Chapter 10 : Quality Management

- Compliled by Roshan Chitrakar

Software Quality Definition

The Software Quality conforms to some explicit characteristics as well as implicit characteristics.

These are:-

- **Explicit Characteristics:**

  The software should explicitly conform to the stated functional and performance requirements. The software should conform explicitly as per the documented development standards

- **Implicit Characteristics:**

  The implicit characteristics of software quality are those expected from all professionally developed software.

This above definition of software quality emphasizes the following three points.

1. Software requirements are the basic foundations from which quality of software is measured. Lack of conformance to the requirement is lack of quality.
2. Software Development criteria comprise of a set of specified standards that guide the manner in which the software is engineered. If the criteria are not followed, software quality will definitely be lost.
3. There is a set of implicit requirements like scope and desire for good maintainability, even if not mentioned, matter a lot towards the software quality.

   If software does not conform to the implicit requirements, no matter how strongly it meets the explicit requirements, the quality of the software will be suspected.

Quality Concepts

Variation control is the heart of quality control (software engineers strive to control the process applied, resources expended, and end product quality attributes). The tape duplication example.

Quality of design - refers to characteristics designers specify for the end product to be constructed. The grade of materials, tolerances, and performance specifications all contribute to the quality of design.

Quality of conformance - degree to which design specifications are followed in manufacturing the product. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

Quality control - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications. Quality control includes a feedback loop to the process that created the work product.

Quality assurance - consists of the auditing and reporting procedures used to provide management with data needed to make proactive (practical and positive) decisions.

Cost of Quality – It includes all the costs incurred in the pursuit of quality or in performing quality related activities. It provides a baseline for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison. Cost of quality includes:

- **Prevention costs** - Includes quality planning, formal technical reviews, test equipment, training.
- **Appraisal costs** – Includes in-process and inter-process inspection, equipment calibration and maintenance, testing. (Appraisal means evaluation or assessment)
- **Failure costs** – Includes rework, repair, and failure mode analysis.
- **External failure costs** – Includes complaint resolution, product return and replacement, help line support, warranty work.

Software Quality Assurance

Conformance to software requirements is the foundation from which software quality is measured. Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered. Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

The three important point of concern for SQA are:

- Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
- Specific standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.

- A set of implicit requirements often goes unmentioned e.g., the desire for ease of use and good maintainability. If the software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspected.

SQA Group Activities

- Prepare SQA plan for the project.
- Participate in the development of the project's software process description.
- Review software engineering activities to verify compliance with the defined software process.
- Audit designated software work products to verify compliance with those defined as part of the software process.
- Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
- Record any evidence of noncompliance and reports them to management.

Software Reviews

The purpose is to find defects (errors) before they are passed on to another software engineering activity or released to the customer. Software engineers (and others) conduct formal technical reviews (FTR) for software engineers. Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

Formal Technical Reviews

- Involves 3 to 5 people (including reviewers)
- Advance preparation (no more than 2 hours per person) required
- Duration of review meeting should be less than 2 hours
- Focus of review is on a discrete work product
- Review leader organizes the review meeting at the producer's request
- Reviewers ask questions that enable the producer to discover his or her own error (the product is under review not the producer)
- Producer of the work product walks the reviewers through the product
- Recorder writes down any significant issues raised during the review

- Reviewers decide to accept or reject the work product and whether to require additional reviews of product or not

For software industry to be more quantitative about quality, the statistical quality assurance implies the following steps:

1. Information about the software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause.

   [e.g. non-conformance to specification, design error, validation of standards, poor communication with customer, etc.]
3. Using the Pareto principle, isolate the 20% of the most vital errors.
4. Now, move to correct the problems that have caused the defects.

