UNIT 4 SOFTWARE QUALITY AND SECURITY

Structure

4.0 INTRODUCTION

Software quality assurance is a series of activities undertaken throughout the software engineering process. It comprises the entire gamut of software engineering life cycle. 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. The objective of security engineering is to ensure that the software developed is secure.

4. 1 OBJECTIVES

The purpose of this unit is to give an insight as to how software quality assurance activity is undertaken in the software engineering process.

After studying this unit, you should be able to understand the:

- concepts of Software quality and quality assurance;
- process of a formal technical review in a quality assurance process;
- concepts of software reliability and software quality standards, and
- concept of security engineering.

4.2 SOFTWARE QUALITY

Quality is defined as conformance to the stated and implied needs of customer. Quality also refers to the measurable characteristics of a software product and these can be compared based on a given set of standards. In the same way, software quality can be defined as conformance to explicitly stated and implicitly stated functional requirements. Here, the explicitly stated functional requirement can be derived from the requirements stated by the customer which are generally documented in some form. Implicit requirements are requirements which are not stated explicitly but are intended. Implicit functional requirements are standards which a software development company adheres to during the development process. Implicit functional requirements also include the requirements of good maintainability.

Quality software is reasonably bug-free, delivered on time and within budget, meets requirements and is maintainable. However, as discussed above, quality is a subjective term. It will depend on who the „customer" is and their overall influence in the scheme of things. Each type of „customer" will have their own slant on „quality". The end-user might define quality as some thing which is user-friendly and bug-free.

Good quality software satisfies both explicit and implicit requirements. Software quality is a complex mix of characteristics and varies from application to application and the customer who requests for it.

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, standardisation, 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.

Structuredness : 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 minimised 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.

Measurement of Software Quality (Quality metrics)

Software quality is a set of characteristics that can be measured in all phases of software development.

Defect metrics

- Number of design changes required
- Number of errors in the code
- Number of bugs during different stages of testing
- Reliability metrics
- It measures the mean time to failure (MTTF), that may be defined as probability of failure during a particular interval of time. This will be discussed in software reliability.

Maintainability metrics

- Complexity metrics are used to determine the maintainability of software. The complexity of software can be measured from its control flow.

Consider the graph of *Figure 4.1.* Each node represents one program segment and edges represent the control flow. The complexity of the software module represented by the graph can be given by simple formulae of graph theory as follows:

V(G) = e – n + 2 where

V(G) : is called Cyclomatic complexity of the program
e = number of edges
n = number of nodes

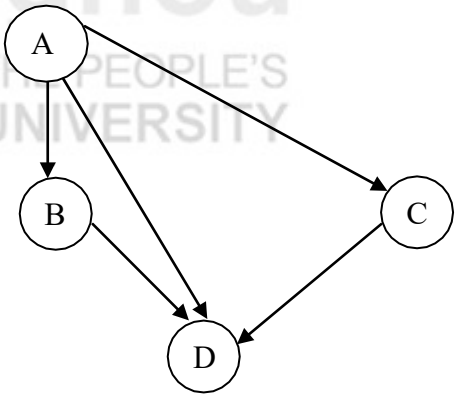

Figure 4.1: A software module

Applying the above equation the complexity V(G) of the graph is found to be 3.

The cyclomatic complexity has been related to programming effort, maintenance effort and debugging effort. Although cyclomatic complexity measures program complexity, it fails to measure the complexity of a program without multiple conditions.

The information flow within a program can provide a measure for program complexity.

Important parameters for measurement of Software Quality

- To the extent it satisfies user requirements; they form the foundation to measure software quality.
- Use of specific standards for building the software product. Standards could be organisation‟s own standards or standards referred in a contractual agreement.
- Implicit requirements which are not stated by the user but are essential for quality software.

Software Quality Assurance

The aim of the Software Quality Assurance process is to develop high-quality software product.

