SOFTWARE QUALITY MANAGEMENT – A STUDY

Rashmi Dewan, Shivangi Kukreja, Nikita Pahuja
Student, Computer Science & Engineering, Maharshi Dayanand University
Gurgaon, Haryana, India

***Abstract*- The intention of this paper is to provide an overview on the subject of software project management. The overview includes concept of software quality. This paper also covers ISO 9126, quality standards, software quality, measurement technique and principles of quality management. Through this paper we are creating awareness among the people about this rising field of SPM. This paper also offers a comprehensive number of references for each concept of SOFTWARE QUALITY.**

***Index Terms*- Customer, Functionality, Quality, Interdependency, Software, Software Management.**

I. INTRODUCTION

In the reference to software engineering, **software quality** introduces two related but distinct concepts that exist wherever quality is defined in a business context:

- Depending upon the functional requirements or specifications, software functional quality reflects how well it adheres to a given design. This attribute can relate to as the fitness for of a piece of software or how it compares to opponents in the marketplace as a beneficial product.
- Software structural quality is the degree to which the software was produced correctly. It also implies how it meets the non-functional requirements which support the delivery of the functional requirements-for example robustness and maintainability.

The functional quality of software is measured and enforced through software testing. On the other hand the structural quality is assessed in accordance with the analysis of the software inner structure and its source code at the unit level, the system level and the technology level.
ISO 9126-3 and the subsequent ISO 25000:2005 quality models, which is also known as Square, defines the classification, structure and the terminology of attributes and the metrics that are applicable to software quality management.
The Consortium for IT Software Quality (CISQ) has defined the following five major desirable structural characteristics needed by software piece to stand up to business value:

1) Security
2) Maintainability
3) Efficiency
4) Reliability
5) Size(adequate)

The extent to what a software or a system rates along each of the above five dimensions is expressed by the **Software Quality Measurement**.

II. MOTIVATION

"A science is as mature as its measurement tools," (Louis Pasteur in Ebert Dumke, p. 91). Software Quality measurement is motivated by following 2 reasons:

- Risk Management: Software failures and errors have caused more inconvenience and fatalities respectively. These causes have ranged from poorly designed user interfaces to direct programming errors. This requires for the development of embedded software in medical and other devices that regulate critical infrastructures. Engineers writing embedded software assume Java programs stalling for one third of a second to perform garbage collection, update the user interface and they visualize airplanes falling out of the sky.
  The Aircraft Certification Service within the Federal Aviation Administration (FAA) is located in the United States. It provides software programs, guidance, policy, training and focus on software and complex electronic hardware that has an effect on the airborne product which may be an aircraft, an engine or a propeller.

- Cost Management: An application that has good structural software quality will cost less. It is easier to understand and maintain such an application as in any other fields. Industry statistics reveal that a poor application structural quality results in expense and schedule overruns and creates trash in the form of rework. Such core applications are Enterprise Resource Planning, Customer Relationship Management. However, an application that has poor structural quality is correlated with high-impact business disruptions due to corrupted data, application outages and performance problems.

Moreover, the distinction between measuring and improving software quality in business software, with emphasis on maintenance and cost and in embedded software, with emphasis on risk management has become irrelevant.

Embedded systems include a user interface and the designs are much concerned with issues that affect user productivity and interfaces.

Nowadays, both the types of software use multi-layered technology stacks and complex architecture, therefore software quality analysis and measurement have to be managed in a comprehensive and consistent manner.

III. ISO 9126

ISO/IEC 9126 for *Software engineering i.e. Product quality* is an international standard for evaluation of software. The standard has been replaced by ISO/IEC 25010:2011. The major objective of the ISO/IEC 9126 standard is to address some of the well-known human biases that can affect the delivery and perception of a software development project in an adverse manner.

The biases include changing priorities after start of the project or not having any clear definition of "success." Then by agreeing on the project priorities and then converting abstract priorities (compliance) to measurable values (output data can be validated against schema X with zero intervention), ISO/IEC 9126 tries to advance a common understanding of the project's objectives and goals.

