qualitative Judgments in Bayesian Networks. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 10, pp. 1420–1432. doi:10.1109/TKDE.2007.1073

Fenton, N. E., Neil, M., & Littlewood, B. (1996). Applying Bayesian belief networks to systems dependability assessment. *Proceedings of 4th Safety Critical Systems Symposium*. Springer Verlag.

Frye, C. (2008). CMMI: Good process doesn't always lead to good quality. Retrieved July 21, 2009, from http://searchsoftwarequality.techtarget.com/news/interview/0,289202,sid92_gci1316383,00.html

Galliers, J., Sutcliffe, A., & Minocha, S. (1999). An impact analysis method for safety-critical user interface design. *ACM Transactions on Computer-Human Interaction,* Vol. 6, No. 4, pp. 341-369.

Garvin, D. A. (1984). What Does "Product Quality" Really Mean?. *Sloan Management Review*, Vol. 26, No.1, pp. 25-43.

Goldenberg, A. Bayes Nets: Learning Parameters and Structure. Retrieved July 13, 2009, from http://www.cs.cmu.edu/~awm/10701/slides/Param_Struct_Learning05v1.pdf

Grady, R. B., & Caswell, D. L. (1987). *Software Metrics: Establishing a Company-wide Program.* Prentice Hall.

Gregoriades, A., & Sutcliffe, A. (2005). Scenario-Based Assessment of Nonfunctional Requirements. *IEEE Transactions on Software Engineering*, Vol. 31, No. 5, pp. 392-409.

Gurp, J.v., & Bosch, J. (2000). SAABNet: Managing Qualitative Knowledge in Software Architecture Assessment. *7th IEEE International Symposium on Engineering of Computer-Based Systems (ECBS 2000)*. IEEE Computer Society.

Han, D., & Han, I. (2004). Prioritization and selection of intellectual capital measurement indicators using analytic hierarchy process for the mobile telecommunications industry. *Expert Systems with Applications*, Vol. 26, No.4, pp. 519–527.

Huang, K., & Henrion, M. (1996). Efficient Search-Based Inference for Noisy-OR BeliefNetworks, *Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 325-331.

Hugin Expert. (2007). Hugin Researcher package, www.hugin.com.

Imperial College of Science, Technology and Medicine & Medical Research Council. (2003). WinBUGS Version 1.4, http://www.mrc-bsu.cam.ac.uk/bugs/.

ISO 9126. (2009). In *Wikipedia*. Retrieved June 11, 2009, from http://en.wikipedia.org/wiki/ISO_9126

ISO/IEC. (1999). ISO/IEC 14598-1: Information technology – Software product evaluation-Part 1: General overview. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. (2001). ISO/IEC 9126-1: Software Engineering-Product quality-Part 1: Quality model. Geneva, Switzerland: International Organization for Standardization.

ISO/IEC. (2003). ISO/IEC 9126-2: Software Engineering-Product quality-Part 2: External metrics. Geneva, Switzerland: International Organization for Standardization.

Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgment Under Uncertainty: Heuristics and Biases*, Cambridge, UK: Cambridge University Press.

Khosravi, K., & Gueheneuc, Y. (2004). A Quality Model for Design Patterns, Technical Report 1249, University of Montreal.

Koller, D., & Pfeffer, A. (1997). Object-Oriented Bayesian Networks. *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 302-313, San Francisco, Ca. Morgan Kaufmann Publishers.

Laskey, K., & Mahoney, S. (1997). Network fragments: Representing knowledge for constructing probabilistic models. *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 334–341, San Francisco, Ca. Morgan Kaufmann Publishers.

Laskey, K. B., & Mahoney, S. M. (2000). Network engineering for agile belief network models, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12, No. 4, pp. 487-498.

Lincke, R., & Löwe, W. (2006). Validation of a Standard- and Metric-Based Software Quality Model, *10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE).*

Lynch, J. (2009). New Standish Group report shows more projects failing and less successful projects. Online at http://www1.standishgroup.com/newsroom/chaos_2009.php.

Mahoney, S., & Laskey, K. (1996). Network Engineering for Complex Belief Networks. *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 389-396.

McCall, J. A., Richards, P. K. & Walters, G. F. (1977). Factors in Software Quality, Volumes I, II, and III, US. Rome Air Development Center Reports NTIS AD/A-049 014, NTIS AD/A-049 015 and NTIS AD/A-049 016, U. S. Department of Commerce.

Milicic, D. Software Quality Models. Retrieved March 6, 2009, from http://www.bth.se/tek/besq.nsf/(WebFiles)/316446EBCD98499CC125706900346 83B/$FILE/chapter_1.pdf

Miyoshi, T., & Azuma, M. (1993). An Empirical Study of Evaluating Software Development Environment Quality. *IEEE Transactions on Software Engineering*, Vol. 19, No. 5, pp. 425-435.