The purpose of Software Quality Assurance is to provide management with appropriate visibility into the process of the software project and of the products being built. Software Quality Assurance involves reviewing and auditing the software products throughout the development lifecycle to verify that they conform to explicit requirements and implicit requirements such as applicable procedures and standards. Compliance with agreed-upon standards and procedures is evaluated through process monitoring, product evaluation, and audits.

Software Quality Assurance (SQA) is a planned, coordinated and systematic actions necessary to provide adequate confidence that a software product conforms to established technical requirements.

Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

The process of Software Quality Assurance

1. Defines the requirements for software controlled system fault/failure detection, isolation, and recovery;

2. Reviews the software development processes and products for software error prevention and/ or controlled change to reduced functionality states; and

3. Defines the process for measuring and analysing defects as well as reliability and maintainability factors.

Software engineers, project managers, customers and Software Quality Assurance groups are involved in software quality assurance activity. The role of various groups in software quality assurance are as follows:

- **Software engineers**: They ensure that appropriate methods are applied to develop the software, perform testing of the software product and participate in formal technical reviews.

- **SQA group**: They assist the software engineer in developing high quality product.They plan quality assurance activities and report the results of review.

☞ **Check Your Progress 1**

1) What is auditability?
……

2) Traceability is the ability of traceacing design back to _____

3) Software quality is designed in to software and can‟t be infused after the product is released. Explain?
……

……

4.3 FORMAL TECHNICAL REVIEW

What is software Review ? Software review can‟t be defined as a filter for the software engineering process. The purpose of any review is to discover errors in analysis, design, coding, testing and implementation phase of software development cycle. The other purpose of review is to see whether procedures are applied uniformly and in a manageable manner.

Reviews are basically of two types, informal technical review and formal technical review.

- Informal Technical Review : Informal meeting and informal desk checking.
- Formal Technical Review: A formal software quality assurance activity through various approaches such as structured walkthroughs, inspection, etc.

What is Formal Technical Review: Formal technical review is a software quality assurance activity performed by software engineering practitioners to improve software product quality. The product is scrutinised for completeness, correctness, consistency, technical feasibility, efficiency, and adherence to established standards and guidelines by the client organisation.

Structured walkthrough is a review of the formal deliverables produced by the project team. Participants of this review typically include end-users and management of the client organization, management of the development organisation, and sometimes auditors, as well as members of the project team. As such, these reviews are more formal in nature with a predefined agenda, which may include presentations, overheads, etc.

An inspection is more formalised than a „Structured walkthrough‟, typically with 3-8 people. The subject of the inspection is typically a document such as a requirements specification or a test plan, and the purpose is to find problems and see what‟s missing, not to suggest rectification or fixing. The result of the inspection meeting should be a written report.

Verification **:** Verification generally involves reviews to evaluate whether correct methodologies have been followed by checking documents, plans, code, requirements, and specifications. This can be done with checklists, walkthroughs, etc.

Validation **:** Validation typically involves actual testing and takes place after verifications are completed.

Objectives of Formal Technical Review

- To uncover errors in logic or implementation
- To ensure that the software has been represented accruing to predefined standards
- To ensure that software under review meets the requirements
- To make the project more manageable.

Each Formal Technical Review (FTR) is conducted as a meeting and requires well coordinated planning and commitments.

For the success of formal technical review, the following are expected:

- The schedule of the meeting and its agenda reach the members well in advance
- Members review the material and its distribution
- The reviewer must review the material in advance.

The meeting should consist of two to five people and should be restricted to not more than 2 hours (preferably). The aim of the review is to review the product/work and the performance of the people. When the product is ready, the producer (developer) informs the project leader about the completion of the product and requests for review. The project leader contacts the review leader for the review. The review leader asks the reviewer to perform an independent review of the product/work before the scheduled FTR.