A. Quality Models Of ISO 9126

The quality model presented in standards of ISO/IEC 9126-1, distinguishes software quality in a structured set of characteristics and sub-characteristics:

- **Functionality** – *It is a set of attributes that can bear the existence of a set of functions and their specified properties. These functions are those that satisfy stated or implied needs.*
  - Suitability
  - Accuracy
  - Interoperability
  - Security
  - Functionality Compliance
- **Reliability** – It is a *set of attributes that bear the capability of software to maintain its level of performance under stated conditions for a stated period of time.*
  - Maturity
  - Fault Tolerance
  - Recoverability
  - Reliability Compliance
- **Usability** – *It is a set of attributes that bear the effort needed for use and on the individual assessment of such use by a stated or implied set of users.*
  - Understandability
  - Learnability
  - Operability
  - Attractiveness
  - Usability Compliance
- **Efficiency** – It is a *set of attributes that bear the relationship between the level of performance of the software and the amount of resources used under stated conditions.*
  - Time Behavior
  - Resource Utilization
  - Efficiency Compliance
- **Maintainability** – It is a *set of attributes that bear the effort needed to make specified modifications.*
  - Analyzability
  - Changeability
  - Stability
  - Testability
  - Maintainability Compliance
- **Portability** – It is a *set of attributes that bear the ability of software to be transferred from one environment to another.*
  - Adaptability
  - Install ability
  - Co-Existence
  - Replace ability
  - Portability Compliance

Each sub-characteristic (e.g. adaptability) is further divided into attributes. An entity in software product that can be verified and

measured is called as an Attribute. As they vary between different products, they are not defined in the standard. Software product encompasses executables, source code, architecture and descriptions. For organizations to define a quality model for a software product, the above standard provides a framework. However, by doing so every organization can precisely specify its own model. This can be done by specifying the target values for quality metrics.

B. Internal Metrics Of ISO 9126

The metrics which do not rely on software execution, static measure are known as Internal Metrics.

C. External Metrics Of ISO 9126

The metrics which are applicable to running software are External Metrics.

D. Quality In Use Metrics Of ISO 9126

When the final product is used in real conditions then the Quality in use metrics come into existence. The quality in use is determined by the external quality and external quality is determined by the internal quality. This method of determination is derived from the GE Model, Presented by McCall et al. in 1977.

Following are the three types of Quality Characteristics:

- Factors (To specify): They describe the external view of the software, as viewed by the users.
- Criteria (To build): They describe the internal view of the software, as seen by the developer.
- Metrics (To control): They are defined and used to provide a scale and method for measurement.

ISO/IEC 9126 distinguishes between a defect and nonconformity. A defect being the nonfulfillment of intended usage requirements, whereas a nonconformity is the nonfulfillment of specified requirements. A similar divergence is made between validation and verification which is known as V&V in the testing trade.

IV. QUALITY STANDARDS

- The Quality Management System(QMS) standards was designed by the International Organization for Standardization(ISO) in 1987.These standards were a series of standards i.e ISO 9000:1987 comprising of ISO 9001:1987,ISO 9002:1987 and ISO 9000:1987.They were applicable in different types of industries which are based on the type of activity or process designing, production and service of delivery.
- The International Organization for Standardization reviews the standards every few years. The 1994 version was called the ISO 9000:1994 series which comprised the ISO 9001:1994, ISO 9002:1994 and ISO 9003:1994 versions.
- The last revision done in the year 2008 gave the series a name ISO 9000:2000 series. A certified standard i.e ISO 9001:2000 is an integrated version of the ISO 9002 and 9003 standards.
- A minor revision was released by ISO named ISO 9001:2008 on October 14, 2008. This standard does not contain any new requirements. Most of the changes that were made were to improve the consistency in grammar which further made translation of the standard into other languages possible.
- The performance improvement guidelines over and above the basic standard ISO 9001:2000 are provided by the ISO 9004:2009 document. The standard further provides a framework for improved quality management that is similar to and based upon the measurement framework for process assessment.
- The standards for quality management are created by ISO and are to certify the processes also the organizations system. The product quality or service is not certified by ISO 9000 standards.
- ISO 22000 Standard was released in 2005 by the International Organization for Standardization which meant for the food industry.

- ISO 22000 includes the principles and values of ISO 9000 and the HACCP standards. This is the only integrated standard released for the food industry and will become popular in coming years.

V. SOFTWARE QUALITY MEASUREMENT TECHNIQUE

Quantification to what extent the system or a software consists the desirable characteristics is referred to Software quality measurement. This can be done by qualitative or quantitative means or by using the both. In both the cases, for every desirable characteristic, there exists a set of measurable attributes. The existence of these attributes which tend to be correlated and associated with this characteristic. For example, the number of target-dependent statements in a program relates to, an attribute associated with portability. These measurable attributes are the "hows" that need to be enforced to enable the "what's" in the Software Quality definition, which are précised using the Quality Function Development approach.

The attributes and metrics that are applicable to software quality management have been extracted from the ISO 9126-3 and further ISO/IEC 25000:2005 quality model. Internal Structural Quality bears the main focus. Subcategories have been created .To handle specific areas consisting of the business application architecture and technical characteristics, subcategories have been devised. Such as data access, manipulation or the notion of transactions are technical characteristics.

