Software Quality Assurance

Unit 3

Red: Indicates important

Contents

Software Quality

- Software quality is the degree of conformance to explicit or implicit requirements and expectations.
- Quality software is:
  - Bug-free
  - Delivered on time
  - Within budget
  - Meets requirements
  - Maintainable

Attributes of quality

- Testability: Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met.
- Understandability: The degree to which the SRS,SDD, and code are clear and understandable to the reader.
- Verifiability: The degree to which SRS, SDD, and code have been written to facilitate verification and testing.
- Traceability: The degree to which 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 reason for its existing.
- Structuredness: The degree to which the SDD and code has a definite pattern in their interdependent parts. The design must have been done in a systematic and orderly manner(top-down or bottom-up). The modules are more cohesive and has minimized coupling.

- Robustness: The degree to which a system or component can function correctly in the presence of invalid inputs and stressful environment.
- Predictability: The degree to which the functionality and performance of the software are determinable for a specified set of inputs.
- Modularity: The degree to which a system or computer program is composed of discrete components in such a way that a change to one component has minimal impact on other components.
- Feasibility: The degree to which the requirements, design, or plans for a system or component can be implemented under existing constraints.
- Correctness: The degree to which a system or component is free from fault in the specification, design and implementation. The degree to which the software, documentation and other items meet its requirements.
- Consistency: The degree of uniformity, standardization and freedom from contradiction among the components and documents.

- Completeness: The degree to which all the software's required functions and design constraints are present and fully developed in the requirements specifications, design document and code.
- Compatibility: The ability of two or more systems or components to perform their functions while sharing the same hardware or software environment.
- Auditability: The ability of software being tested against conforms to standard.
- Causes of error :
  - Misinterpretation of customer's requirements/communication.
  - Incomplete/erroneous system specification
  - Error in logic
  - Not following program/software standards
  - Incomplete testing
  - No documentation
  - Deviation from specification
  - Error in data modeling and representation

Measurement of software quality

- 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) .
  - MTTF is the probability of failure during a particular interval of time.
- Maintainability metrics
  - Complexity metrics are used to determine the maintainability of software.
  - Complexity of software can be measured from its control flow.
  - Each node represents one program segment and edges represent control flow.
  - Complexity can be determined using the formula:

    $V(G)=e-n+2$ where

- V(G) is cyclomatic complexity

  E: number of edges

  N:number of nodes

- Important parameters for measurement of software quality:
  - To the extent it satisfies user requirement
  - Use of specific standards for developing the software product.
  - Implicit users which are not stated by user

Software Quality Assurance(SQA)

- Aim of SQA process is to develop a product of high quality .
- Purpose of SQA is to provide management with appropriate visibility of the product built and into the process of software project.
- It involves auditing and reviewing the software products to verify that they comply to the implicit and explicit requirements.
- Its monitoring the processes and products throughout the SDLC.
- Process monitoring, product evaluation, and audits are done to evaluate the compliance with standards and procedures.
- SQA is a systematic, planned and coordinated actions required to provide confidence that a software product complies to the requirements.

- The process of SQA:
  1. Defines the requirement 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.
  3. Define the process for measuring and analyzing defects as well as reliability and maintainability factors.
- Software engineers, project managers, customers an Software Quality Assurance groups are involved in SQA activities.
- Software engineers: ensure that appropriate methods are applied to develop the software, perform testing of the software product and participate in FTR.
- SQA group: They assist the software engineer in developing a high quality product. SQA plans quality assurance activities and report the results of review.

Software Review

- Purpose of any review is to discover errors in analysis, design, coding, testing and implementation phase of software development cycle. Other purpose is to see whether the procedures are applied uniformly and in a manageable manner.
- Reviews are of two types:
  1. Formal technical review
  2. Informal 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..

Formal Technical Review(FTR)

- It’s a software quality assurance activity performed by software engineering practitioners to improve software product quality.
- The product is scrutinized for completeness, correctness, consistency, technical feasibility, efficiency, and adherence to established standards and guidelines by the client organization.
- Each FTR is conducted as a meeting and requires well coordinated planning and commitments.
- FTR techniques:
  - Structured walkthrough
  - Inspection
  - Verification
  - Validation

- Structured walkthrough :
  - It’s 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 organization, and sometimes auditors, as well as members of the project team.
  - Its formal in nature with a predefined agenda, which includes presentations, overheads, etc..
- Inspection :
  - Its more formalized than a ‘Structured walkthrough’ typically with 3-5 people.
  - 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.
  - Result is a written report.

- Verification :
  - It 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 :
  - Typically involves actual testing and takes place after verifications are completed.

