A Maturity Model of Software Product Quality

Rafa E. Al-Qutaish
Department of Management Information Systems, Al-Ain University of Science and Technology – Abu Dhabi Campus, P.O. Box 112612, Abu Dhabi, United Arab Emirates
Email: rafa@ieee.org

Alain Abran
Department of Software Engineering and IT, École de Technologie Supérieure (ÉTS), University of Québec, Montréal, Québec H3C 1K3, Canada
Email: alain.abran@etsmtl.ca

The quality of a product can be assessed either directly by looking into the product itself, or indirectly through assessing the process used to develop that product. In the software engineering field, there are currently numerous capability and maturity models for assessing a set of specific software processes, but very few product maturity models for those interested in assessing the quality of software products. This paper presents a maturity model designed to directly assess the quality of a software product, i.e. the Software Product Quality Maturity Model (SPQMM). This model is based on the six-sigma view of product quality and handles – in submodels – the three views of quality specified in ISO 9126, that is, the Software Product Internal Quality Maturity Model (SPIQMM), the Software Product External Quality Maturity Model (SPEQMM), and the Software Product Quality-in-Use Maturity Model (SPQiUMM).

Keywords: Maturity Models, Software Product Quality, Quality Assessment and Evaluation, Integrity Levels, ISO 15026, ISO 9126, Six-Sigma

ACM Classifications: D.2.0 (Software Engineering – General – Standards), D.2.8 (Software Engineering – Metrics – Product Metrics), D.2.9 (Software Engineering – Management – Software Quality Assurance)

1. INTRODUCTION

Software is critical to providing a competitive edge to many organizations and has become a key component of business systems, products, and services. The quality of software products is now considered to be an essential element in business success (Veenendaal et al., 1997). The software industry has for many years expended considerable effort in trying to improve the quality of their products, the main focus up to now having been on software process improvement as an indirect approach to achieving that quality (Veenendaal et al., 1997).

The software process is defined as “a set of interrelated activities implemented by an organization to develop software” (McGarry et al., 2002), while the software product is defined as “a set of computer programs, procedures, and associated documentation and data” (McGarry et al., 2002). Among the many capability and maturity models reported in the software engineering literature for assessing and evaluating a specific process to produce a corresponding maturity level...
A Maturity Model of Software Product Quality

are the following: the CMMi (SEI, 2002), the Software Maintenance Maturity Model, S3M (April et al, 2004; April et al, 2005), the Testing Maturity Model, TMM (Burnstein et al, 1996a; Burnstein et al, 1996b), and the Open Source Maturity Model, OSMM (Golden, 2004). However, these assessment models are about process quality, while purchasers and users of software are mostly interested in the direct assessment of the quality of the product. The design of a maturity model to assess the quality of a software product therefore represents a new research challenge in software engineering.

A Software Product Quality Maturity Model (SPQMM) could be used to determine the maturity of the quality of a specific software product. More specifically:

• to certify a quality maturity level for a new software product, which could help promote it in the marketplace;
• to benchmark existing software products to assist in deciding which of them to select based on their quality maturity level;
• to assess the quality of a software product during the development life cycle in order to investigate the relationships between the development stages and to find the weaknesses of the product with a view to improving them;
• to assess the maturity of the internal quality of a software product to be reused in other software products; and
• to compare the maturity levels of the life-cycle stages of quality (i.e. internal, external, and in-use).

This paper describes a Software Product Quality Maturity Model (SPQMM) which consists of three quality maturity sub models that can be used not only once the software has been delivered, but also throughout the life cycle:

• Software Product Internal Quality Maturity Models (SPIQMM).
• Software Product External Quality Maturity Models (SPEQMM).
• Software Product Quality-in-Use Maturity Models (SPQiU MM).

Each of these maturity sub models itself consists of a number of sub models based on their ISO 9126 quality characteristics and related product quality measures.

The rest of this paper is structured as follows: Section 2 presents related work and Section 3 explains the related standards, such as ISO 9126 on quality of software products, ISO 15026 on software integrity levels, and IEEE Std. 1012 on software verification and validation. Section 4 introduces an architectural view of the quality maturity model. Section 5 describes the structure of the software product quality maturity model. Section 6 outlines the quality maturity level determination process. Finally, Section 7 provides a discussion and concludes the paper.

