Software Quality Assurance and Testing

Software Quality

Definition of Software Quality

Software quality is defined as an effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

Types of Software Quality

- **Quality of Design:** Encompasses requirements, specifications, and the overall design of the system.
- **Quality of Conformance:** Focuses primarily on the implementation of the software.

User Satisfaction

User satisfaction is achieved when the product is compliant, of good quality, and delivered within budget and schedule.

Effective Software Process

An effective software process provides the infrastructure for building high-quality software through:

- Management aspects that prevent chaos.
- Software engineering practices for problem analysis and design.
- Umbrella activities like change management and technical reviews.

Useful Product

A useful product delivers desired content, functions, and features reliably and error-free. It satisfies both explicit (stated) and implicit (expected, e.g., ease of use) requirements.

Adding Value

High-quality software adds value by:

- **For the Software Organization:** Reducing maintenance effort, bug fixes, and customer support costs.
- **For the User Community:** Providing useful capabilities that expedite business processes.
- **Overall Result:** Increased revenue, better profitability, and improved information availability.

Quality Dimensions (Garvin)

- **Performance Quality:** Delivers specified content, functions, and features effectively.
- **Feature Quality:** Provides features that delight users.
- **Reliability:** Delivers functionality without failure and is available when needed.
- **Conformance:** Adheres to relevant software standards and de facto design/coding conventions.
- **Durability:** Allows for maintenance and correction without unintended side effects.
- **Serviceability:** Can be maintained or corrected in an acceptable timeframe with adequate support information.
- **Aesthetics:** Possesses an elegance and unique flow.

- **Product Operations:**
    - Correctness: Does it do what is intended?
    - Reliability: Does it do it accurately all the time?
    - Efficiency: Does it perform optimally on the hardware?
    - Integrity: Is it secure?
    - Usability: Is it designed for the user?
- **Product Revision:**
    - Maintainability: Can it be fixed?
    - Flexibility: Can it be changed?
    - Testability: Can it be tested?
- **Product Transition:**
    - Portability: Can it be used on different machines?
    - Reusability: Can parts of the software be reused?
    - Interoperability: Can it interface with other systems?

Measuring Quality

- General quality dimensions are insufficient; targeted questions are needed.
- Subjective measures can be mere opinions.
- Software metrics are indirect measures used to quantify quality assessment.

The Software Quality Dilemma

There's a trade-off between producing high-quality software (which can be expensive and time-consuming) and meeting market demands. The goal is often a "good enough" balance.

"Good Enough" Software

Delivers desired functions with high quality but may have known bugs in less critical areas. This approach can be risky for small companies or critical application domains due to potential reputational damage or litigation.

The Cost of Quality

- **Prevention Costs:** Quality planning, formal reviews, training.
- **Internal Failure Costs:** Rework, repair, failure mode analysis.
- **External Failure Costs:** Complaint resolution, product returns, warranty work, help line support.

The cost to find and repair errors increases dramatically as development progresses (Requirements < Design < Coding < Testing < Maintenance).

Software Quality Assurance (SQA)

Achieving Software Quality

- Quality is a result of good project management and solid engineering practice.
- Understanding the problem and creating a conforming design are crucial.

Key Components of SQA

- **Project Management:** Includes explicit quality and change management in the project plan.
- **Quality Control (QC):** Inspections, reviews, and tests to ensure work product conformance to specifications.
- **Quality Assurance (QA):** Auditing and reporting procedures to provide data for proactive management decisions.

- Change management
- Education
- Vendor management
- Security management
- Safety
- Risk management

Role of the SQA Group

- Prepares an SQA plan (evaluations, audits, standards, error reporting, documentation, feedback).
- Participates in developing the project's software process description, ensuring compliance.
- Reviews software engineering activities for process compliance.
- Identifies, documents, and tracks deviations and ensures corrections.
- Audits work products for compliance.
- Reports periodically to the project manager.
- Ensures noncompliance is documented and handled appropriately.
- Tracks noncompliance items until resolution.