In view of the above steps, during software development cycle, various errors encountered can be attached to one of more of the following causes: -

- IES = Incomplete or Erroneous specification
- MCC= Misinterpretation of Customer Communication.
- IDS = Intentional Deviation from Specification.
- VPS = Violation of Programming Standards.
- EDR= Error in Data Representation.
- IMI= Inconsistent Module Interface
- EDL = Error in Design Logic
- IET = Incomplete or Erroneous Testing
- IID = Inaccurate or Incomplete Documentation
- PLT=error in Programming Language Translation of design
- HCI = ambiguous or inconsistent Human-Computer Interface.
- MIS= Miscellaneous

It is seen that the causes IES, MCC, EDR and IET account for most of the major errors. Once the vital few causes are located, the software development origination can plan for necessary corrective actions.

In addition to the collection of defect information, software developer can calculate Defect Index (DI) for each major step in the software development process. After analysis, design, coding, testing and release, the following data are gathered: -

Di = total number of defects encountered during i^{th} please of the software engineering process.

Si = number of serious defects

Mi = number of moderate defects

Ti= number of minor (trivial) defects.

PS= Size of the product (LOC, design statements, pages of documentation) at i^{th} step.

wi = Weighting factor for serious, moderate, and trivial defects, where j= 1 to 3

At each step of the software development t process, a phase index PI_i is competed, as follows: -

PI = w1(si/Di) + w2(Mi/Di) + w3(Ti/Di)

Finally, the defect index DI_i is calculated by computing the cumulative effect of each PI_i, giving higher weights to the errors encountered later in the software engineering process than those encountered earlier.

$DI= \sum(i*PI_i)/PS$

$= (PI_1+2PI_2+3PI_3+.....+ iPI_i)/PS$

Software Reliability

The reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are accepted.

Software reliability can be measured directly and estimated using historical and developmental data. Software reliability is defined in statistical terms as "The probability of failure free operation of a computer program in a specified environment for a specified time."

Software Reliability Index

In context of a computer-based system, a simple measure of the software reliability is Mean Time Between Failure (MTBF),

MTBF= MTTF + MTTR

Where, MTTF = Mean Time To Failure,

MTTR = Mean Time to Recover/Repair

In addition to the reliability measure, a measure of the availability is also important. Software reliability is defined as “The probability that the program is operating according to requirements at a given point of time"

Software Availability is given by:-

Availability = MTTF/(MTTF+MTTR)*100%

Six Sigma for Software Engineering

Six Sigma is the most widely used strategy for statistical quality assurance in industry today. Originally popularized by Motorola in the 1980s, the Six Sigma strategy “is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company’s operational performance by identifying and eliminating defects’ in manufacturing and service-related processes”. The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard. The Six Sigma methodology defines three core steps:

1. Define customer requirements and deliverables and project goals via well-defined methods of customer communication.
2. Measure the existing process and its output to determine current quality performance (collect defect metrics).
3. Analyze defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:

- Improve the process by eliminating the root causes of defects.
- Control the process to ensure that future work does not reintroduce the causes of defects.

These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method. If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows:

- Design the process to (1) avoid the root causes of defects and (2) to meet customer requirements.
- Verify that the process model will, in fact, avoid defects and meet customer requirements.

This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.

Measuring Software Quality

Software Quality is a complex mix of functions that vary according to the following: -

- Applications requirements, and
- Customer's request of requirements

The following two categories of measurable factors affect the software quality:

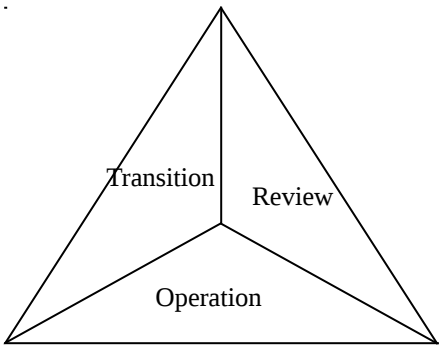
- Factors that can be directly measured: -
    1. Errors
    2. KLOC
    3. Unit time
- Factors that can only be measured indirectly: -
    1. Usability
    2. Maintainability

However, in each case, measurement must be done. The software (documents, programs, and data) must be compared against some information to arrive at an indicator of the quality.