2. RELATED WORK

2.1 Software Quality Models

The expression software quality is defined by the IEEE (1990) as the degree to which a system, component, or process meets specified requirements and customer (user) needs (expectations). Pressman (2004) defines it as “conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.” The ISO, by contrast, defines quality in ISO 14598-1 (ISO, 1999) as “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs” and Petrasch (1999) defines it as “the existence of characteristics of a product which can be assigned to requirements.”
A key decision in building a maturity model is the selection of an underlying quality model. In software engineering, there is a diversity of quality models available with a variety of views on quality and the detailed techniques they embody: for example, McCall’s quality model (McCall et al., 1977), Boehm’s quality model (Boehm et al., 1976; Boehm et al., 1978), Dromey’s quality model (Dromey, 1995; Dromey, 1996), and the FURPS quality model (Grady, 1992). In addition, hundreds of software measures – or what are commonly called software metrics – have been proposed to assess and evaluate various aspects of the software quality, but, unfortunately, without wide international consensus on the use and interpretation of these quality models and their software measures.

The development of a consensus in software engineering standards organizations (ISO and IEEE) is now leading to some agreement on both the content of the quality model and its corresponding measures. Examples of software quality related ISO and IEEE standards are:


In addition to the above standards, the six-sigma concepts will be used to align the mapping between quality and the maturity levels for a software product.

A strategy is then to build a maturity model based not on individual views of software quality, but on the community consensus documented in the software engineering standards.

### 2.2 Process Maturity Models

The maturity models of software development processes are useful because they indicate the various levels of process performance and, consequently, the direction in which a software process should be taken in order to improve it (McBride et al., 2004). For instance, the CMM was developed by the Software Engineering Institute (SEI) of Carnegie Mellon University in response to a request to provide the US Defense Department with a method for assessing its software contractors (SEI, 1993). Within the updated version of the CMMi, four bodies of knowledge (disciplines) are currently embedded: System Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (SEI, 2002).

The components of the CMMi-SW are process areas, specific goals, specific practices, generic goals, generic practices, typical work products, sub practices, notes, discipline amplifications, generic practice elaborations, and references (SEI, 2002). CMMi-SW can be viewed as a staged or a continuous representation. The staged representation includes five maturity levels to support and guide process improvement; in addition, it groups together process areas by maturity level, indicating which process areas to implement to achieve each maturity level (SEI, 2002). Figure 1 presents the CMMi-SW maturity level architecture.

In addition to the SEI maturity model (CMMi), there are a number of maturity models in the literature dealing with the software process, for example, the Testing Maturity Model – TMM (Burnstein et al., 1996a; Burnstein et al., 1996b). The TMM is a quantitative model, that is, it is based on measurements. It consists of five levels of testing maturity (see Figure 2), each of which has maturity goals identifying testing improvements that must be addressed to achieve the next level of maturity.

---

*Journal of Research and Practice in Information Technology, Vol. 43, No. 4, November 2011*
2.3 Product Maturity Models

In the software literature, only the following two maturity models are related to the software product:

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

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

1. Assess maturity elements (define requirements, locate resources, assess element maturity, and assign element score);
2. Assign weighting factors;
3. Calculate product maturity score.

The OSSM is designed to be a lightweight process which can evaluate an open source product’s maturity in two weeks or less (Golden, 2004). However, it does not address the quality of the software product specifically.

In addition, Nastro (1997) developed a maturity model for the software product. His model consists of three core elements (product capability, product stability, and product maintainability) and two sub elements (product repeatability and product compatibility), which may apply to specific software applications.

Based on the computed maturity level of each of the core and sub elements, Nastro (1997) proposed the following equation: \( PC \times (PS + PR + PM) / 3 \), to calculate the product maturity level of an embedded, real-time, or signal processing system, where:

- \( PC \) is the Product Capability maturity level;
A Maturity Model of Software Product Quality

- PS is the Product Stability maturity level;
- PR is the Product Repeatability maturity level;
- PM is the Product Maintainability maturity level.

This innovative software product maturity model has the following limitations, however:

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

Furthermore, Jakobsen et al (1999) have proposed a maturity model for software product quality and evaluation which is called SCOPE Maturity Model (SMM). However, this maturity model describes the maturity of software products and their evaluation, whereas our model is about the maturity of software product quality.