SQA Goals

- **Requirements Quality:** Ensure correctness, completeness, and consistency.
- **Design Quality:** Assess all design elements for quality and conformance to requirements.
- **Code Quality:** Ensure source code conforms to standards and facilitates maintainability.
- **Quality Control Effectiveness:** Optimize resource allocation for maximum quality impact.

Statistical SQA

1. Collect and categorize information about software errors and defects.
2. Trace errors/defects to their underlying causes.
3. Identify the "vital few" causes (e.g., using the Pareto principle).
4. Correct the problems causing these defects.

Generic Causes of Problems

- Incomplete or Erroneous Specifications (IES)
- Misinterpretation of Customer Communication (MCC)
- Intentional Deviation from Specifications (IDS)
- Violation of Programming Standards (VPS)
- Error in Data Representation (EDR)
- Inconsistent Component Interface (ICI)
- Error in Design Logic (EDL)
- Incomplete or Erroneous Testing (IEL)
- Inaccurate or Incomplete Documentation (IID)
- Error in Programming Language Translation of Design (PLT)
- Ambiguous or Inconsistent Human / Computer Interface (HCI)
- Miscellaneous (MIS)

Software Reliability

Defined as the probability of failure-free operation in a specified environment for a specified time.

- **Mean-Time-Between-Failure (MTBF):** $MTBF = MTTF + MTTR$

Design features can be incorporated to eliminate or control hazards early on.

Software Quality Standards

Key Standards Organizations

- IEEE (Institute of Electrical and Electronics Engineers) Computer Society
- ISO (International Organization for Standardization)
- DOD (US Department of Defense)
- ANSI (American National Standards Institute)
- IEC (International Electro Technical Commission)
- EIA (Electronic Industries Association)

ISO 9001 Standards

- Specifies requirements for a Quality Management System (QMS).
- Demonstrates the ability to consistently provide products and services meeting customer and regulatory requirements.
- ISO 9001:2015 is relevant for software engineering.
- Key requirements include: Leadership, Quality Policy, Planning, Support, Customer Communication, Product Identification, Customer Satisfaction, Management Review, Internal Audit, Continual Improvement, Nonconformity and Corrective Action.

Capability Maturity Model (CMM)

- Developed by the Software Engineering Institute (SEI).
- Focuses on improving the software engineering process.
- Organizations are assessed and certified on Levels 1 to 5.

Capability Maturity Model Integration (CMMI)

- Successor to CMM.
- Supports Agile Software Development and high maturity practices.
- Offers staged and continuous representations.

Process Maturity Model Levels

- Level 1: Initial
- Level 2: Managed
- Level 3: Defined
- Level 4: Quantitatively Managed
- Level 5: Optimizing

Six-Sigma for Software Engineering

- Aims for extremely high quality (3.4 defects per million occurrences).
- Methodology:
  - **Define:** Customer requirements and project goals.
  - **Measure:** Existing process performance and defect metrics.
  - **Analyze:** Identify root causes of defects.
  - **Improve:** Eliminate root causes.
  - **Control:** Ensure defects are not reintroduced.

Software Testing Strategies

Strategic Approach to Testing

- Testing and debugging are distinct activities.

Verification vs. Validation

- **Verification:** Ensures the software correctly implements its specifications ("Are we building the product right?").
- **Validation:** Ensures the software meets customer requirements ("Are we building the right product?").

Who Tests the Software?

- **Developer:** Understands the system but may test "gently" and is driven by delivery.
- **Independent Tester:** Learns the system, attempts to break it, and is driven by quality.

Testing Hierarchy

System Engineering -> Analysis Modeling -> Design Modeling -> Code Generation -> Unit Test -> Integration Test -> System Test -> Validation Test

Overall Testing Strategies

- **Testing in the Small:** Focuses on individual components or classes.
- **Testing in the Large:** Focuses on integrating components/classes into a complete system.
- For OO software, the "unit" is often a class.

Strategic Issues in Testing

- Quantifiable product requirements.
- Explicit testing objectives.
- Understanding user profiles.
- Rapid cycle testing.
- Robust, self-testing software design.
- Effective technical reviews.
- Reviewing the test strategy and cases.
- Continuous improvement of the testing process.