Result of FTR

- Meeting decision
  - Whether to accept the product/work without any modification
  - Accept the product/work with certain changes
  - Reject the product/work due to error

- Review summary report
  - What was reviewed?
  - Who reviewed it?
  - Findings of the review
  - Conclusion

Checklist - Typical activities for review at each phase are described below:

Software Concept and Initiation Phase

Involvement of SQA team in both writing and reviewing the project management plan in order to assure that the processes, procedures, and standards identified in the plan are appropriate, clear, specific, and auditable.

- Have design constraints been taken in to account?
- Whether best alternatives have been selected.

Software Requirements Analysis Phase

During the software requirements phase, review assures that software requirements are complete, testable, and properly expressed as functional, performance, and interface requirements. The output of a software requirement analysis phase is Software Requirements Specification (SRS). SRS forms the major input for review.

Compatibility

- Does the interface enables compatibility with external interfaces?
- Whether the specified models, algorithms, and numerical techniques are compatible?

Completeness

- Does it include all requirements relating to functionality, performance, system constraints, etc.?
- Does SRS include all user requirements?
- Are the time-critical functions defined and identified?
- Are the requirements consistent with available resources and budget?
- Does the SRS include requirements for anticipated future changes?

Consistency

- Are the requirements consistent with each other? Is the SRS free of contradictions?
- Whether SRS uses standard terminology and definitions throughout.
- Has the impact of operational environment on the software been specified in the SRS?

Correctness

- Does the SRS conform to SRS standards?
- Does the SRS define the required responses to all expected types of errors and failure? hazard analysis?
- Were the functional requirements analysed to check if all abnormal situations are covered by system functions?
- Does SRS reference development standards for preparation?
- Does the SRS identify external interfaces in terms of input and output mathematical variables?
- Has the rationale for each requirement been defined?
- Are the requirements adequate and correctly indicated?
- Is there justification for the design/implementation constraints?

Traceability

- Are the requirements traceable to top level?
- Is there traceability from the next higher level of specification?
- Is SRS traceable forward through successive software development phases like the design, coding and testing ?

Verifiability and Testability

- Are validation criteria complete?
- Is there a verification procedure defined for each requirement in the SRS?
- Are the requirements verifiable?

Software Design Phase

Reviewers should be able to determine whether or not all design features are consistent with the requirements. And, the program should meet the requirements. The output of the software design phase is a system design document (SDD) and forms an input for a Formal Technical Review.

Completeness

- Whether all the requirements have been addressed in the SDD?
- Have the software requirements been properly reflected in software architecture?
- Are algorithms adequate, accurate, and complete?
- Does the design implement the required program behavior with respect to each program interface?
- Does the design take into consideration all expected conditions?
- Does the design specify appropriate behaviour in case of an unexpected or improper input and other abnormal conditions?

Consistency

- Are standard terminology and definitions used throughout the SDD? Is the style of presentation and the level of detail consistent throughout the document.
- Does the design configuration ensure integrity of changes?
- Is there compatibility of the interfaces?
- Is the test documentation compatible with the test requirements of the SRS?
- Are the models, algorithms, and numerical techniques that are specified mathematically compatible?
- Are input and output formats consistent to the extent possible?
- Are the designs for similar or related functions are consistent?
- Are the accuracies and units of inputs, database elements, and outputs that are used together in computations or logical decisions compatible?

Correctness

- Does the SDD conform to design documentation standards?
- Does the design perform only that which is specified in the SRS?
- Whether additional functionality is justified?
- Is the test documentation current and technically accurate?
- Is the design logic sound i.e., will the program do what is intended?
- Is the design consistent with documented descriptions?
- Does the design correctly accommodate all inputs, outputs, and database elements ?

Modifiability

- The modules are organised such that changes in the requirements only require changes to a small number of modules.
- Functionality is partitioned into programs to maximize the internal cohesion of programs and to minimize program coupling.
- Is the design structured so that it comprises relatively small, hierarchically related programs or sets of programs, each performing a particular unique function?
- Does the design use a logical hierarchical control structure?