An article has been published by OMG which is one of the founding member of the Consortium for IT Software Quality(CISQ), which says "How to Deliver Resilient, Secure, Efficient, and Easily Changed IT Systems in Line with CISQ Recommendations" which states that basic code errors are 92% of the total errors in the source code. Only 10% of the defects are accounted by various code level issues eventually.

A. Code-Based Analysis

Many of the existing software measures count structural elements of the application that result from parsing the source code for such individual instructions (Park, 1992), tokens (Halstead, 1977), control structures (McCabe, 1976) and objects (Chidamber & Kemerer, 1994).

Quantification to what extent the system or a software consists the desirable characteristics is referred to Software quality measurement This can be done by qualitative or quantitative means or by using the both. In both the cases, for every desirable characteristic, there exists a set of measurable attributes. The existence of these attributes which tend to be correlated and associated with this characteristic.

The quality of structure of software is analyzed and measured through analysis of the source code, the architecture, the software framework, and the database schema in relationship to the principles and the standards that will together describe the logical and conceptual architecture of the system. This is distinct from the basic, local, component-level code analysis typically performed by development tools which are mostly concerned with implementation considerations and are crucial during debugging and testing activities.

B. Reliability

The main causes of the poor reliability are found in combination of non-compliance and good architectural and the coding practices. Measurement of the static quality attributes of an application helps in detecting the non-compliance. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

Application Architecture Practices
Coding Practices
Complexity of algorithms
Complexity of programming practices
Compliance with Object-Oriented and Structured Programm
best practices (when applicable)
Component or pattern re-use ratio

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered software.

C. Efficiency

As with Reliability, the causes of performance inefficiency are often found in violations of good architectural and coding practice which can be detected by measuring the static quality attributes of an application. These static attributes predict potential operational performance bottlenecks and future scalability problems, especially for applications requiring high execution speed for handling complex algorithms or huge volumes of data.

Assessing performance efficiency requires checking at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Appropriate interactions with expensive and/or remote resources
- Data access performance and data management
- Memory, network and disk space management
- Coding Practices
- Compliance with Object-Oriented and Structured Programming best practices (as appropriate)
- Compliance with SQL programming best practices

D. Security

Poor coding and architectural practices such as SQL injection or cross-site scripting result in most security vulnerabilities. The lists maintained by CWE and the SEI/Computer Emergency Center (CERT) at Camegie Mellon University consist of these vulnerabilities.

Assessing security requires at least checking the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Multi-layer design compliance
- Security best practices (Input Validation, SQL Injection, Cross-Site Scripting, etc.)
- Programming Practices (code level)
- Error & Exception handling
- Security best practices (system functions access, access control to programs)

E. Maintainability

Maintainability includes concepts of modularity, understandability, changeability, testability, reusability, and transferability from one development team to another. These do not take the form of critical issues at the code level. Rather, poor maintainability is typically the result of thousands of minor violations with best practices in documentation, complexity avoidance strategy, and basic programming practices that make the difference between clean and easy-to-read code vs. unorganized and difficult-to-read code.

Assessing maintainability requires checking the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Architecture, Programs and Code documentation embedded in source code
- Code readability
- Complexity level of transactions
- Complexity of algorithms
- Complexity of programming practices
- Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Component or pattern re-use ratio
- Controlled level of dynamic coding

Maintainability is closely related to Ward Cunningham's concept of technical debt, which is an expression of the costs resulting of a lack of maintainability. Reasons for why maintainability is low can be classified as reckless vs. prudent and deliberate vs. inadvertent, and often have their origin in developers' inability, lack of time and goals, their carelessness and discrepancies in the creation cost of and benefits from documentation and, in particular, maintainable source code.

F. Size

Measuring software size requires that the whole source code be correctly gathered, including database structure scripts, data manipulation source code, component headers, configuration files etc. There are essentially two types of software sizes to be measured, the technical size (footprint) and the functional size:

- There are several software technical sizing methods that have been widely described. The most common technical sizing method is number of Lines Of Code (#LOC) per technology, number of files, functions, classes, tables, etc.,

from which backfiring Function Points can be computed;

- The most common for measuring functional size is Function Point Analysis. Function Point Analysis measures the size of the software deliverable from a user's perspective. Function Point sizing is done based on user requirements and provides an accurate representation of both size for the developer/estimator and value (functionality to be delivered) and reflects the business functionality being delivered to the customer. The method includes the identification and weighting of user recognizable inputs, outputs and data stores. The size value is then available for use in conjunction with numerous measures to quantify and to evaluate software delivery and performance (Development Cost per Function Point; Delivered Defects per Function Point; Function Points per Staff Month.).