Unit Testing

- Focuses on individual modules or components.
- Performed by the software engineer.
- Tests include: interface, local data structures, boundary conditions, independent paths, and error handling paths.
- Requires a test environment with stubs (for called modules) and drivers (to call the module).

Integration Testing

- Combines tested units into larger subsystems or the complete system.
- **Strategies:**
  - **Big Bang:** Integrate all at once (risky).
  - **Incremental Construction:** Integrate modules step-by-step.
    - **Top-Down Integration:** Start with the top module and integrate downwards, using stubs.
    - **Bottom-Up Integration:** Start with the lowest-level modules and integrate upwards, using drivers.
    - **Sandwich Testing:** Combines top-down and bottom-up approaches.

Regression Testing

functionality works.
- Aims to uncover "show stopper" errors early.

General Testing Criteria

- **Interface Integrity:** Test interfaces between modules.
- **Functional Validity:** Uncover functional defects.
- **Information Content:** Test for errors in data structures.
- **Performance:** Verify performance bounds.

System Testing

- Focuses on user-visible actions and outputs.
- **Validation Testing:** Demonstrates conformity with requirements.
- **Alpha Testing:** Performed at the developer's site by representative end-users.
- **Beta Testing:** Performed at end-user sites in a natural environment.
- **Acceptance Testing:** Formal verification by the customer before accepting the software.

High Order Testing

- **Recovery Testing:** Verifies system recovery from failures.
- **Security Testing:** Verifies protection mechanisms against unauthorized access.
- **Stress Testing:** Pushes the system to its resource limits.
- **Performance Testing:** Assesses run-time performance within the integrated system.

Testing Object-Oriented Applications

- Testing strategy shifts focus from modules to classes.
- **Class Testing:** Equivalent to unit testing for classes, testing operations and state behavior.
- **Integration Strategies:**
  - **Thread-based Testing:** Integrates classes needed for a specific input/event response.
  - **Use-based Testing:** Integrates classes needed for a use case.
  - **Cluster Testing:** Integrates classes demonstrating a specific collaboration.

Testing Web Applications

Quality Dimensions for Web Apps

- **Content:** Syntactic and semantic correctness, consistency, lack of ambiguity.
- **Function:** Correctness, stability, conformance to standards.
- **Structure:** Proper content/function delivery, extensibility, supportability.
- **Usability:** Ease of learning and use for all user categories.
- **Navigability:** Exercising all navigation paths for errors.
- **Performance:** Responsiveness under various operating conditions and loads.
- **Compatibility:** Works across different client and server configurations.
- **Interoperability:** Proper interfacing with other applications/databases.
- **Security:** Assessment of vulnerabilities and penetration attempts.

WebApp Testing Process

Includes testing of content, interface, navigation, components, configuration, performance, and security.

Testing Mobile Apps

- **Testing-in-the-Wild:** Real-world user environments.
- **Certification Testing:** Meeting distribution standards.

Software Testing Techniques

Software Testing

- **Definition:** Exercising a program with the intent of finding errors before delivery.
- **Goal:** Design test cases with a high probability of finding errors.
- **What Testing Shows:** Errors, requirements conformance, performance indications, and an indication of quality.

Testability Attributes

- Operability
- Observability
- Controllability
- Decomposability
- Simplicity
- Stability
- Understandability

What is a "Good" Test?

- High probability of finding an error.
- Not redundant.
- "Best of breed."
- Neither too simple nor too complex.

Internal and External Views of Testing

- **External (Black-Box):** Test based on specified function and requirements.
- **Internal (White-Box):** Test based on internal workings and component exercise.

Test Case Design Principles

- **Objective:** Uncover errors.
- **Criteria:** Complete manner.
- **Constraint:** Minimum effort and time.
- **"Bugs lurk in corners and congregate at boundaries..."**

Exhaustive Testing vs. Selective Testing

- **Exhaustive Testing:** Testing every possible path (often infeasible due to complexity).
- **Selective Testing:** Choosing paths and conditions to test based on risk and likelihood of errors.