2.4 Six-Sigma for Industrial Products

The six-sigma approach is a quantitative approach to product quality (Head, 1994; Kingbehin, 2005; Sauro et al., 2005a; Sauro et al., 2005b). However, the term *sigma* is used in statistics to denote standard deviation, a statistical measurement of variation, that is, the exceptions to expected outcomes. The standard deviation can be considered as a comparison between expected results (or outcomes) in a group of operations and those that fail. Thus, the measurement of standard deviation shows that the rates of defects, or exceptions, are measurable. The term *six sigma* defines outcomes that are as close as possible to perfection; for example, with six sigma and using a 1.5 sigma shift, we arrive at 3.4 defects per million opportunities, or a 99.9997% level of quality (Thomsett, 2005).

In the six-sigma approach, the value 3.4 DPMO (defects per million opportunities) is followed by a footnote or an asterisk, although its fine print is typically ignored by readers. In this approach, the 3.4 DPMO is presumed to be the long-term process performance after the occurrence of a *sigma shift*. The *sigma shift* is a 1.5 sigma difference between 6 and 4.5 sigma performance, the underlying assumption being that short-term performance (of, say, 6 sigma) is actually 4.5 sigma in the long-term as entropy sets in (Wheeler, 2004). It is to be noted that in six-sigma the parts-per-million values are dependent upon the assumption that there is a 1.5 standard deviation shift.

3. RELATED INTERNATIONAL STANDARDS

This section presents overviews of the ISO and IEEE standards selected for building our software product quality maturity model. Readers familiar with these standards can skip this section and go directly to Section 4, which presents the architecture of the proposed model.

3.1 ISO 9126 on Software Product Quality – Overview

In 1991, the ISO published its first international consensus on the terminology for the quality characteristics for software product evaluation; this standard was called Software Product Evaluation – Quality Characteristics and Guidelines for Their Use (ISO, 1991). From 2001 to 2004, the ISO published an expanded version, containing both the ISO quality models and inventories of
A Maturity Model of Software Product Quality


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

The first part of ISO 9126 (ISO, 2001) describes, in some detail, a software product quality model which should be used with the other three parts. It is defined as “a framework which explains the relationship between different approaches to quality” (ISO, 2001), and considers software product quality from three distinct viewpoints (stages); that is, internal quality, external quality, and quality-in-use:

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

Figure 3 shows the ISO view of the expected relationships between internal quality, external quality, and quality-in-use attributes. The internal quality attributes influence the external quality attributes, while the external attributes influence the quality-in-use attributes. Furthermore, quality-in-use depends on external quality, while external quality depends on internal quality (ISO, 2001).

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

![Figure 3: Product quality during the lifecycle (ISO, 2001)](image-url)
A Maturity Model of Software Product Quality

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

The first part of the two-part quality model determines six characteristics, which are subdivided into twenty-seven subcharacteristics for internal and external quality, as depicted in Figure 4 (ISO, 2001). These subcharacteristics are a result of internal software attributes, and are noticeable externally when the software is used as a part of a computer system. The second part of the two-part model indicates four quality-in-use characteristics, as in Figure 5 (ISO, 2001). These characteristics and subcharacteristics are clearly defined in ISO 9126-1.

The second and third parts of the ISO 9126 series (external and internal measures) contain a basic set of measures for each external and internal quality subcharacteristic, explanations of how to apply and use software external and internal quality measures, and examples of how to apply these measures during the software product life cycle (ISO, 2003a; ISO, 2003b). The external and internal measures are classified by the characteristics and the subcharacteristics defined in ISO 9126-1.

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

3.2 ISO 15026 on Software Integrity Levels

The ISO 15026 international standard on system and software integrity levels establishes the requirements for determining the level of system and software integrity. By identifying the requirements for determining the system and software level, the software integrity requirements can be determined (ISO, 1998). This international standard defines the concepts associated with integrity levels, defines the processes for determining integrity-level requirements, and imposes requirements on each process (ISO, 1998). It is a standard that can be applied to software only.
A Maturity Model of Software Product Quality

However, the integrity level of the system and the integrity levels of the hardware components are only required in this international standard to determine the integrity levels of the software components (ISO, 1998).

