MCA Part-II

Paper-XI: Software Engineering

Topic: Software Quality Assurance

PREPARED BY: DR. KIRAN PANDEY

(School of Computer Science)

Email-id: kiranpandey.nou@gmail.com

INTRODUCTION

It is also monitoring the processes and products throughout the SDLC. The objective of the software quality assurance process is to produce high quality software that meets customer requirements through a process of applying various procedures and standards.

Software quality is defined as:

Conformance to the explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed S/W.

This implies the existence of a set of standards used by the developer and customer expectations that a product will work well. Conformance to implicit requirements (e.g. ease of use and reliable performance) is what sets software engineering apart from simply writing programs that work most of the time. Several sets of software quality factors are described.

The definition serves to emphasize three important points:

1. S/W requirements are the foundation from which quality is measured. Lack of conformance to requirements. is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which S/W is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. There is a set of implicit requirements that often goes unmentioned. If S/W conforms to its explicit requirements but fails to meet implicit requirements, S/W quality is suspect.

Attributes of Quality

The following are some of the attributes of quality:

Auditability: The ability of software being tested against conformance to standard.

Compatibility: The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.

Completeness: The degree to which all of the software's required functions and design constraints are present and fully developed in the requirements specification, design document and code.

Consistency: The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.

Correctness: The degree to which a system or component is free from faults in its specification, design, and implementation. The degree to which software, documentation, or other items meet specified requirements.

Feasibility: The degree to which the requirements, design, or plans for a system or component can be implemented under existing constraints.

Modularity: The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Predictability: The degree to which the functionality and performance of the software are determinable for a specified set of inputs.

Robustness: The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

Structured-ness: The degree to which the SDD (System Design Document) and code possess a definite pattern in their interdependent parts. This implies that the design has proceeded in an orderly and systematic manner (e.g., top-down, bottom-up). The modules are cohesive and the software has minimized coupling between modules.

Testability: The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

Traceability: The degree to which a relationship can be established between two or more products of the development process. The degree to which each element in a software development product establishes its reason for existing (e.g., the degree to which each element in a bubble chart references the requirement that it satisfies). For example, the system's functionality must be traceable to user requirements.

Understandability: The degree to which the meaning of the SRS, SDD, and code are clear and understandable to the reader.

Verifiability: The degree to which the SRS, SDD, and code have been written to facilitate verification and testing.

Causes of error in Software

- Misinterpretation of customers' requirements/communication
- Incomplete/erroneous system specification
- Error in logic
- Not following programming/software standards
- Incomplete testing
- Inaccurate documentation/no documentation
- Deviation from specification
- Error in data modeling and representation.

Mc Call's Quality Factors

McCall's quality factors were proposed in the early 1970s. They are as valid today as they were in that time. It's likely that software built to conform to these factors will exhibit high quality well into the 21st century, even if there are dramatic changes in technology. The factors that affect S/W quality can be categorized in two broad groups:

1. factors that can be directly measured (defects uncovered during testing)
2. factors that can be measured only indirectly (Usability and maintainability)

McCall proposed a useful categorization of factors that affect software quality. These software quality factors, shown in Figure below, focus on three important aspects of a software product:

(a) its operational characteristics,
(b) its ability to undergo change,
(c) its adaptability to new environments.

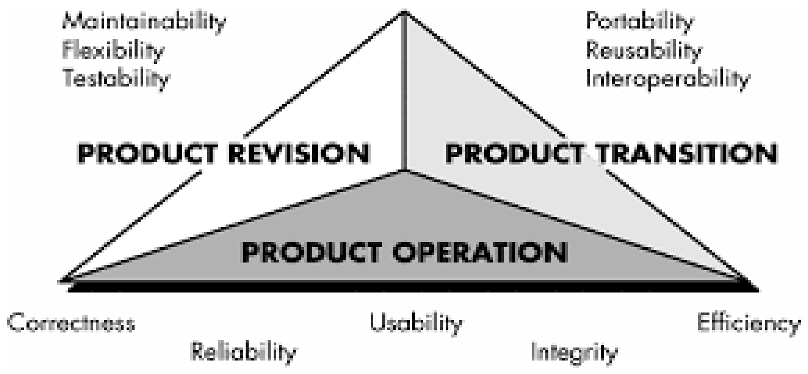

Figure 1: Mc Call's Quality factors

Referring to these factors, McCall and his colleagues provide the following descriptions:

Correctness: The extent to which a program satisfies its specs and fulfills the customer's mission objectives.