Software Testing Methods

- **White-Box Methods:** Focus on internal code structure.
- **Black-Box Methods:** Focus on external behavior and requirements.

White-Box Testing

- **Goal:** Ensure all statements and conditions have been executed.
- **Rationale:** Logic errors and assumptions are often inversely proportional to path execution probability; untested paths may contain errors.

Basic Path Testing

    - $V(G) = P + 1$ (where P = number of predicate nodes)
    - Higher $V(G)$ indicates higher error probability.
2. **Derive Independent Paths:** The number of independent paths equals $V(G)$.
3. **Derive Test Cases:** Create test cases to execute each independent path.

Control Structure Testing

- **Condition Testing:** Exercises logical conditions within a module.
- **Data Flow Testing:** Selects test paths based on variable definitions and uses.

Loop Testing

Tests focus on loop execution:

- **Simple Loops:** Skip, one pass, two passes, m passes $(m < n)$, $n-1, n, n+1$ passes (n is max allowable).
- **Nested Loops:** Test innermost loop first, then outer loops, holding others at minimum or typical values.
- **Concatenated Loops:** Treat independently if possible; otherwise, treat as nested.

Black-Box Testing

- Focuses on inputs, outputs, events, and requirements without knowledge of internal code.
- **Questions Addressed:** How is functional validity tested? What inputs are good test cases? Where are boundaries? What is the system's tolerance for data rates/volume?
- **Error Categories:** Incorrect/missing functions, interface errors, data structure errors, behavior/performance errors, initialization/termination errors.

Black-Box Testing Methods

- **Graph-Based Testing:** Based on modeling objects and their relationships.
- **Equivalence Partitioning:** Divides input domain into classes; derive test cases from each class.
- **Boundary Value Analysis (BVA):** Selects test cases at the boundaries of input/output domains.

Testing Object-Oriented Applications (Specifics)

- Testing extends to analysis and design models.
- **'Testing' OO Models:** Review semantic correctness and consistency of class definitions, attributes, operations, and relationships.
- **Class Testing:** Focuses on individual classes, their operations, and state behavior.
- **OO Testing Methods:**
    - **Fault-based Testing:** Designs tests to exercise plausible faults.
    - **Class Hierarchy Testing:** Inherited classes still require thorough testing.
    - **Scenario-Based Test Design:** Tests based on user tasks (use cases).
    - **Random Testing:** Generates random valid test sequences.
    - **Partition Testing:**
        - **State-based:** Categorizes operations by state changes.
        - **Attribute-based:** Categorizes operations by attributes used.
        - **Category-based:** Categorizes operations by generic function.
    - **Inter-Class Testing:** Tests interactions between client and server classes.
    - **Behavior Testing:** Ensures all allowable states of an object are covered.

Debugging

- **Definition:** A diagnostic process to find and correct errors.

- **Debugging Techniques:** Brute force, backtracking, induction, deduction.
- **Correcting the Error:** Consider if the bug might be elsewhere, potential side effects of the fix, and how to prevent similar bugs in the future.

Software Project Management

Project Definition

A group of tasks performed within a definable time period to meet specific objectives. Projects are typically unique, have a defined lifecycle, scope, budget, and require resources.

Project Management Definition

The planning, organizing, staffing, directing, and controlling of a company's resources to meet its objectives. For software projects, it's planning, organizing, directing, and controlling resources for a specific time to meet one-time objectives.

Primary Objectives of Project Management

- Meet specified performance.
- Within cost.
- On schedule.

Project Management Activities

Establishing objectives, defining requirements, determining timing, establishing resources, creating a cost baseline, optimizing the plan, tracking actuals, comparing progress to baseline, evaluating performance, forecasting, and recommending corrective actions.

Benefits of Project Management

Clear responsibilities, reduced reporting needs, defined timelines, trade-off analysis methodology, accomplishment measurement, early problem identification, improved estimating, and knowing when objectives cannot be met.