Traceability

- Is the SDD traceable to requirements in SRS?
- Does the SDD show mapping and complete coverage of all requirements and design constraints in the SRS?
- Whether the functions in the SDD which are outside the scope of SRS are defined and identified?

Verifiability

- Does the SDD describe each function using well-defined notation so that the SDD can be verified against the SRS
- Can the code be verified against the SDD?
- Are conditions, constraints identified so that test cases can be designed?

Review of Source Code

The following checklist contains the kinds of questions a reviewer may take up during source code review based on various standards

Completeness

- Is the code complete and precise in implementation?
- Is the code as per the design documented in the SDD?
- Are there any unreferenced or undefined variables, constants, or data types?
- Whether the code follows any coding standard that is referenced?

Consistency

- Is the code consistent with the SDD?
- Are the nomenclature of variables, functions uniform throughout?

Correctness

- Does the code conform to specified coding standards?
- Are all variables properly specified and used?
- Does the code avoid recursion?
- Does the code protect against detectable runtime errors?
- Are all comments accurate and sufficient?
- Is the code free of unintended infinite loops?

Modifiability

- Is the program documentation sufficient to facilitate future changes?
- Does the code refer to constants symbolically so as to facilitate change?
- Is the code written in a language with well-defined syntax and semantics?
- Are the references or data dictionaries included to show variable and constant access by the program?
- Does the code avoid relying on defaults provided by the programming language?

Traceability

- Is the code traceable to design document?
- Does the code identify each program uniquely?
- Is there a cross-reference through which the code can be easily and directly traced to the SDD?
- Does the code contain, or has reference to, a revision history of all code modifications done by different authors?
- Whether the reasons for such modification are specified and authorized?

Understandability

- Do the comment statements in the code adequately describe each routine?
- Whether proper indent and formatting has been used to enhance clarity?
- Has the formatting been used consistently?
- Does naming patterns of the variables properly reflect the type of variable?
- Is the valid range of each variable defined?

Software Testing and Implementation Phase

- Whether all deliverable items are tested.

- Whether test plans are consistent and sufficient to test the functionality of the systems.
- Whether non-conformance reporting and corrective action has been initiated?.
- Whether boundary value is being tested.
- Whether all tests are run according to test plans and procedures and any non-conformances are reported and resolved?
- Are the test reports complete and correct?
- Has it been certified that testing is complete and software including documentation are ready for delivery?

Installation phase

Completeness

- Are the components necessary for installation of a program in this installation medium ?
- Whether adequate information for installing the program, including the system requirements for running the program available.
- Is there more than one operating environment?
- Are installation procedures for different running environments available?
- Are the installation procedures clearly understandable?

☞ Check Your Progress 2

1) The purpose of review is to uncover errors and non conformity during testing phase. Yes ☐ No ☐

2) The result of a review does not include
   a) The items that are to be reviewed
   b) The person who reviews it
   c) Findings of the review
   d) Solution to a problem

3) What could become an input for FTR in design phase?
   ……
   ……

4.4 SOFTWARE RELIABILITY

Software Reliability: Unlike reliability of the hardware device, the term software reliability is difficult to measure. In the software engineering environment, software reliability is defined as the probability that software will provide failure-free operation in a fixed environment for a fixed interval of time.

Software reliability is typically measured per a unit of time, whereas probability of failure is generally time independent. These two measures can be easily related if you know the frequency with which inputs are executed per unit of time. Mean-time-to-failure (MTTF) is the average interval of time between failures; this is also sometimes referred to as Mean-time-before-failure.

Possibly the greatest problem in the field of software reliability estimation has to do with the accuracy of operational profiles. Without accurate profiles, estimates will almost certainly be wrong. An operational profile is the probability density function (over the entire input space) that best represents how the inputs would be selected during the life-time of the software. There is nothing fancy about operational profiles; they are really just "guesses" about what inputs will occur in the future.