Reliability: The extent to which a program can be expected to perform its intended function with required precision.

Efficiency: The amount of computing resources and code required to perform is function.

Integrity: The extent to which access to S/W or data by unauthorized persons can be controlled.

Usability: The effort required to learn, operate, prepare input for, and interpret output of a program.

Maintainability: The effort required to locate and fix errors in a program.

Flexibility: The effort required to modify an operational program.

Testability: The effort required to test a program to ensure that it performs its intended function.

Portability: The effort required to transfer the program from one hardware and/or software system environment to another.

Reusability: The extent to which a program can be reused in other applications-related to the packaging and scope f the functions that the program performs.

Interoperability: The effort required to couple one system to another.

Other Quality Factors

There are various quality factors defined as:

(a) ISO 9126 Quality Factor

ISO 9126 is an international standard for the evaluation of software quality. The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

- **Functionality** - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
    - Suitability
    - Accuracy
    - Interoperability
    - Compliance
    - Security

- **Reliability** - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
    - Maturity
    - Recoverability

- **Usability** - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
    - Learnability
    - Understandability
    - Operability

- **Efficiency** - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
    - Time Behavior
    - Resource Behavior

- **Maintainability** - A set of attributes that bear on the effort needed to make specified modifications.
    - Stability
    - Analyzability
    - Changeability
    - Testability

- **Portability** - A set of attributes that bear on the ability of software to be transferred from one environment to another.
    - Installability

- Replaceability
- Adaptability

(b) Targeted Quality Factors

The quality dimensions and factors presented above focus on the software as a whole and can be used as a generic indication of the quality of an application. A software team can develop a set of quality characteristics and associated questions that would probe3 the degree to which each factor has been satisfied. For example, McCall identifies *usability* as an important quality factor. If you were asked to review a user interface and assess its usability, how would you proceed? You might start with the sub-attributes suggested by McCall—understandability, learnability, and operability—but what do these mean in a pragmatic sense? To conduct your assessment, you'll need to address specific, measurable (or at least, recognizable) attributes of the interface. For example:

Intuitiveness. The degree to which the interface follows expected usage patterns so that even a novice can use it without significant training.
• Is the interface layout conducive to easy understanding?
• Are interface operations easy to locate and initiate?
• Does the interface use a recognizable metaphor?
• Is input specified to economize key strokes or mouse clicks?
• Does the interface follow the three golden rules?
• Do aesthetics aid in understanding and usage?

Efficiency. The degree to which operations and information can be located or initiated.
• Does the interface layout and style allow a user to locate operations and information efficiently?
• Can a sequence of operations (or data input) be performed with an economy of motion?
• Are output data or content presented so that it is understood immediately?
• Have hierarchical operations been organized in a way that minimizes the depth to which a user must navigate to get something done?

Robustness. The degree to which the software handles bad input data or inappropriate
user interaction.
• Will the software recognize the error if data at or just outside prescribed boundaries is input? More importantly, will the software continue to operate without failure or degradation?
• Will the interface recognize common cognitive or manipulative mistakes and explicitly guide the user back on the right track?
• Does the interface provide useful diagnosis and guidance when an error condition (associated with software functionality) is uncovered?

Richness. The degree to which the interface provides a rich feature set.
• Can the interface be customized to the specific needs of a user?
• Does the interface provide a macro capability that enables a user to identify a sequence of common operations with a single action or command?
As the interface design is developed, the software team would review the design prototype and ask the questions noted. If the answer to most of these questions is "yes," it is likely that the user interface exhibits high quality. A collection of questions similar to these would be developed for each quality factor to be assessed.

ELEMENTS OF SOFTWARE QUALITY ASSURANCE

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality. These can be summarized in the following manner:

Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers.
Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

Testing. Software testing is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices (Chapter 22) have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

Vendor management. Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft *Office*), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and *contracted software* that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

Security management. With the increase in cybercrime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety. Because software is almost always a pivotal component of human rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
Risk management. Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established. In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals)
are conducted or produced with quality as a dominant concern.