Software Project Influences

Size, delivery deadline, budgets, application domain, technology, system constraints, user requirements, and available resources.

Project Management Concerns

Product quality, risk assessment, measurement, cost estimation, scheduling, customer communication, staffing, resource management, and project monitoring.

The Four P's

- **People:** The most critical element.
- **Product:** The software to be built.
- **Process:** Framework activities and tasks.
- **Project:** All work required to make the product a reality.

Common Project Management Problems

Inadequate resources, unrealistic deadlines, unclear goals, uncommitted team members, insufficient planning, communication breakdowns, changing goals/resources, and inter-departmental conflicts.

Project Management Skills

Communication, organization, team building, leadership, coping, and technological skills.

People in a Project

- Customers
- End-users

Software Teams

Consider collaboration, leadership, organization, motivation, and idea generation.

Team Leader (MOI Model)

- **Motivation:** Encouraging technical people to perform their best.
- **Organization:** Molding processes to translate concepts into products.
- **Ideas/Innovation:** Encouraging creativity within project bounds.

Software Team Structures

Factors influencing structure: problem difficulty, program size, team lifetime, modularity, quality/reliability needs, delivery date rigidity, and required sociability.

Agile Teams

Characterized by trust, appropriate skill distribution, self-organization, autonomy, and adaptive structures.

Team Coordination & Communication

- **Formal, Impersonal:** Documents, memos, schedules, change requests, error reports.
- **Formal, Interpersonal:** Status review meetings, inspections.
- **Informal, Interpersonal:** Group meetings, "collocation."
- **Electronic:** Email, bulletin boards, conferencing.
- **Interpersonal Networking:** Informal discussions.

Product, Process, and Project Metrics

Why Measure?

- Assess project status.
- Track risks.
- Uncover problems early.
- Adjust workflow.
- Evaluate team's quality control ability.

Measures, Metrics, and Indicators

- **Measure:** A quantitative indication of an attribute's extent.
- **Metric:** A quantitative measure of the degree to which a system/component/process possesses an attribute.
- **Indicator:** A metric or combination providing insight.

Measurement Principles

- Establish objectives before data collection.
- Define metrics unambiguously.
- Metrics should be based on valid theory for the domain.
- Metrics should be tailored to specific products/processes.

Measurement Process

- **Formulation:** Deriving appropriate measures and metrics.
- **Collection:** Accumulating data.

- **Function-based:** Use Function Points (FP).
- **Specification metrics:** Count requirements by type.

Function-Based Metrics (Function Points - FP)

- Measures functionality delivered by a system.
- Derived from information domain values:
  - Number of External Inputs (EI)
  - Number of External Outputs (EO)
  - Number of External Inquiries (EQ)
  - Number of Internal Logical Files (ILF)
  - Number of External Interface Files (EIF)
- FP calculation involves weighting these values based on complexity.

Class-Oriented Metrics (Chidamber & Kemerer, Lorenz & Kidd)

Examples include: Weighted Methods per Class, Depth of Inheritance Tree, Number of Children, Coupling Between Object Classes, Response for a Class, Lack of Cohesion in Methods, Class Size, Operations Overridden/Added.

Process Measurement

- Measured indirectly through outcomes: errors uncovered, defects delivered, productivity, effort, time, schedule conformance.
- Also measured by characteristics of specific tasks.

Process Metrics Examples

- **Quality-related:** Work product quality.
- **Productivity-related:** Work products vs. effort.
- **Statistical SQA data:** Error categorization, defect removal efficiency.
- **Reuse data:** Number and reusability of components.

Project Metrics

- Minimize development schedule.
- Assess product quality continuously.
- Measure:
  - **Inputs:** Resources used.
  - **Outputs:** Deliverables created.
  - **Results:** Effectiveness of deliverables.

Typical Project Metrics

Effort/time per task, errors per review hour, scheduled vs. actual dates, number and characteristics of changes.

Typical Size-Oriented Metrics

Errors per KLOC (Thousand Lines of Code), Defects per KLOC, Cost per LOC, LOC per person-month.