The software integrity level is an assignment of either the degree of reliability of providing a mitigating function, or the limit on the frequency of failure that could result in a threat, that is, the degree of confidence that the overall system will not fail (ISO, 1998). In addition, a software integrity level refers to a range of values of a software property necessary to maintain system risks within acceptable limits (ISO, 1998).

More specifically, this ISO standard provides an example of a risk matrix (see Table 1) which could be used to calculate the risk associated with each threat. This calculation relates the frequency of occurrence of an initiating event to the severity of the consequence(s) of that initiating event. The result will be the risk class, which could be one of the following: high, intermediate, low, or trivial (ISO, 1998). Using this result, the system integrity level could be determined by mapping the risk class to the corresponding system integrity level using Table 2.

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

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

<table>
<thead>
<tr>
<th>Frequency of Occurrence</th>
<th>Indicative Frequency</th>
<th>Catastrophic</th>
<th>Severity of Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(per year)</td>
<td>High</td>
<td>Major</td>
</tr>
<tr>
<td>Frequent</td>
<td>&gt; 1</td>
<td>High</td>
<td>Intermediate</td>
</tr>
<tr>
<td>Probable</td>
<td>1 – 10^1</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Occasional</td>
<td>10^-1 – 10^-2</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Remote</td>
<td>10^-2 – 10^-3</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Improbable</td>
<td>10^-3 – 10^-4</td>
<td>High</td>
<td>Intermediate</td>
</tr>
<tr>
<td>Incredible</td>
<td>&lt; 10^-4</td>
<td>Intermediate</td>
<td>Trivial</td>
</tr>
</tbody>
</table>

Table 1: An example of a risk matrix (ISO, 1998)

<table>
<thead>
<tr>
<th>Risk Class</th>
<th>System Integrity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>A</td>
</tr>
<tr>
<td>Intermediate</td>
<td>B</td>
</tr>
<tr>
<td>Low</td>
<td>C</td>
</tr>
<tr>
<td>Trivial</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 2: Mapping the risk class to the system integrity level (ISO, 1998)
3.3 IEEE Standard 1012 on Software Verification and Validation

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

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

In this IEEE standard, the assignment of the integrity level of any particular software product will be completely based on the error consequences and their estimated occurrence (IEEE, 1998). Table 3 shows the assignment of the software integrity levels using the possible error consequences and their occurrences. For example, if the error consequence for specific software is critical and its occurrence is occasional, then the integrity level for this software will be 3.

<table>
<thead>
<tr>
<th>Error Consequence</th>
<th>Likelihood of Occurrence of an Operating State that Contributes to the Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reasonable</td>
</tr>
<tr>
<td>Catastrophic</td>
<td>4</td>
</tr>
<tr>
<td>Critical</td>
<td>4</td>
</tr>
<tr>
<td>Marginal</td>
<td>3</td>
</tr>
<tr>
<td>Negligible</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3: Assignment of software integrity levels (IEEE, 1998)

4. QUALITY MATURITY MODEL: AN ARCHITECTURAL VIEW

The architecture of the proposed maturity model is based on two sets of concepts accepted in industry in general, that is:

- Levels of quality of a product, and
- Quantitative approach to product quality.

For a quantitative approach to product quality, the six-sigma approach to assessing software product quality has been used to build the quality maturity models presented in this paper.

4.1 Maturity Levels of Product Quality

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

- Guaranteed
- Certified
- Neutral
- Dissatisfied
- Completely Dissatisfied

The sigma values are used to determine the quality maturity levels in the SPQM – see Figure 6. This maturity scale can, in turn, be applied to the two different viewpoints; that is, not only for the quality of the whole software product, but also for the life-cycle-stage quality (i.e. internal quality, external quality, and quality-in-use of the software product).
4.2 Quantitative Approach to Product Quality

To make communication with both customers and managers easier, it is more convenient to refer to a quality characteristic by a single value rather than a set of values (since each characteristic is represented by a number of related measures). To produce this single value, an organization needs to assign different weights to the individual quality views (characteristics, subcharacteristics, or measures) on a software product.