The Function Point Analysis sizing standard is supported by the International Function Point Users Group (IFPUG). It can be applied early in the software development life-cycle and it is not dependent on lines of code like the somewhat inaccurate Backfiring method. The method is technology agnostic and can be used for comparative analysis across organizations and across industries.

Since the inception of Function Point Analysis, several variations have evolved and the family of functional sizing techniques has broadened to include such sizing measures as COSMIC, NESMA, Use Case Points, FP Lite, Early and Quick FPs, and most recently Story Points. However, Function Points has a history of statistical accuracy, and has been used as a common unit of work measurement in numerous application development management (ADM) or outsourcing engagements, serving as the "currency" by which services are delivered and performance is measured.

One common limitation to the Function Point methodology is that it is a manual process and therefore it can be labor-intensive and costly in large scale initiatives such as application development or outsourcing engagements. This negative aspect of applying the methodology may be what motivated industry IT leaders to form the Consortium for IT Software Quality focused on introducing a computable metrics standard for automating the measuring of software size while the IFPUG keep promoting a manual approach as most of its activity rely on FP counters certifications.

CISQ announced the availability of its first metric standard, Automated Function Points,to the CISQ membership, in CISQ Technical. These recommendations have been developed in OMG's Request for Comment format and submitted to OMG's process for standardization.

G. Identifying Critical Programming Errors

Critical Programming Errors are specific architectural and/or coding bad practices that result in the highest, immediate or long term, business disruption risk.

These are quite often technology-related and depend heavily on the context, business objectives and risks. Some may consider respect for naming conventions while others – those preparing the ground for a knowledge transfer for example – will consider it as absolutely critical.

Critical Programming Errors can also be classified per CISQ Characteristics. Basic example below:

- Reliability
  - Avoid software patterns that will lead to unexpected behavior (Uninitialized variable, null pointers, etc.)
  - Methods, procedures and functions doing Insert, Update, Delete, Create Table or Select must include error management
  - Multi-thread functions should be made thread safe, for instance servlets or struts action classes must not have instance/non-final static fields
- Efficiency
  - Ensure centralization of client requests (incoming and data) to reduce network traffic
  - Avoid SQL queries that don't use an index against large tables in a loop
- Security
  - Avoid fields in servlet classes that are not final static
  - Avoid data access without including error management
  - Check control return codes and implement error handling mechanisms
  - Ensure input validation to avoid cross-site scripting flaws or SQL injections flaws
- Maintainability
  - Deep inheritance trees and nesting should be avoided to improve comprehensibility
  - Modules should be loosely coupled (fanout, intermediaries,) to avoid propagation of modifications
  - Enforce homogeneous naming conventions

VI. PRINCIPLE OF QUALITY MANAGEMENT

- The International Standard for Quality management (ISO 9001:2008) adopts a number of management principles that can be used by top management to guide their organizations towards improved performance.

A. *Customer Focus*

Since the organizations depend on their customers, they should understand current and future customer needs, should meet customer requirements and should try to exceed the expectations of customers. An organization attains customer focus when all people in the organization know both the internal and external customers and also what customer requirements must be met to ensure that both the internal and external customers are satisfied.

B. *LEADERSHIP*

Leaders of an organization establish unity of purpose and direction of it.They should go for creation and maintenance of such an internal environment, in which people can become fully involved in achieving the organization's quality objective.

C. *INVOLVEMENT OF PEOPLE*

People at all levels of an organization are the essence of it. Their complete involvement enables their abilities to be used for the benefit of the organization.

D. *PROCESS APPROACH*

The desired result can be achieved when activities and related resources are managed in an organization as a process.

E. *SYSTEM APPROACH TO MANAGEMENT*

An organization's effectiveness and efficiency in achieving its quality objectives are contributed by identifying, understanding and managing all interrelated processes as a system. Quality Control involves checking transformed and transforming resources in all stages of production process.

F. *CONTINUAL IMPROVEMENT*

The permanent quality objective of an organization should be the continual improvement of its own overall performance, leveraging clear and concise PPMs (Process Performance Measures).

G. *FACTUAL APPROACH TO DECISION MAKING*

Data analysis and information form the basis of Effective decisions.

H. *MUTUALLY BENEFICIAL SUPPLIER RELATIONSHIPS*

The above mentioned eight principles form the basis for the quality management system Standard i.e ISO 9001:2008. Since the organization and suppliers are interdependent, therefore, a mutually beneficial relationship between them increases the ability of both to add value.