Objectives of FTR

- To uncover errors in logic or implementation
- To ensure that the software has been represented accruing to predefined standards
- To ensure that the software under review meets the requirements.
- To make the project more manageable.
- For the success of FTR, 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. The aim of the review is to review the product/work and the performance of the people. When the product is ready, the producer informs the project leader about the completion of the product and request 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

Typical activities for review at each phase

- 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.
  - In this phase review is performed to do following checks:
    - Have design constraints been taken into account?
    - Whether best alternatives are selected.
- Software Requirements Analysis Phase :
  - In this 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 SRS. It 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?
    - Are the requirements adequate and correctly indicated?
    - Does the SRS identify external interfaces in terms of input and output mathematical variables?

- ❑ 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.
  - Program should meet the requirements.
  - Output of software design phase is SDD. It forms an input for a FTR.

- ❑ Completeness
  - Whether all the requirements have been addressed in the SDD?
  - Are algorithms adequate, accurate, and complete?
  - Does the design take into consideration all expected conditions?
  - Does the design specify appropriate behavior in case of an unexpected or improper input and other abnormal conditions?
- ❑ Consistency
  - Are standard terms and definitions used throughout the SDD? Is the style and level of detail consistent throughout.
  - Is there compatibility of the interface?
  - Are input and output format consistent?
  - Are the designs for similar or related functions consistent?
  - Is the test documentation compatible with the test requirements of the SRS?
  - Are the models, algorithms, and numerical techniques mathematically 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 specified?
  - Is the test documentation current and accurate?
  - Will the program do what is intended?
- ❑ Modifiability
  - The modules are organized such that changes in the requirements only require changes to small number of modules.
  - Does the design use a logical hierarchical control structure?
  - Functionality is divided into programs to maximize cohesion and minimize coupling.
- ❑ Traceability
  - Is the SDD traceable to requirements in SRS?
  - Does the SDD show mapping and complete details of requirements and design constraints in the SRS?
  - Whether the functions in the SDD that are outside the scope of SRS are defined and identified?

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

- Review of Source code :
  - 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?
  - Consistency
    - Is the code consistent with SDD?
    - Are the nomenclature of variables, functions same throughout?

- Correctness
  - Does the code conform to specified coding standard?
  - Are all variables properly specified and used?
  - Does the code avoid recursion?
  - Are all comments accurate and sufficient?
  - Is the code free of 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?
- Traceability
  - Is the code traceable to design document
  - Does the code identify each program uniquely?
  - Does the code contain a revision history of all code modifications done by different authors?
  - Whether the reasons for such modification are specified and authorized?

- ❑ Understandability
  - Do the comments describe each routine?
  - Whether proper indent and formatting used to enhance clarity?
  - Has the formatting been used consistently?
  - Does naming patters 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 system?
  - Whether boundary value is being tested.
  - Are the test reports complete and correct?
  - Whether all tests are run according to test plans and procedures and any non-conformances are reported and resolved?
  - 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?
    - Is there more than one reporting environment?
    - Are installation procedures for different running environments available?
    - Are the installation procedures clearly understandable?
    - Whether adequate information for installation program including software requirements is available.

Software Reliability

- It's 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.

- It's 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.
  - It describes the occurrence of defects/failures as a function of time.
  - Assumptions for software reliability models :
    - The failures are independent of each other, ie, one failure has no impact on other failures.
    - The inputs are random samples.
    - Failure intervals are independent and all software failures are observed.
    - Time between failures is exponentially distributed.

- The formula below gives a 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.
- Mean time to failure (MTTF) :

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

MTTF : is the average interval of time between failures; aka Mean-time-before-failure.

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.
- A number of organizations have developed standards on quality in general and software quality in specific.
- Software Engineering Institute has developed what is called a 'Capability Maturity Model '(CMM)now known as 'Capability Maturity Model Integration'(CMMI) to help organizations to improve software development processes.

CMMI

- It’s a model of 5 levels of process maturity that determines the effectiveness of an organization in delivering quality software. Organizations can receive CCI ratings by undergoing assessments by qualified auditors. It can be rated as CMM Level 1, CMM Level 2, CMM Level 3, etc….

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

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

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

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

5. 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.

- ISO 9001 :2000 :

- The International Organization for Standardisation (ISO) developed this standard that helps the organization 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 organization involved I production, manufacturing service including an organization providing software services.

- It covers documentation, design, development, production, testing, installation, servicing, and other processes.

- ISO certification does not necessarily indicate quality products. It only indicates that the organization follows a well documented established process.
- List of ISO standards : (for more block 2, page 42, table 3.1)

Standards	Title
ISO 9001: 2000	Quality Management Systems-Requirements
IEEE 1028	Software review and audits
IEEE 1012	Software verification and validation plans
IEEE 730	Software Quality Assurance Plan