McCall Categorization of Software Quality

As per McCall, the software quality emphasizes the following three major aspects of software product. These are: -

1) Product operations - The operational characteristics of the software
2) Product transition - the ability to undergo changes
3) Product revision - the scope of adaptability to new environments



Above three aspects are all equally important. The three aspects provide the following descriptions as per McCall.

Product Operation comprises of the following descriptions: -

- Correctness - (Does the software do what the customer need?)
- Reliability - (Does the software perform the task accurately at all time?)
- Efficiency - (Does the software run on user’s hardware efficiently?)

- Integrity - (Is the functionality of the software secured in respect of unauthorized user's access?)
- Usability - (Is the software developed keeping user's convenience in view e.g. to learn, to operate, etc.?)

Product Revision aspect comprises of the following descriptions: -

- Maintainability - If any error is occurred, can the user locate it and fix it?
- Flexibility - Can the user make necessary changes to incorporate his changing requirement?
- Testability - Can the user make necessary test to ascertain the functionality?

Product Transition aspect comprises of the following descriptions: -

- Portability – Whether the user can use the software on another hardware setup?
- Reusability - Will the user be able to reuse full or part of the software?
- Inter operability - Will the user be able to interface the software with another system?

However, in most of the cases, it is impossible to develop direct measurement of the above quality factors as proposed by McCall. Therefore, a set of metrics are defined and used to develop expressions for each of the factors according to the following relationship:

$$F_q = c_1 \times m_1 + c_2 \times m_2 + \ldots\ldots\ldots\ldots\ldots + c_n \times m_n$$

where,

F_q = Software quality factor

c_n = The regression factor, and

m_n = metrics that affect the quality factor

However, the metrics defined above are not practicable to measure; rather they can only be measured subjectively. The metrics may be kept in the form of check list that is used to grade specific attributes of the software, of the basis of 10 point grading scale.

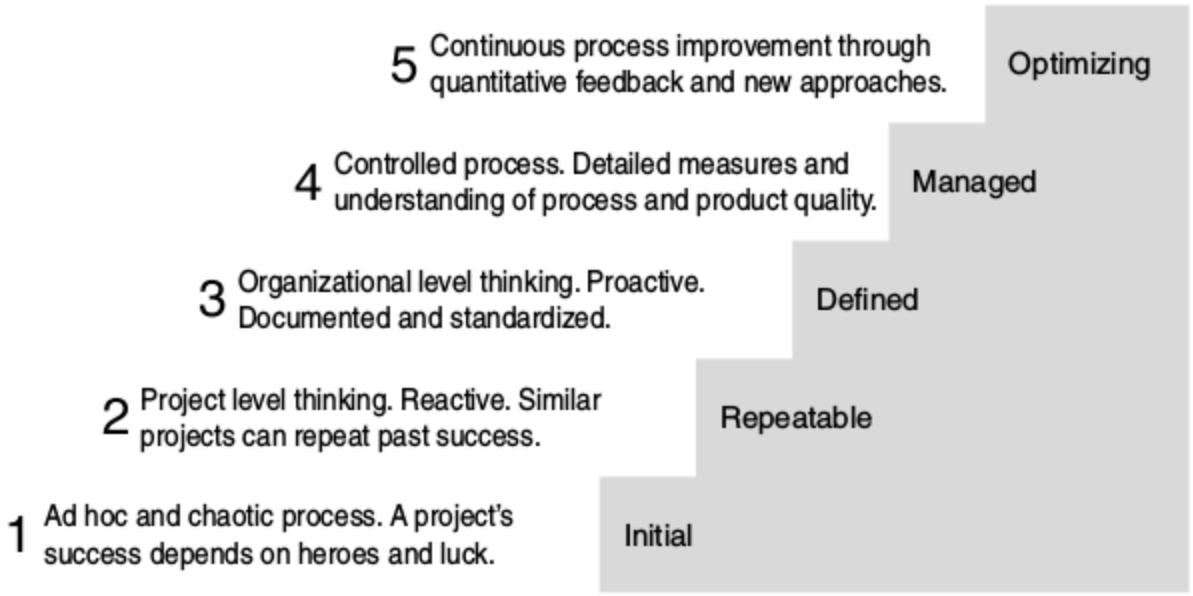
Software Quality Standards

There are different standards related to software quality. CMM and ISO are described in this section.