Definitions of Software reliability

- IEEE Definition: The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system in the software. The inputs to the system determine whether existing faults, if any, are encountered.

- The ability of a program to perform its required functions accurately and reproducibly under stated conditions for a specified period of time.

Software Reliability Models

A number of models have been developed to determine software defects/failures. All these models describe the occurrence of defects/failures as a function of time. This allows us to define reliability. These models are based on certain assumptions which can be described as below:

- The failures are independent of each other, i.e., one failure has no impact on other failure(s).
- The inputs are random samples.
- Failure intervals are independent and all software failures are observed.
- Time between failures is exponentially distributed.

The following formula gives the cumulative number of defects observed at a time „t".

$$D(t) = T_d (1 - e^{-bct})$$

D(t) = Cumulative number of defects observed at a time t

T_d = Total number of defects

„b" and „c" are constants and depend on historical data of similar software for which the model is being applied

We may find the mean time to failure (MMFT) as below:

$$MTTF(t) = e^{bct} / c\, T_d$$

4.5 SOFTWARE QUALITY STANDARDS

Software standards help an organization to adopt a uniform approach in designing, developing and maintaining software. There are a number of standards for software quality and software quality assurance. Once an organisation decides to establish a software quality assurance process, standards may be followed to establish and operate different software development activities and support activities. A number of organisations have developed standards on quality in general and software quality in specific.

Software Engineering Institute (SEI) has developed what is called a „*Capability Maturity Model'* (CMM) now called the *Capability Maturity Model Integration* (CMMI) to help organisations to improve software development processes.

It is a model of 5 levels of process maturity that determine the effectiveness of an organisation in delivering quality software. Organizations can receive CMMI ratings

by undergoing assessments by qualified auditors. The organizations are rated as CMM Level 1, CMM Level 2 etc. by evaluating their organisational process maturity.

SEI-CMM Level 1: Characterised by un organised, chaos, periodic panics, and heroic efforts required by individuals to successfully complete projects. Successes depend on individuals and may not be repeatable.

SEI-CMM Level 2 : Software project tracking, requirements management, realistic planning, and configuration management processes are in place; successful practices can be repeated.

SEI-CMM Level 3: Standard software development and maintenance processes are integrated throughout an organisation; a Software Engineering Process Group is in place to oversee software processes, and training programs are used to ensure understanding and compliance.

SEI-CMM Level 4: Metrics are used to track productivity, processes, and products. Project performance is predictable, and quality is consistently high.

SEI-CMM Level 5: The focus is on continuous process improvement. The impact of new processes and technologies can be predicted and effectively implemented when required.

The International Organisation for Standardisation (ISO) developed the ISO 9001:2000 standard (which replaces the previous set of three standards of 1994) that helps the organisation to establish, operate, maintain and review a quality management system that is assessed by outside auditors. The standard is generic in nature and can be applied to any organisation involved in production, manufacturing service including an organisation providing software services.

It covers documentation, design, development, production, testing, installation, servicing, and other processes. It may be noted that ISO certification does not necessarily indicate quality products. It only indicates that the organisation follows a well documented established process. *Table 4.1* gives a list of standards along with the corresponding titles.

Table 4.1 : List of standards

Standards	Title
ISO/IEC 90003	Software Engineering. Guidelines for the Application of ISO 9001:2000 to Computer Software
ISO 9001:2000	Quality Management Systems - Requirements
ISO/IEC 14598-1	Information Technology-Evaluation of Software Products-General Guide
ISO/IEC 9126-1	Software Engineering - Product Quality - Part 1: Quality Model
ISO/IEC 12119	Information Technology-Software Packages-Quality Requirements and Testing
ISO/IEC 12207	Information Technology-Software Life Cycle Processes
ISO/IEC 14102	Guideline For the Evaluation and Selection of CASE Tools
IEC 60880	Software for Computers in the Safety Systems of Nuclear Power Stations
IEEE 730	Software Quality Assurance Plans
IEEE 730.1	Guide for Software Assurance Planning
IEEE 982.1	Standard Dictionary of Measures to produce reliable software
IEEE 1061	Standard for a Software Quality Metrics Methodology
IEEE 1540	Software Life Cycle Processes - Risk Management
IEEE 1028	Software review and audits
IEEE 1012	Software verification and validation plans

