SOFTWARE ENGINEERING
SEM-6TH
SECTION-D (NOTES)

SOFTWARE QUALITY:

In the context of **software** engineering, **software quality** refers to two related but distinct notions that exist wherever **quality** is defined in a business context: **Software** functional **quality** reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. Software quality measurement quantifies to what extent a software or system rates along each of these five dimensions. An aggregated measure of software quality can be computed through a qualitative or a quantitative scoring scheme or a mix of both and then a weighting system reflecting the priorities.

Software quality is the degree of conformance to explicit or implicit requirements and expectations.

Explanation:

- *Explicit*: clearly defined and documented
- *Implicit*: not clearly defined and documented but indirectly suggested
- *Requirements*: business/product/software requirements
- *Expectations*: mainly end-user expectations

SOFTWARE QUALITY ASSURANCE:

Software quality assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications. SQA is an ongoing process within the software development life cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures.Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

How to Determine the Software Quality Assurance?

The 2 approaches to determine the Software Quality Assurance are:

The Defect Management Approach

The defects are categorized on the basis of the severity. The counts of the defects are taken and the actions are decided by analyzing the occurrence of defects. Defect management process is based on some principles:

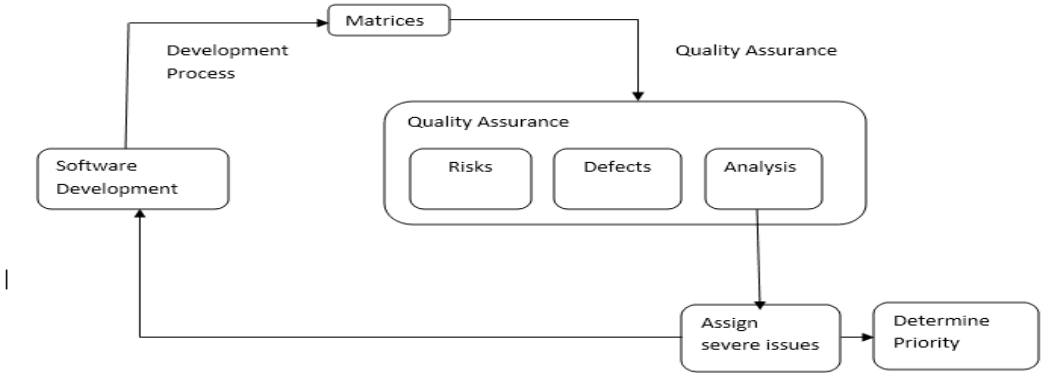
- Preventing defect is the primary goal of defect management approach. But preventing defects completely is not possible and so the purpose is to find out the defects as early as possible and to minimize the impact of the defects.

- To prevent the defects some process should be altered.

- The defect measurement processes should be integrated into the software development process, and thereby the process can be improved.

- Defect information always helps to improve the processes and hence the defect information is very useful for perfect completion of the software developed.

The diagram below explains the various stages of the defect management approach.



The Defect Management Diagram

The Software Quality Assurance Attribute Approach

There is a list of attributes which describes the step by step approach to obtain Software Quality Assurance. The attributes are given as in the diagram below:

The Quality Attribute Approach

Functionality: The attributes considers the set of all the functions used in the software.

- Suitability: Ensures the functions of the software are appropriate.
- Accuracy: Ensures the accurate usage of the functions.
- Interoperability: Ensure the effective interaction of the software with other components.
- Security: Ensure the software is capable of handling any security issues

Reliability: The purpose of the attribute is to check the capability of the system to perform without delay during any conditions

- Maturity: Less possibility of failure of the software in any activities.
- Recoverability: The rate of recovery ability once a failure occurs.

Usability: The purpose is to ensure the use of a function

- Understandability: How much effort a user needs to understand the functions.

Efficiency: The attribute depends on the architecture used and the coding practices.

Maintainability: The way to analyze and fix a fault/issue in the software

- Analyzability: Finding out the cause of failure.
- Changeability: How the system response to necessary changes.
- Stability: How stable the system is when the changes made.
- Testability: Testing efforts

Adaptability: Ability of the system to adopt the changes in its environment.

