IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 5, Oct-Nov, 2013
ISSN: 2320 - 8791
www.ijreat.org

A Study on how to Measure Software Quality

Khallikkunaisa[1]

[1]**Department of Computer Science and Engineering, VTU/HKBK/Bangalore, Karnataka, 560045/India**

Abstract

As software products grow bigger in size and complexity, the software quality assurance becomes more and more important. In the end, it is the quality of the software which determines how well the product succeeds in the market. Although a good quality process most often leads to a better quality, it does not mean that one can forget the quality of the software product itself. Instead of the quality processes or the quality of the processes, the focus in this paper is on the quality of the software product itself and how it can be measured using-quality management standards. It defines how the company manages quality in general but it does not help the software teams validate the quality of their software products.. This paper consists of the theoretical study of the software quality in general and provides suggested quality metrics from the literature. This paper introduces different concepts of software quality metrics based on both the ISO/IEC 9126 and ISO/IEC 25000 software quality standards and explains the meaning of software quality model.

Keywords:*SoftwareQuality, ,Qualitymetrics,Quality model,Quality standards.*

1. Introduction

Software quality is investigated by looking into its meaning in general and defining it with the software engineering literature. Means for software quality measurement are explained by introducing general software metrics and ways to calculate them. Chapter ends by introducing ISO's (the International Organization for Standardization) and IEC's (the International Electro technical Commission) ISO/IEC 9126 and ISO/IEC 25000 families of standards, which are designed for software product quality.

1.1Software Quality in General

Simple definition for software quality is a hard task to achieve. The quality of the product can be seen as bad or good depending on who is judging it. In general software quality is an abstract term which consists of people's expectations and experiences of the system. People have their own opinions on how a product should work, how fast it responds to their commands and so on.Quality is a multidimensional concept that consists of entity of interest, the viewpoint of that entity and the quality attributes of the entity. Quality is an abstract concept that can have different layers. This means that people can have very different definitions for the quality depending on their backgrounds.

To end user, good software quality can mean that the product provides efficient and necessary functionalities to complete the task it was designed for. This means, for example, in an online book store easy and safe credit card transactions so that the wanted book is easy to order and the payment is not charged twice. To the software developer good quality can mean good maintainability or testability, how easy it is to maintain and fix bugs or how easy it is to write unit tests. To software architects good quality can mean the reusability of the used software components as well as the quality of the documentation of the system.Juran and Gryna[1] defined quality as "fitness for use". Ioarmis and Pangiotis (2007, 7) raise two meanings from it. First, the quality consists of the features which are needed to satisfy the customer requirements and thus produce product satisfaction. Secondly, the good quality brings freedom from the deficiencies.

Crosby[2] introduces definition for quality as "conformance to requirements". Kan(2002, 2) states that it means software requirements must be clearly written to avoid any misunderstandings. This is monitored during production phase using regular measurements. Any deviation from those requirements is considered to be a defect.

To summarize, one can say: The quality of the software product means its ability to fulfill or exceed all the expectations of the user. This should be achieved by using reasonable amount of resources and containing acceptable level of system complexity.

1.2 Measuring Software Quality

Software products are getting bigger and bigger in size and in numbers of components. Different components exchange information using different interfaces to other components. This means that the overall complexity of the systems grows. It has been estimated that 50-80% of costs of the software This is the reason why it is important for a software company to understand the quality of their products in order to increase efficiency of the software development.. The purpose is to give an insight to the software quality metrics as well as set the ground level for further analysis.

This is the reason why it is important for a software company to understand the quality of their products in order to increase efficiency of the software development.

1.2.1 Static and Dynamic Quality Analysis

In static quality analysis the actual product is not executed, instead the quality of its parts, the source code and documentation are analysed. Analysis tools can be used to predict the overall complexity of the product by calculating number of lines, number ofcomponents or interfaces between components. With object-oriented programming languages more complicated complexity metrics can also be used. Using these static metrics can help people understand how maintainable or reusable the software product is.

As opposite to static quality analysis, the dynamic quality analysis is run by executing the product in specific environment. This is normally done during testing phase for example at the end of the building process. Running dynamic analysis gives a better understanding for example how effective and reliable the product is. Effectiveness can be measured by monitoring the product's use of resources and reliability can be measured by calculating the test coverage.