Check Your Progress 3

1) Success of a project depends on individual effort. At what stage of maturity doesthe organisation‟s software process can be rated?
……..
……...

2) Why ISO 9001 : 2000 is called generic standard?
……..
……...

3) What is the difference between SEI CMM standards and ISO 9000 : 2000 standards?
……..
……...

4.6 SECURITY ENGINEERING

Whenever you need to develop software, one aspect on which you need to focus is “security”. One important question is :To what extent is the software that is developed by you can thwart attacks. So, its essential to analyze software security requirements. Security is a very important aspect on which you need to focus if you are on social media. Security is equally important when working with Mobile applications, IOT(Internet of Things) applications as well as applications in the cloud. As part of security requirements analysis, there is need to elicit ate security requirements, develop a security policy, design security measures and perform security checks during the entire lifecycle of software development. As part of project planning, its important to identify and analyze security risks. Identifying the assets, creating an architecture overview, decomposing an application, identifying the threats, documenting the threats and rating the threats are some of the steps to create a threat model.

4.7 SUMMARY

Software quality assurance is applied at every stages of system development through a process of review to ensure that proper methodology and procedure has been followed to produce the software product. It covers the entire gamut of software development activity. Software quality is conformance with explicitly defined requirements which may come from a customer requirement or organization may define their own standards. Software review also called a Formal Technical Review is one of the most important activity of software quality assurance process. Software reliability provides measurement of software dependability and probability of failure is generally time independent. The aim of software quality assurance is to improve both software product and process through which the software is built. The help of various software standards taken to establish and operate software quality assurance process. One very important aspect that needs to be taken care of whenever Software is to be developed is “Security”. Any software that is developed needs to be secure. It should be ensured that it takes care of all the threats that arise.

4.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Auditability is that attribute of software which determines the degree to which a software can be tested for conformance against a standard.

2) Requirements.

3) Software quality assurance activity is an umbrella activity comprising activities like application of standards, technical methods, review, software testing, changecontrol, measurement, control & reporting during the process of software development life cycle. High quality product can come from high quality design specification. Unlike quality, testing quality assurance can‟t be achieved at the end of the product completion phase.

Check Your Progress 2

1) No. The purpose of any review is to discover errors in the analysis, design, coding, testing and maintenance phases of software development life cycle.

2) Solution to a problem.

3) Output of the software design phase is a system design document (SDD) and it is an input for the Formal Technical Review.

Check Your Progress 3

1) SEI CMM level 1.
2) The standards are called generic in nature as they can be applied to any organisation involved in the production, manufacturing service including organisations providing software services which involve no manufacturing process.
3) SEI CMM standards are developed to rate the maturity of organisation‟s software development process. The maturity is rated from level 1 to level 5, i.e., from immature (no formal process) to predictable matured process with focus on continuous improvement. In case of ISO 9001 : 2000, the organisation‟s process (quality system) is tested against conformance to the requirements of ISO 9000 : 2000 standard.

4.9 FURTHER READINGS

1) *Software Quality,* Mordechai Ben-Menachem and Garry S.Marliss; Thomson Learning.

2) *Software Engineering,* Ian Sommerville; PearsonEducation.

3) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill Education.

4) *Software Security Technologies:A Programmatic Approach*, Richard Sinn; Cengage Learning.

Reference websites

http://www.rspa.com
http://www.ieee.org
https://en.wikipedia.org/wiki/Security_engineering