A number of techniques exist for combining multiple values. For instance, Sauro and Kindlund (2005b) have introduced a method to integrate three usability measures (effectiveness, efficiency, and satisfaction) into a single value that measures and communicates usability as a quality characteristic of a software product. This is achieved using the principal component analysis (PCA) technique (Sauro et al., 2005a) to find the weight of each of the individual usability measures using historical data.

In our models, in order to produce single values for the life-cycle stage, characteristic, or subcharacteristic quality levels, we initially assume that all measures have equal weight (contribution) in the computation of the subcharacteristic quality level (and characteristic quality level, in the case of quality-in-use), and each subcharacteristic has equal weight in the calculation of the characteristic quality level. This single value can be easily converted to a sigma value using the NORMSINV function, which is a Microsoft Excel function that delivers the inverse of the cumulative standardized normal distribution (Sauro et al., 2005b).

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

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

In Table 4, the sigma values have been calculated and shifted based on the quality levels and software integrity levels respectively. Again, the NORMSINV function is used to produce the sigma values for all the quality levels, and the integrity levels have been used to determine the value of the sigma shift. Table 4 is made up of three parts:

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

Using the upper part and the lower part, we can obtain the assigned sigma range value from the right-most part.
In Table 4, we identified six values for the sigma shift (i.e. 0, 1.5, 2.0, 2.5, 3.0, and 3.5) to determine the various gaps between the quality level values; a higher sigma shift will produce a greater gap between the quality levels. For example, we used a zero sigma shift with an integrity of level 5, because software products with this integrity level are very sensitive to quality; therefore, the use of this sigma shift (zero) produced ranges of quality levels with small gaps, that is, from 97.724% to 99.9997% (see Table 4 – zero sigma shift column). While the 1.5 sigma shift is used with integrity level 4, because software products with this integrity level are less sensitive to quality than software products with integrity level 5, the ranges of quality levels produced have larger gaps, that is, from 69.146% to 99.865% (see Table 4 – 1.5 sigma shift column). Table 4 will be used as the reference to assess the product quality maturity level (see Subsection 6.1).

Table 4 can be used directly to obtain the assigned sigma range for a quality level when the integrity level is known for the software product being assessed. Therefore, the same quality level for a specific software product could lead to different maturity levels of that software product based on its integrity level. (See the next subsection for an example of this.)

4.3 Example of a Quality Level with its Various Maturity Levels

Table 5 presents an example of a software product with a quality level of 99% (using the NORMSINV Excel function, the original sigma (without a sigma shift) value for 99% is 2.32σ).

<table>
<thead>
<tr>
<th>Quality Level</th>
<th>Original Sigma Value (OSV)</th>
<th>Intensity Level (using Table 8)</th>
<th>Sigma Shift Value (SSV)</th>
<th>Shifted Sigma Value (OSV+SSV)</th>
<th>Corresponding Maturity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>99% 2.32σ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
<td>(2.32+0.0) = 2.32σ</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.5</td>
<td>(2.32+1.5) = 3.82σ</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>(2.32+2.0) = 4.32σ</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.5</td>
<td>(2.32+2.5) = 4.82σ</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.0</td>
<td>(2.32+3.0) = 5.32σ</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3.5</td>
<td>(2.32+3.5) = 5.82σ</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: An example of a software product with a quality level of 99%
A Maturity Model of Software Product Quality

There are six possibilities for evaluating the quality maturity level of a software product, based on the integrity level of that particular product:

1. If the integrity level is 5,
   • using Table 4, the corresponding sigma shift = 0.0;
   • the shifted sigma value = $2.32\sigma + 0.0\sigma = 2.32\sigma$; and
   • using Figure 6, the corresponding maturity level = 2.

2. If the integrity level is 4,
   • using Table 4, the corresponding sigma shift = 1.5;
   • the shifted sigma value = $2.32\sigma + 1.5\sigma = 3.82\sigma$; and
   • using Figure 6, the corresponding maturity level = 3.

3. If the integrity level is 3,
   • using Table 4, the corresponding sigma shift = 2.0;
   • the shifted sigma value = $2.32\sigma + 2.0\sigma = 4.32\sigma$; and
   • using Figure 6, the corresponding maturity level = 4.

4. If the integrity level is 2,
   • using Table 4, the corresponding sigma shift = 2.5;
   • the shifted sigma value = $2.32\sigma + 2.5\sigma = 4.82\sigma$; and
   • using Figure 6, the corresponding maturity level = 4.