1.2.2 Lines of Code

The "lines of code value" is the simplest measurement for the complexity in the software system. Its abbreviation is LOC or KLOC (1000 lines of code) for large programs. It has not been fully defined how LOC should be calculated. According to Lee, Gunn, Pham and Ricaldi (1994) LOC means all the non-executables lines of code, including comments and headers. It is important to use one single defination throughout the analyses of the software product.

Kan states (Kan, 2002, 312) that LOC isnormally calculated from the number of executed statements in source code. Studies show that defect density (defects per KLOC) is related to LOC count. Figure 2 illustrates the curvilinear relationships between defect density and product module size in LOC.

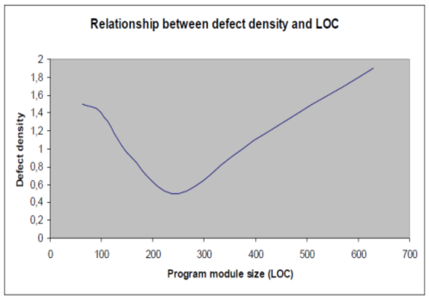


Figure 2. Relationship between found defects and program module size.

According to Kan (Kan, 2002, 313) there might be optimum balance for software product size and defect rate. Such a balance would lead to lowest amount of defects per product. Finding such a balance would require more empirical studies of the subject.

1.2.3 Halstead's Complexity Metrics

Referring to Lee et al. (1994) Halstead[3] separated software science from the computer science by dividing software programming to operators and operands. Halstead defined four basic measurements from the source code.

Primitive measures of software science:

n1 = Number of distinct operators in a program

n2 = Number of distinct operands in a program

N1 = Number of operator occurrences

N2 = Number of operand occurrences

He then used these to derive program length, program volume, program size, program difficulty, mental effort and estimated number of errors.

Program Length (N) Program Volume (V) Program Size (S) Program Difficulty (D) Mental effort (E) Estimated number of errors (B)	=N1 +N2 =(N1+N2) In (n1+n2) = (n1) In (n1) + (n2) In (n2) =[(n1)/2] (N2/n2) = [(nl) (N2) (N1+N2)In(nl+n2)] / 2(n2) (5) = [E* x (2/3)] / 3000

Halstead's calculations have had huge affect on software metrics. Biggest criticism towards Halstead's complexity metrics is that the calculations are dependent on N1 and N2. This means that the calculation to be accurate, the program has to be nearly fmished. Also the estimated number of errors (equation 6) states simply that number of errors in software program depends on the size of th

1.2.4 Cyclomatic Complexity

From the study of Kan (2002, 315) we learn that McCabe[4] introduced in 1976 the measurement of the cyclomatic complexity. It was created to illustrate testability and maintainability of the program.

McCabe cyclomatic complexity

$$M = V(G) = e - n + 2p$$

where

V(G) is Cyclomatic number of G,

e is the number of edges,

n is the number of nodes and

p is the number of unconnected parts of the graph.

McCabe's cyclomatic complexity number can be used to calculate the number of different individual paths through the program's logic. (Lee et al., 1994) This will give us a rough estimate of the needed test cases to cover 100% of the source code during unit testing. McCabe equation can be used to validate degree of test coverage results by comparing it to the number of actual execution rounds

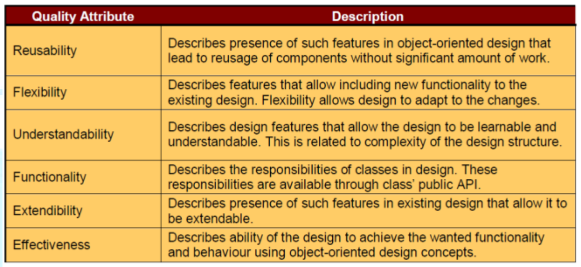
1.2.5 Object-Oriented Metrics

In object-oriented (00) software the classes and functions are the basic building blocks of the software. It is natural that the 00 metrics are closely related to classes, methods, and the size (lines of code). When measuring complexity of the 00 components, the metrics should take 00 characteristics such as inheritance, instance variables, and coupling into account.