SQA Activities to Assure the Software Quality

The Software Quality Assurance of the software is analyzed and ensured by performing a series of activities. The activities are performed as step by step process and the result analysis is reported for the final evaluation process. The activities are performed as step by step process and the result analysis is reported for the final evaluation process.

A Quality Management Plan

A Quality Management Plan is designed and developed for the Software Quality Assurance Process. The plan includes the proper technical methods to manage the

Software Quality Assurance activities. The plan requires a tracking as a live plan based on the SDLC.

Applying Software Engineering Techniques

The software engineering techniques are selected to achieve software quality. The techniques to be used for Software Quality Assurance are determined by analyzing the requirements collected. The requirement evaluation can be done by using some techniques eg: Facilitated Application Specification Technique[FAST].

Also, a project estimate is prepared with the help of techniques such as Work Break Down[WBS] and Source Line of Code[SLOC] Estimation.

Technical Reviews

The Formal Technical Reviews[FTR] are conducted to assess the quality and design of the quality management plan. FTR is performed in the presence of the technical people and so will be helpful to find the defects in the early stages. FTR helps to avoid the need for reworking as the reviews in each phase are done with discussing the technical experts.

Applying the Testing Strategy

The testing strategy is designed and applied. The various levels of testing are designed and scheduled. The testing strategies are designed based on the policies of the company, the stages for each test phase execution are designed and scheduled for the concerned persons. Alpha testing and Beta testing with selected clients are also conducted to test the product before delivered.

Ensuring Process Adherence

The process adherence is the combination of 2 tasks product evaluation and process monitoring.Product evaluation is the process of ensuring all the requirements identified in the product development result to the completion of the functionalities.

The Change Control Process

The Change Control is the process which formalizes the request for changes, evaluates the quality/nature of changes, controls the impact of changes. The Change Control Mechanism is designed and implemented during the design and development stages.

SOFTWARE QUALITY ASSURANCE ACTIVITIES

The Benefits of Software Quality Assurance are:

- Monitoring and Improving the Project Management Process.

- Ensuring the Standards are followed for handling procedures.
- Preventing the severe Software Quality Assurance Issues.

The following are the Test Phases for a Test Management Process:

The Software Quality AssurancevActivities are designed and performed based on the test phases scheduled. The Software Quality Assurance diagram below explains each and every step of the Activities designed. In each SQA phase, the Software Quality Assurance team provides consultation and review of the project plan, work products, and procedures with regards to the organizational policies.

Once these activities are completed, the next step is to check for:

- Any defects/weaknesses in any activity/process
- Improve the system performance by correcting those weaknesses on a priority basis.

The software Quality Assurance [SQA] is accomplished by following some standards such as ISO 9000, CMMI or Six Sigma.

The Project Delivery Life Cycle

The project delivery life cycle incorporates QA activities and the deliverables.The Life Cycle has divided into 5 different phases and the activities and deliverables are associated with each phase.

PHASE 1: ASSESSMENT

This is the phase at which an assessment of the requirements are done and developed for realizing certain business objectives and project design. QA Deliverable: Software Quality Assurance Analyst submits a revised document on the requirement analysis and Quality Assurance Plan.

PHASE 2: PLANNING

This is the phase at which the strategic plan for the project around the information architecture is developed. The functions for various processes are also designed and double checked.

QA Activities:

- Deciding the standards and procedures.
- Develop Test Matrix: Design the test matrices.Decide the scope for testing and connect the test objectives to the requirement specifications.
- Auditing: The standards and procedures are audited and quality standards are ensured.

QA Deliverables

- Test Matrix
- Revised Documents on the testing plan.

PHASE 3: DESIGN

This is the phase in which all the necessary system components are identified based on phase 1 and 2. Then detailed design specifications are created for each component.

QA Activities:

- Auditing Procedures and Standards
- Design QA PLAN, QA Test Plan

QA Deliverables

- QA Plan for testing.
- A revision of the test matrix.

PHASE 4: DEVELOPMENT

This is the phase at which the developers constructs the project based on the design phase.

QA Activities:

- Planning Test cases: The test cases for the STLC are designed.

- Prepare the Quality Assurance Test cases.
- Prepare the test environment.