5. If the integrity level is 1,
   • using Table 4, the corresponding sigma shift = 3.0;
   • the shifted sigma value = $2.32\sigma + 3.0\sigma = 5.32\sigma$; and
   • using Figure 6, the corresponding maturity level = 5.

6. If the integrity level is 0,
   • using Table 4, the corresponding sigma shift = 3.5;
   • the shifted sigma value = $2.32\sigma + 3.5\sigma = 5.82\sigma$; and
   • using Figure 6, the corresponding maturity level = 5.

5. SOFTWARE PRODUCT QUALITY MATURITY MODEL

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

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

Figure 7 illustrates the relationships between the above two types of quality maturity models.
5.1 The Whole Software Product Quality Maturity Model
The whole software product view of our model (Software Product Quality Maturity Level) is illustrated in Figure 8, and shows the components that should be computed to arrive at the quality maturity level of the whole software product. The resulting maturity level is derived from the quality levels of the software product stages (internal, external, and in-use).

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

- Software Product Internal Quality Maturity Model (SPIQ\textsuperscript{MM}),
- Software Product External Quality Maturity Model (SPEQ\textsuperscript{MM}), and
- Software Product Quality-in-Use Maturity Model (SPQiU\textsuperscript{MM}).

Each of the above life-cycle-stage-view maturity models is based on the ISO 9126 software product quality characteristics selected. These characteristics and their subcharacteristics should be selected based on the type of software (e.g. embedded, real-time, application, system, etc.) to be evaluated. Figure 9 shows the components of the software product life-cycle-stage quality maturity models.

5.3 Determination of the Software Integrity Level
The software integrity level is used in our models to classify software products. Therefore, software products with high integrity levels are very sensitive to quality. In order to identify the integrity level of a software product, we first need to know the consequences of any failure of that product. Table 6 illustrates the definitions of the consequences that have been expanded into six consequences, instead of the four in the IEEE standard for software verification and validation (IEEE, 1998). In addition, the occurrence of those consequences is very important for determining

---

Figure 8: Components of the quality maturity level for the whole software product
A Maturity Model of Software Product Quality

Figure 9: Components of the quality maturity levels from the life-cycle-stage viewpoint

Table 6: Definitions of expanded consequences

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

Table 7: Indicative frequency for each occurrence (IEEE, 1998)

<table>
<thead>
<tr>
<th>Occurrence</th>
<th>Indicative Frequency (IFreq) (per year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>IFreq ≥ 1</td>
</tr>
<tr>
<td>Probable</td>
<td>0.1 &lt; IFreq ≤ 1</td>
</tr>
<tr>
<td>Occasional</td>
<td>0.01 &lt; IFreq ≤ 0.1</td>
</tr>
<tr>
<td>Remote</td>
<td>0.001 &lt; IFreq ≤ 0.01</td>
</tr>
<tr>
<td>Improbable</td>
<td>0.00001 &lt; IFreq ≤ 0.0001</td>
</tr>
<tr>
<td>Incredible</td>
<td>IFreq &lt; 0.000001</td>
</tr>
</tbody>
</table>

Table 8: Determination of the software integrity level using the consequences and their occurrence

<table>
<thead>
<tr>
<th>Consequence</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Frequent 5</td>
</tr>
<tr>
<td>Critical</td>
<td>Frequent 5</td>
</tr>
<tr>
<td>Severe</td>
<td>Frequent 4</td>
</tr>
<tr>
<td>Marginal</td>
<td>Frequent 3</td>
</tr>
<tr>
<td>Minor</td>
<td>Frequent 2</td>
</tr>
<tr>
<td>None</td>
<td>Frequent 0</td>
</tr>
</tbody>
</table>
the integrity level. The frequency of their occurrence can be estimated using the indicative
frequency (IFreq) of previous and similar software products (see Table 7).

Based on the consequences and their occurrence, we have classified the software integrity levels
into an ordered scale ranging from 0 to 5 – see Table 8. For example, if the consequences and their
occurrence are “severe” and “occasional” respectively, then the integrity level of that software is 3.