1.2.6 Quality Model for Object-Oriented Design

Quality Model for Object-Oriented Design (QMOOD).The QMOOD is a comprehensive quality model that presents clearly defined and empirically validated model to estimate object-oriented design attributes. (7)

Table 1: Quality Model for Object Oriented Design.



Quality Attribute	Description
Reusability	Describes presence of such features in object-oriented design that lead to reusage of components without significant amount of work.
Flexibility	Describes features that allow including new functionality to the existing design. Flexibility allows design to adapt to the changes.
Understandability	Describes design features that allow the design to be learnable and understandable. This is related to complexity of the design structure.
Functionality	Describes the responsibilities of classes in design. These responsibilities are available through class' public API.
Extendibility	Describes presence of such features in existing design that allow it to be extendable.
Effectiveness	Describes ability of the design to achieve the wanted functionality and behaviour using object-oriented design concepts.

Table 1 introduce 6 quality attributes for the QMOOD quality model: reusability, flexibility, understandability, functionality, extendibility and effectiveness. These attributes are loose related to ISO/IEC 9126 standard.

2.ISO/IEC 9126 Series of standards - Software Product Quality

ISO/IEC 9126 series of standard family is the series of standards that introduces concepts of software quality model. ISO/IEC FDIS 9126:2000 version of the standard replaces the older ISO/IEC 9126:1991 standard. Software qualityevaluation was removed from ISO/IEC 9126:1991 to its own standard, the ISO/IEC 14598 standard. Documents included in ISO/IEC 9126 are software quality model (ISO/IEC 9126-1); external metrics (ISO/IEC 9126-2); internal metrics (ISO/IEC 9126-3) and quality in use (ISO/IEC 9126-4). (ISO/IEC 9126:2000, v)

In ISO/IEC 9126:2000 the software quality model is divided into two parts. The first part contains external and internal metrics of the software. External and internal metrics are categorized using six quality characteristics and those are then further divided into sub characteristics. The second part contains software quality in use, which is divided into four characteristics. (ISO/IEC 9126:2000, 1

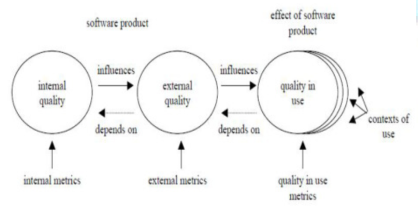
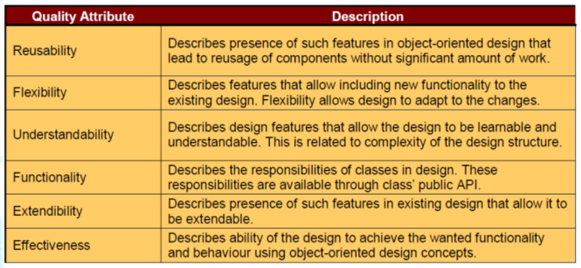


Figure 3. Relationship between software product quality metrics. (ISO/IEC 9126-2:2001, 3)

2.1 Internal and External Quality Metrics

Table 2 contains the characteristics and corresponding sub characteristics for internal and external software product quality metrics. These are quality perspectives which may be used in company's quality assurance. This chapter provides brief description of each characteristic and its sub characteristics. To get a deeper understanding of different perspectives, one should study the ISO/IEC 9126-2 (External metrics) and the ISO/IEC 9126-3 (Internal metrics) standards. These documents introduce more detailed ideas for example how one should measure each of these quality attributes.

Table 2: ISO/IEC 9126-2 and ISO/IEC 9126-3 software product quality metrics for internal and external metric



Quality Attribute	Description
Reusability	Describes presence of such features in object-oriented design that lead to reusage of components without significant amount of work.
Flexibility	Describes features that allow including new functionality to the existing design. Flexibility allows design to adapt to the changes.
Understandability	Describes design features that allow the design to be learnable and understandable. This is related to complexity of the design structure.
Functionality	Describes the responsibilities of classes in design. These responsibilities are available through class' public API.
Extendibility	Describes presence of such features in existing design that allow it to be extendable.
Effectiveness	Describes ability of the design to achieve the wanted functionality and behaviour using object-oriented design concepts.