QA Deliverables

- Submits a set of Test cases.
- Set up the QA Environment

PHASE 5: IMPLEMENTATION

This is the phase in which the team concentrates on the testing and review of all aspects of the system. The team develop proper documentation for system training, market test plans etc…for making the system ready to launch

QA Activities

- Executes all test cases in the QA Test Plan

QA Deliverables

- Test Results and Reports.
- Bug Report.

SOFTWARE REVIEW

COST IMPACT OF SOFTWARE DEFECTS:

The cost of defects can be measured by the impact of the defects and when we find them. Earlier the defect is found lesser is the cost of defect. For example if error is found in the requirement specifications then it is somewhat cheap to fix it. The correction to the requirement specification can be done and then it can be re-issued. In the same way when defect or error is found in the design then the design can be corrected and it can be re-issued. But if the error is not caught in the specifications and is not found till the user acceptance then the cost to fix those errors or defects will be way too expensive.

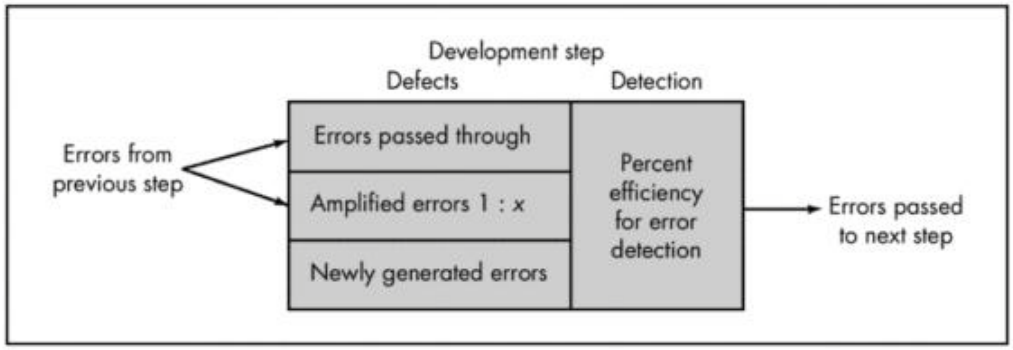
If the error is made and the consequent defect is detected in the **requirements phase** then it is relatively cheap to fix it.

Similarly if an error is made and the consequent defect is found in the **design phase** then the design can be corrected and reissued with relatively little expense.

The same applies for **construction phase**. If however, a defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix. This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in the requirements may well propagate into several places in the design and code; and because all the testing work done-to that point will need to be repeated in order to reach the confidence level in the software that we require.

Defect Amplification and Removal

- Defect amplification model:
    1. used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process.



    2. A box represents a software development step.
    3. During the step, errors may be inadvertently generated.
    4. Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through.

Formal Technical Reviews (FTR) –

1. to uncover errors in function, logic, or implementation for any representation of the software
2. to verify that the software under review meets its requirements
3. to ensure that the software has been represented according to predefined standards
4. to achieve software that is developed in a uniform manner;
5. to make projects more manageable.

The Review Meeting

Review meeting constraints:

- Between three and five people should be involved in the review.

- Advance preparation should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.

FTR focuses on a specific (and small) part of the overall software.

1 Walkthroughs are conducted for each component or small group of components.

2 FTR focuses on a work product (e.g., a portion of a requirements specification, a detailed component design, a source code listing for a component).

3 Individual who has developed the work product informs the project leader that the work product is complete and that a review is required.

4 The project leader contacts a *review leader,* who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.

5 Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.

6 Review meeting attended by
- A review leader
- B all reviewers
- C the producer.

7 One reviewer takes on the role of the *recorder;* that is, the individual who records (in writing) all important issues raised during the review.
- A The FTR begins with an introduction of the agenda and a brief introduction by the producer.
- B The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation.
- C When valid problems or errors are discovered, the recorder notes each.

8 The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings

Review Reporting and Record Keeping

1 All issues that have been raised are summarized at the end of the review meeting and a

2 ***Review summary report***

A answers three questions:
- a What was reviewed?

b Who reviewed it?
c What were the findings and conclusions?

B single page form (with possible attachments).
a Becomes part of the project historical record and may be distributed to the project leader and other interested parties.