Capability Maturity Model (CMM)

The Capability Maturity Model for Software (CMM or SW-CMM) is an industry-standard model for defining and measuring the maturity of a software company’s development process and for providing direction on what they can do to improve their software quality. It was developed by the software

development community along with the Software Engineering Institute (SEI) and Carnegie Mellon University, under direction of the U.S. Department of Defense.



Here are descriptions of the five CMM Maturity Levels:• Level 1: Initial. The software development processes at this level are ad hoc and often chaotic. The project's success depends on heroes and luck. There are no general practices for planning, monitoring, or controlling the process. It's impossible to predict the time and cost to develop the software. The test process is just as ad hoc as the rest of the process.

• Level 2: Repeatable. This maturity level is best described as project-level thinking. Basic project management processes are in place to track the cost, schedule, functionality, and quality of the project. Lessons learned from previous similar projects are applied. There's a sense of discipline. Basic software testing practices, such as test plans and test cases, are used.

• Level 3: Defined. Organizational, not just project specific, thinking comes into play at this level. Common management and engineering activities are standardized and documented. These standards are adapted and approved for use on different projects. The rules aren't thrown out when things get stressful. Test documents and plans are reviewed and approved before testing begins. The test group is independent from the developers. The test results are used to determine when the software is ready.

• Level 4: Managed. At this maturity level, the organization's process is under statistical control. Product quality is specified quantitatively beforehand (for example, this product won't release until it has fewer than 0.5 defects per 1,000 lines of code) and the software isn't released until that goal is met. Details of the development process and the software's quality are collected over the project's development, and adjustments are made to correct deviations and to keep the project on plan.

• Level 5: Optimizing. This level is called Optimizing (not "optimized") because it's continually improving from Level 4. New technologies and processes are attempted, the results are measured, and both incremental and revolutionary changes are instituted to achieve even better quality levels. Just when everyone thinks the best has been obtained, the crank is turned one more time, and the next level of improvement is obtained.

ISO 9000

ISO is an international standards organization that sets standards for everything from nuts and bolts to, in the case of ISO 9000, quality management and quality assurance.

ISO 9000 is a family of standards on quality management and quality assurance that defines a basic set of good practices that will help a company consistently deliver products (or services) that meet their customer's quality requirements.

It's impossible to detail all the ISO 9000 requirements for software in this chapter, but the following list will give you an idea of what types of criteria the standard contains.

- Develop detailed quality plans and procedures to control configuration management, product verification and validation (testing), nonconformance (bugs), and corrective actions (fixes).
- Prepare and receive approval for a software development plan that includes a definition of the project, a list of the project's objectives, a project schedule, a product specification, a description of how the project is organized, a discussion of risks and assumptions, and strategies for controlling it.
- Communicate the specification in terms that make it easy for the customer to understand
- and to validate during testing.

- Plan, develop, document, and perform software design review procedures.
- Develop procedures that control software design changes made over the product’s life cycle.
- Develop and document software test plans.
- Develop methods to test whether the software meets the customer’s requirements.
- Perform software validation and acceptance tests.
- Maintain records of the test results.
- Control how software bugs are investigated and resolved.
- Prove that the product is ready before it’s released.
- Develop procedures to control the software’s release process.
- Identify and define what quality information should be collected.
- Use statistical techniques to analyze the software development process.
- Use statistical techniques to evaluate product quality.