===

2.2 Quality in Use Metrics

Quality in use metrics are divided into 4 different characteristics which all measure how well the final product fits to its purpose to allow user to achieve his or hers goals in specified context of use. Table 3 lists characteristics and their meanings. (ISO/IEC 9126-4:2001, 5)

Table 3: ISO/IEC 9126-4 software product quality metrics for 'Quality in use

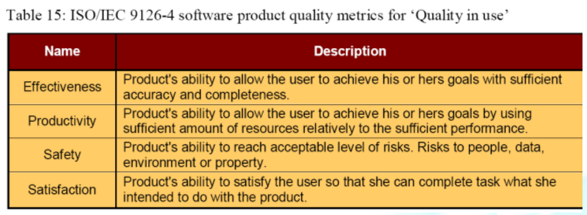


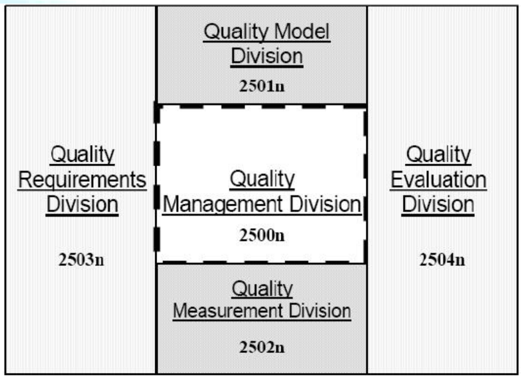
Table 15: ISO/IEC 9126-4 software product quality metrics for 'Quality in use'

Name	Description
Effectiveness	Product's ability to allow the user to achieve his or hers goals with sufficient accuracy and completeness.
Productivity	Product's ability to allow the user to achieve his or hers goals by using sufficient amount of resources relatively to the sufficient performance.
Safety	Product's ability to reach acceptable level of risks. Risks to people, data, environment or property.
Satisfaction	Product's ability to satisfy the user so that she can complete task what she intended to do with the product.

3. ISO/IEC 25000 Series of standards - Software Quality Requirements and Evaluation

ISO/IEC 25000 series of standards replace the ISO/IEC 9126 and ISO/IEC 14598 standard families. It binds them into one standard family providing best practices and lessons learned from both ISO/IEC 9126 and ISO/IEC 14598 standards. ISO/IEC 25000 is often regarded as SQuaRE, Software Quality Requirements and Evaluation.

General idea in SQuaRE is to take into use a logically ordered and unified standard that is divided into two main processes: software quality requirements specification and software quality evaluation. Both of these processes are supported by software quality measurement process. SQuaRE is created to aid those who develop software, those who acquire software products and those who evaluate the software quality. This is established by defining criteria for the requirements, measurements and the evaluation of the software quality. SQuaRE offers two-part quality model which introduces recommended software quality metrics to be used by the developers, acquirers and evaluators. As distinction to ISO 9000 standards, SQuaRE is dedicated to the software product quality instead of the Quality Management processes


Quality Model Division
2501n
Quality Requirements Division
2503n
Quality Management Division
2500n
Quality Evaluation Division
2504n
Quality Measurement Division
2502n
isions of SQuaRE series of standards. (ISO/IEC 2500

4. Conclusions

Over the years the software products have grown bigger in size and complexity, and that has raised the need for a good software quality assurance. To be able to provide a sufficient level of software quality, different software quality assurance processes must be in place. Unfortunately, these quality processes do not often take into account the quality of the actual software product itself.

References

Books, articles and web pages

Ioarmis G. Stamelos, and PanagiotisSfetsos, 2007. Agile Software Development Quality Assurance, Information Science Reference.

Kan, H. Stephen, 2002. Metrics and Models in Software Quality Engineering, Second Edition.Addisson Wesley.

Tian, Jeff, 2005, Software Quality Engineering — Testing, Quality Assurance, and Quantifiable Improvement, John Wiley & Sons, Inc.

Lee, T. Alice, Gunn Todd, Pham Tuan and Ricaldi Ron, 1994. Technical Memorandum