4 ***Review issues list***
a serves two purposes:
1. identify problem areas within the product
2. serve as an action item checklist that guides the producer as corrections are made.
b attached to the summary report.

Review Guidelines

Minimum set of guidelines for formal technical reviews:

1. *Review the product, not the producer.*
2. *Set an agenda and maintain it.*
3. *Limit debate and rebuttal.*
4. *Enunciate problem areas, but don't attempt to solve every problem noted.*
5. *Take written notes.*
6. *Limit the number of participants and insist upon advance preparation.*
7. *Develop a checklist for each product that is likely to be reviewed.*
8. *Allocate resources and schedule time for FTRs.*
9. *Conduct meaningful training for all reviewers.*
10. *Review your early reviews.*

Statistical Software Quality Assurance

Statistical quality assurance implies the following steps:

1. Information about software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause (e.g., non-conformance to specifications, design error, violation of standards, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").
4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.

Software Reliability

1 Software reliability can be measured, directed and estimated using historical and developmental data.

2 *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

Measures of Reliability and Availability

Simple measure of reliability - *meantime-between-failure* (MTBF)

MTBF = MTTF + MTTR

MTTF - mean-time-to-failure MTTR - mean-time-to-repair

Software availability - the probability that a program is operating according to requirements at a given point in time and is defined as

Availability = [MTTF/(MTTF + MTTR)] **x** 100%

MTBF reliability measure is equally sensitive to MTTF and MTTR.

Availability measure is more sensitive to MTTR, an indirect measure of the maintainability of software.

Software Safety

Software safety - software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
Modeling and analysis process is conducted as part of software safety.
1. Initially, hazards are identified and categorized by criticality and risk.
2. E.g., some of the hazards associated with a computer-based cruise control for an automobile :
    a. causes uncontrolled acceleration that cannot be stopped
    b. does not respond to depression of brake pedal (by turning off)
    c. does not engage when switch is activated
    d. slowly loses or gains speed

The ISO 9000 Quality Standards

Quality assurance system - defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

1. Created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.

2. Cover wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process.

3. ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.

The ISO Approach to Quality Assurance Systems

1. ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes.
2. ISO 9000 describes the elements of a quality assurance system in general terms.
3. Elements needed to implement:

a. quality planning
b. quality control
c. quality assurance
d. quality improvement

ISO 9000 does not describe how an organization should implement quality system elements.

ISO 9001 Standard

1 ISO 9001 is the quality assurance standard that applies to software engineering.
2 Contains 20 requirements that must be present for an effective quality assurance system.

Requirements delineated by ISO 9001 address topics such as:

1 management responsibility
2 quality system
3 contract review
4 design control
5 document and data control
6 product identification and traceability,
7 process control.

The SQA Plan

SQA Plan provides a road map for instituting software quality assurance.
Serves as a template for SQA activities that are instituted for each software project.
Standard for SQA plans has been recommended by the IEEE

1 Reviews and audits section
2 Test section references the *Software Test Plan and Procedure*

Software Configuration Management

Output of the software process:
computer programs (both source level and executable forms)
documents that describe the computer programs (targeted at both technical practitioners and users)
data (contained within the program or external to it). The items that comprise all information produced as part of the software process are collectively called a *software configuration.*

Number of *software configuration items* (SCIs) grows rapidly as software process progresses.

1. Change may occur at any time, for any reason.

First Law of System Engineering:

1 Four sources of change:

1. New business or market conditions dictate changes in product requirements or business rules.
2. New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
3. Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
4. Budgetary or scheduling constraints cause a redefinition of the system or product.

2 **Software configuration management**:

1 Set of activities that have been developed to manage change throughout the life cycle of computer software.
2 Viewed as a software quality assurance activity that is applied throughout the software process.
3 Software Configuration Items

4 SCIs are organized to form *configuration objects*

1 cataloged in the project database with a single name.

The SCM Process

SCM introduces a set of complex questions:

- How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?
- How can we ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made?