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

5.5 Identifying the Required Base Measures for each Selected Characteristic
Abran et al (2005) have classified the ISO 9126 measures into base and derived measures, based on
the ISO 15939 international standard on the software measurement process, and have provided a list
of proposed base measures in ISO 9126, parts 2, 3, and 4. This classification helps in determining
which measures should be collected before starting the measurement process, taking into account
that most ISO 9126 measures are derived and could not be computed without first identifying and
collecting the base measures. In addition, a cross-reference table of base measure usage provided
by Abran et al (2005) identifies, for each subcharacteristic (or characteristic, in the case of quality-
in-use), which base measures should be collected to compute the related derived measures for that
subcharacteristic or characteristic – see Table 9. In particular, these lists can help in:

• identifying, selecting, and collecting a base measure (once), and then reusing this base measure
to evaluate a number of derived measures; and

<table>
<thead>
<tr>
<th>Base Measure Name</th>
<th>Unit</th>
<th>Functionality</th>
<th>Reliability</th>
<th>Usability</th>
<th>Efficiency</th>
<th>Maintainability</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Functions</td>
<td>Function</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Operation Time</td>
<td>Minute</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Number of Inaccurate Computations Encountered by Users</td>
<td>Case</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Data Formats</td>
<td>Format</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Illegal Operations</td>
<td>Operation</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Items Requiring Compliance</td>
<td>Item</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Number of Interfaces Requiring Compliance</td>
<td>Interface</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Faults</td>
<td>Fault</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: F1, …, R1, …, U1, …, E1, …, M1, …, and P1,… denote the subcharacteristics of the Functionality,
Reliability, Usability, Efficiency, Maintainability, and Portability quality characteristics, respectively, as in Figure 4.

Table 9: Examples of the use of base measures (Abran et al, 2005)
A Maturity Model of Software Product Quality

- learning which base measures are required to evaluate specific software quality attributes (characteristics and subcharacteristics).

5.6 Computing the Quality Levels of the Software Product Quality Characteristics Selected

At this point, the characteristics, subcharacteristics, and measures have been selected and the base measures identified and collected. In order to calculate the quality maturity level of the software product, the quality levels of the selected characteristics must be computed. Figure 10 shows the steps to follow to compute the quality level of any of the selected internal or external characteristics and subcharacteristics, and, in Figure 11, of the quality-in-use characteristics.

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

\[
SQL = \left( \frac{1}{n} \sum_{i=1}^{n} q_{mi} \right)
\]

\[
CQL = \left( \frac{1}{m} \sum_{i=1}^{m} SQL_i \right)
\]

\[
QL = \left( \frac{1}{L} \sum_{i=1}^{L} CQL_i \right)
\]

where:
- **SQL** is the Subcharacteristic Quality Level,
- **CQL** is the Characteristic Quality Level,
- **QL** is the Internal/External Quality Level,

![Figure 10: Computation of the quality level for each of the ISO 9126 internal and external characteristics](image-url)
A Maturity Model of Software Product Quality

• \( qm \) is the selected Quality Measure,
• \( n \) is the number of selected measures of that subcharacteristic,
• \( m \) is the number of selected subcharacteristics of that characteristic, and
• \( L \) is the number of selected internal/external/in-use characteristics of that software product.

In the case of the in-use software product, there are no subcharacteristics for each characteristic, but rather a number of related measures. Equation 4 can be used to calculate the quality levels of the characteristics, whereas the quality level of the in-use software product (in-use stage) can be calculated using Equation 3:

\[
CQL = \left( \frac{\sum_{i=1}^{m} qm_i}{m} \right)
\]  

(4)

where \( m \) is the number of selected measures of that characteristic.

For the whole software product, Equation 4 can be used to arrive at a single value of the quality level of the whole software product:

\[
WQL = (IQL + EQL + iUQL)^{\frac{1}{3}}
\]

(5)
A Maturity Model of Software Product Quality

where:

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

For Equations 1 to 5, we have made the assumption that all measures, subcharacteristic quality levels, characteristic quality levels, and stage quality levels make the same contribution (have the same weight) in the calculation of the corresponding subcharacteristic quality level, characteristic quality level, stage (internal, external, or in-use) quality level, and whole software product quality level respectively. Of course, if an organization wishes to assign different weights, appropriate techniques must be used to integrate them – see Subsection 4.2.