VII. SUMMARY

Software quality is a branch of study and practice which outlines the advantageous attributes of software products.

There are two approaches to software quality that are prevalent:

- Defect Management Approach
  Any failure to address end user requirements is regarded as Software Defect. The common defects can be missed or misunderstood requirements and the errors in software design, functional logic, data relationships, validity checking, process timing and coding and so on.
  The defect management approach progresses by counting and managing defects. Herein the defects are categorized by severity and the numbers from every category are used for planning. Tools such as Defect Leakage Matrices-for counting the number of defects that pass through development phases prior to detection are used by more mature software development organizations. Also the control charts are used to measure and improve the development process capability.
- Quality Attributes Approach
  The above approach to quality of software is best This approach to software quality is

best represented by fixed quality models which are as ISO/IEC 9126. The ISO/IEC 9126 defines six quality characteristics-each composed of sub-categories:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

ACKNOWLEDGMENTS

We would like to thank our guides for their timely help, giving interesting ideas and encouragement to finish this research work successfully.

SIDE BAR

Comparison: Comparison is an act of assessment or evaluation of things simultaneously in order to analyze, exactly to what extent they are similar or different. It is used to draw the comparison between two things of same type mostly to discover essential features or meaning either scientifically or otherwise.

Content: The amount of things contained in something. Content are those things written or spoken in a book, an article, a programme, a speech and so on.

REFERENCES

1) CISQ 2009 Executive Forums Report
2) McConnell, Steve (1993), Code Complete (First ed.), Microsoft Press]
3) Crosby, P., *Quality is Free*, McGraw-Hill, 1979
4) DeMarco, T., *Management Can Make Quality (Im)possible*, Cutter IT Summit, Boston, April 1999
5) Weinberg, Gerald M. (1992), *Quality Software Management: Volume 1, Systems Thinking*, New York, NY: Dorset House Publishing, p. 7
6) Weinberg, Gerald M. (1993), *Quality Software Management: Volume 2, First-Order Measurement*, New York, NY: Dorset House Publishing, p. 108
7) http://www.omg.org/CISQ_compliant_IT_Systemsv.4-3.pdf
8) Park, R.E. (1992). Software Size Measurement: A Framework for Counting Source Statements. (CMU/SEI-92-TR-020). Software Engineering Institute, Carnegie Mellon University
9) Halstead, M.E. (1977). Elements of Software Science. Elsevier North-Holland.
10) Chidamber, S. & C. Kemerer. C. (1994). A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20 (6), 476-493
11) Nygard, M.T. (2007). Release It! Design and Deploy Production Ready Software. The Pragmatic Programmers.
12) Martin, R. (2001). Managing vulnerabilities in networked systems. IEEE Computer.
13) Boehm, B., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., & Merritt, M.J. (1978). Characteristics of Software Quality. North-Holland.
14) "CWE - Common Weakness Enumeration". Cwe.mitre.org. Retrieved 2013-10-18.
15) "CWE's Top 25". Sans.org. Retrieved 2013-10-18.

RELATED REFERENCE

1. Rose, Kenneth H. (July 2005). *Project Quality Management: Why, What and How*. Fort Lauderdale, Florida: J. Ross Publishing. p. 41. ISBN 1-932159-48-7.
2. Paul H. Selden (December 1998). "**Sales Process Engineering: An Emerging Quality Application**". *Quality Progress*: 59–63.
3. Quality Management Strategy, May 2010.
4. Cianfrani, Charles A.; West, John E. (2009). *Cracking the Case of ISO 9001:2008 for Service: A Simple Guide to Implementing Quality Management to Service Organizations* (2nd ed.). Milwaukee: American Society for Quality. pp. 5–□"7. ISBN 978-0-87389-762-4.
5. Westcott, Russell T. (2003). *Stepping Up To ISO 9004: 2000 : A Practical Guide For Creating A World-class Organization*. Paton Press. p. 17. ISBN 0-9713231-7-8.
6. "Taking the First Step with PDCA". 2 February 2009. Retrieved 17 March 2011.
7. "Object Oriented Quality Management, a model for quality management.". Statistics Netherlands, The Hague. 2009-04-29.
8. http://ssrn.com/abstract=1488690 "Thareja" Thareja P(2008), "Total Quality Organization Thru' People, Each one is Capable", FOUNDRY, Vol. XX, No. 4, July/Aug 2008
9. "ISO 9001 Quality Management System QMS Certification". Indian Register Quality Systems. Retrieved 13 March 2014.
10. Littlefield, Matthew; Roberts, Michael (June 2012). "Enterprise Quality Management Software Best Practices Guide". *LNS Research Quality Management Systems*: 10.