Monti, S., & Carenini, G. (2000). Dealing with the expert inconsistency in probability elicitation, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 12, No. 4, pp. 499-508.

Morris, A. T. (2007). Revealing the ISO/IEC 9126-1 Clique Tree for COTS Software Evaluation. AIAA Infotech@Aerospace 2007 Conference and Exhibit, AIAA Paper 2007-2960.

Morris, A. T., & Beling, P. A. (2004). Extracting Acyclic Dependency Models from Quality Standards for COTS Software Evaluation. *AIAA 1st Intelligent Systems Technical Conference*, Chicago, Illinois, Sep. 20-22, 2004.

Murphy, K. (2009). Software Packages for Graphical Models / Bayesian Networks. Online at http://people.cs.ubc.ca/~murphyk/Software/bnsoft.html

Neil, M., Fenton, N.E., & Nielsen, L. (2000). Building Large-scale Bayesian Networks, *The Knowledge Engineering Review*, Vol. 15, No. 3, pp. 257-284.

Nicholson, A. E., & Korb, K. B. (2006). Bayesian AI Tutorial. Retrieved February 20, 2009, from http://www.csse.monash.edu.au/bai/tutorial/BOMJuly06.pdf

Norsys Software Corp.. (2006). Netica, www.norsys.com.

Pearl, J. (2000). *Causality: models, reasoning, and inference*. Cambridge: Cambridge University Press.

Pressman, R. S. (2000). *Software Engineering: A Practitioner's Approach* (5th ed.). New York: McGraw-Hill.

Punter, T., Solingen, R., & Trienekens, J. (1997). Software Product Evaluation, presented at 4th IT Evaluation Conference (EVIT-97), Netherlands, Delft.

Renooij, S. (2000). Probability elicitation for belief networks: issues to consider. *The Knowledge Engineering Review*, Vol. 16, No. 3, pp. 255-269.

Saaty, T. L. (1980). *The Analytic Hierarchy Process.* New York: McGraw-Hill.

Saaty, T. L. (2003). Decision-making with the AHP: Why is the principal eigenvector necessary, *European Journal of Operational Research*, Elsevier, Vol. 145, No.1, pp. 85-91.

Selby, R. W. (Ed.). (2007). *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research*, IEEE CS Press-John Wiley &Sons.

Software quality analyst. (2009). In *Wikipedia*. Retrieved August 17, 2009, from http://en.wikipedia.org/wiki/Software_quality_analyst

Stefani, A., Stavrinoudis, D., & Xenos, M. (2004). Experimental Based Tool Calibration Used for Assessing the Quality of E-Commerce Systems, *Proceedings of the First IEEE International Conference on E-Business and Telecommunication Networks*, Portugal, Vol. 1, pp. 26-32.

Stefani, A., & Xenos, M. (2001). A model for assessing the quality of e-commerce systems, *Proceedings of the Panhellenic Conference with International Participation in Human Computer Interaction (PC-HCI 2001)*, pp. 105- 109.

Stefani, A., Xenos, M., & Stavrinoudis, D. (2003). Modelling E-Commerce Systems' Quality with Belief Networks, *International Symposium on Virtual Environments, Human-Computer Interfaces, and Measurement Systems*, Switzerland, pp. 13-18.

Van der Gaag, L. C., Renooij, S., Witteman, C. L. M., Aleman, B. M. P., & Taal, B. G. (1999). *How to elicit many probabilities*. Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence.

Van der Gaag, L. C., Renooij, S., Witteman, C. L. M., Aleman, B. M. P., & Taal, B. G. (2001). Probabilities for a Probabilistic Network: A Case-study in Oesophageal Carcinoma, University of Utrecht, UU-CS-2001-01.

Van der Gaag, L. C., Renooij, S., Witteman, C. L. M., Aleman, B. M. P., & Taal, B. G. (2002). Probabilities for a Probabilistic Network: A Case-study in Oesophageal Cancer, *Artificial Intelligence in Medicine*, Vol. 25, No. 2, pp.123-148.

Wu, W. (2007). Architectural Reasoning for Safety-Critical Software Applications. The University of York.

Xu, D., Liu, Z. T., Zhu, B. & Xing, D. H. (2005). Metric Based Software Quality Assurance System. *Current Trends in High Performance Computing and Its Applications*, Part II, pp. 551-555. doi: 10.1007/3-540-27912-1_76

Xu, W., Dong, Y., & Xiao, W. (2008). Is It Reasonable for Saaty's Consistency Test in the Pairwise Comparison Method?. *Proceedings of the 2008 ISECS international Colloquium on Computing, Communication, Control, and Management - Volume 03*, pp. 294-298. doi: 10.1109/CCCM.2008.136

Zagorecki, A. & Druzdzel, M. (2004). An Empirical Study of Probability Elicitation under Noisy-OR Assumption, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS.2004)*, pp. 880-885.