5.7 Identifying the Sigma Value and the Maturity Level

In the previous subsections, instructions have been provided for calculating the quality level for any of the two viewpoints (the life-cycle stage and the whole software product), and how to determine the software integrity level.

It is easy to arrive at the corresponding sigma range for the quality level of any of the two viewpoints using Table 4. For example, if the following information about a specific software product is available:

• the Software Integrity Level = 2 (i.e. 'low'), and
• the External Quality Level = 80%,

then, using Table 4, the sigma range is within $4 > \sigma \geq 3$. From Figure 6 of the quality maturity levels, the Software Product External Quality Maturity level for this specific sigma range is 3 (neutral).

6. DISCUSSION AND CONCLUSION

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

Several capability and maturity models have been designed for the software process, but, up to now, there has been no corresponding product maturity model for assessing the quality of software products. The design of a software product maturity model to assess the quality of a software product therefore represented a challenge. In this paper, we have presented a product quality assessment model based on ISO 9126 and ISO 15026. Specifically, we have discussed the structure of the quality maturity model from the following two distinct points of view:

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

In the literature, there are many quality models and hundreds of measures dealing with the software product. Selecting which to use for evaluating software product quality is a challenge. To address this diversity of alternatives, the ISO has come up with a consensus on a quality model and an inventory of measures to evaluate the quality of a software product (ISO 9126). However, using individual ISO 9126 measures to evaluate the quality of a software product is also a challenge, since we will obtain a set of numbers that reflects the quality level of each measure for each quality
characteristic. Moreover, it is difficult to interpret these numbers on the one hand, and to integrate them into a decision-making model on the other.

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

The software product quality maturity models discussed in the paper are limited to the ISO 9126 quality model and its set of measures. Another limitation is that the results yielded by the quality maturity models discussed are based entirely on the assumption of the equal weight of all measures, all characteristics, and all subcharacteristics. To avoid making this assumption, an organization can apply relevant statistical techniques. Alternatively, it can assign its own weights using, for instance, the analytical hierarchy process (AHP) technique (Koscianski et al., 1999) and then apply appropriate techniques to combine them into aggregated values at higher levels, such as is done, for example, in the QEST multi-dimensional models for quality and performance (Buglione et al., 1999; Buglione et al., 2002).

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

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

REFERENCES
A Maturity Model of Software Product Quality


WHEELER, D. J. (2004): The six sigma practitioner’s guide to data analysis. Knoxville, TN, USA, SPC Press.
A Maturity Model of Software Product Quality

BIOGRAPHICAL NOTES

Rafa E. Al-Qutaish holds a PhD in software engineering (2007) from the School of Higher Technology, University of Québec (Canada), MSc degree in software engineering (1998) from the University of Putra (Malaysia), and BSc degree in computer science (1993) from Yarmouk University (Jordan).

He is currently an assistant professor at Al-Ain University of Science and Technology (Abu Dhabi, UAE). His research interests include software measurements, software engineering standards, software quality engineering, maturity models, software comprehension and maintenance, reverse engineering, software testing, communication software engineering, and applied artificial intelligence. He has published over 32 peer-reviewed research papers.

Dr Al-Qutaish is a Canadian representative and Committee Member of ISO/IEC JTC1/SC7 Work Group 6 on Software Evaluation and Measurement. In addition, he is a senior member of the IEEE Computer Society, member of the United Kingdom Software Metrics Association (UKSMA), senior member, International Association of Computer Science and Information Technology (IACSIT), and acting member of ACM.

Alain Abran holds a PhD in electrical and computer engineering (1994) from École Polytechnique de Montréal (Canada) and Master degrees in management sciences (1974) and electrical engineering (1975) from the University of Ottawa.

He is a professor and the Director of the Software Engineering Research Laboratory at the École de Technologie Supérieure (ETS) – Université du Québec (Montréal, Canada). He has over 15 years of experience in teaching in a university environment as well as more than 20 years of industry experience in information systems development and software engineering. His research interests include software productivity and estimation models, software engineering foundations, software quality, software functional size measurement, software risk management and software maintenance management. He has published over 300 peer-reviewed publications and he is the author of the book ‘Software Metrics and Software Metrology’ and a co-author of the book ‘Software Maintenance Management’ (Wiley Interscience Ed. & IEEE-CS Press).