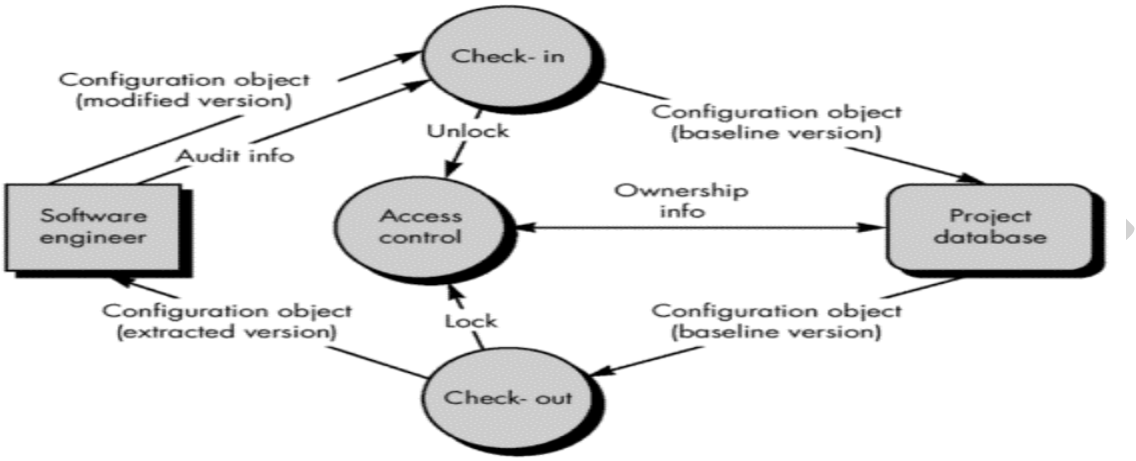
Must define five SCM tasks:

- *Identification*
- *version control*
- *change control*
- *configuration auditing*
- *reporting*

- "check-in" and "check-out" process implements two elements of change control

Access control - governs which software engineers have the authority to access and modify a particular configuration object.

Synchronization control - helps to ensure that parallel changes, performed by two different people, don't overwrite one another



CASE software

Computer-aided software engineering (CASE) is the domain of **software** tools used to design and implement applications. CASE tools are similar to and were partly inspired by **computer-aided** design (CAD) tools used for designing hardware products. CASE software is often associated with methods for the development of information systems together with automated tools that can be used in the software development process.

A. Fuggetta classified CASE software into 3 categories:

1. *Tools* support specific tasks in the software life-cycle.
2. *Workbenches* combine two or more tools focused on a specific part of the software life-cycle.
3. *Environments* combine two or more tools or workbenches and support the complete software life-cycle.

CASE Tools

CASE tools supports specific tasks in the software development life-cycle. They can be divided into the following categories:

1. Business and Analysis modeling. Graphical modeling tools. E.g., E/R modeling, object modeling, etc.

2. Development. Design and construction phases of the life-cycle. Debugging environments. E.g., GNU Debugger.
3. Verification and validation. Analyze code and specifications for correctness, performance, etc.
4. Configuration management. Control the check-in and check-out of repository objects and files. E.g., SCCS, CMS.
5. Metrics and measurement. Analyze code for complexity, modularity (e.g., no "go to's"), performance, etc.
6. Project management. Manage project plans, task assignments, scheduling.

INTEGRATED CASE ENVIRONMENT AND ARCHITECTURE:

The benefits of integrated CASE (I-CASE) include **(1)** smooth transfer of information (models, programs, documents, data) from one tool to another and one software engineering step to the next; **(2)** a reduction in the effort required to perform umbrella activities such as software configuration management, quality assurance, and document production; **(3)** an increase in project control that is achieved through better planning, monitoring, and communication; and **(4)** improved coordination among staff members who are working on a large software project.

• Provide a mechanism for sharing software engineering information among all tools contained in the environment.
• Enable a change to one item of information to be tracked to other related information items.
• Provide version control and overall configuration management for all software engineering information.
• Allow direct, nonsequential access to any tool contained in the environment.

The Role of the Repository in I-CASE

The repository for an I-CASE environment is the set of mechanisms and data structures that achieve data/tool and data/data integration. It provides the obvious functions of a database management system, but in addition, the repository performs or precipitates the following functions :

• Data integrity
• Information sharing.
• Data/tool integration
• Data/data integration
• Methodology enforcement.
• Document standardization
• Storage of sophisticated data structures.
• Integrity enforcement.
• Semantics-rich tool interface.
• Process/project management