Typical Function-Oriented Metrics

Errors per FP, Defects per FP, Cost per FP, FP per person-month.

Project Planning and Estimation

- Define required resources (human, reusable software, environmental).
- Estimate cost and effort (decompose problem, use multiple techniques, reconcile estimates).
- Develop a project schedule (define tasks, network, use techniques, track).

Estimation

Requires experience, historical data (metrics), and the courage to make quantitative predictions. Inherent risk leads to uncertainty.

Software Project Plan Components

Objectives, Project Scope, Estimates, Risks, Schedule, Control Strategy.

Objectives

Quantitative and qualitative goals that are measurable, unambiguous, and unique. Typically, at least three are established.

Scope

Describes functions, features, data input/output, performance, constraints, interfaces, and reliability. Defined via narrative description or use cases.

Project Estimation Considerations

- Scope must be understood.
- Elaboration (decomposition) is necessary.
- Historical metrics are helpful.
- Use at least two different techniques.
- Uncertainty is inherent.

Estimation Techniques

- Past project experience.
- Conventional techniques (task breakdown, size estimates).
- Empirical models.
- Automated tools.

Estimation Accuracy Factors

- Proper size estimation.
- Accurate translation of size to effort/time/cost (via reliable metrics).
- Team's abilities reflected in the plan.
- Stability of requirements and environment.

Functional Decomposition

Breaking down the statement of scope into smaller, manageable functional components.

Conventional Methods (LOC/FP Approach)

1. Estimate LOC or FP using information domain values.
2. Use historical data to build project estimates (effort, cost, schedule).

Example Calculation (LOC Approach)

Given estimated LOC for modules and average productivity (LOC/pm), calculate total estimated effort (pm) and cost (). $* Total LOC = Sum of LOC for all modules *$ $Estimated Effort(pm) = Total LOC / Average Productivity(LOC/pm) *$ $Estimated Cost = Estimated Effort(pm) * Burdened Labor Rate($/pm)

- Estimated Cost = Estimated Effort (pm) * Burdened Labor Rate ($/pm)

Project Scheduling

Why Projects Are Late?

Unrealistic deadlines, changing requirements, underestimation, unconsidered risks, unforeseen technical/human difficulties, miscommunication, and failure to act on schedule slippage.

Scheduling Principles

- **Compartmentalization:** Define distinct tasks.
- **Interdependency:** Indicate task relationships.
- **Effort Validation:** Ensure resources are available.
- **Defined Responsibilities:** Assign people to tasks.
- **Defined Outcomes:** Each task must have an output.
- **Defined Milestones:** Review for quality.

Effort Allocation (Typical)

- **Front-end activities (Communication, Analysis, Design, Review):** 40-50%
- **Construction (Coding):** 15-20%
- **Testing and Installation:** 30-40%

Defining Task Sets

Determine project type, required rigor, adaptation criteria, and select appropriate software engineering tasks.

Task Set Refinement

Detailed breakdown of tasks, including sub-tasks and formal technical reviews (FTRs).

Define a Task Network

Visual representation of task dependencies (e.g., using nodes for tasks and edges for dependencies).

Scheduling Techniques

- Gantt Chart
- Timeline Chart
- Network Diagram
- PERT Analysis
- Critical Path Scheduling (CPS)

Gantt Chart

A bar chart illustrating task durations and timelines.

Timeline Chart

Similar to Gantt charts, often used for visualizing project schedules over time.

PERT Analysis

- **Program Evaluation Review Technique.**
- Uses optimistic (o), pessimistic (p), and realistic (r) time estimates.
- Estimated Time (ET): $ET = \frac{o+4r+p}{6}$

Schedule Tracking

- Informal discussions with team members.

Critical Path Scheduling (CPS)

- Identifies the sequence of tasks that determines the shortest possible project completion time.
- **Critical Path:** The longest path through the network; any delay here delays the project.
- **Slack Time:** The amount of time an activity can be delayed without delaying the project.

Critical Path Calculation

1. Calculate **Earliest Possible Completion Time (TE)** for each activity.