SUDDHANANDA SCHOOL OF MANAGEMENT & COMPUTER SCIENCE

LECTURE NOTES ON

SOFTWARE TESTING AND QUALITY ASSURANCE (STQA)

UNIT – I

INTRODUCTION TO SOFTWARE TESTING AND QUALITY ASSURANCE:

INTRODUCTION TO SOFTWARE TESTING

Software Testing is the process of evaluating a software application to identify defects, errors, or missing requirements and to ensure that the software works according to specified requirements.

Testing helps improve software quality, reliability, performance, and security before the software is delivered to users.

Software testing is one of the most important phases of the Software Development Life Cycle (SDLC).

DEFINITION OF SOFTWARE TESTING

Software Testing is the process of executing a program or system with the intention of finding defects and verifying that it satisfies specified requirements.

OBJECTIVES OF SOFTWARE TESTING

1. Detect software defects and errors.
2. Verify software functionality.
3. Validate customer requirements.
4. Improve software quality.
5. Increase customer satisfaction.
6. Reduce maintenance cost.
7. Ensure software reliability.

NEED FOR SOFTWARE TESTING

Software systems are becoming increasingly complex. Errors in software may lead to financial loss, security breaches, or system failures.

Testing is necessary because:

- Developers may make mistakes.
- Requirements may be misunderstood.
- Complex systems contain hidden defects.
- Software must be reliable and secure.

SOFTWARE FAILURE, ERROR, DEFECT AND BUG

These terms are often used interchangeably, but they have different meanings.

Error

An error is a human mistake made during software development.

Example

A programmer writes an incorrect formula for calculating interest.

Defect (Bug)

A defect is a flaw in software caused by an error.

Example

Incorrect interest calculation in the software output.

Failure

A failure occurs when software does not perform its intended function.

Example

The banking application displays the wrong balance.

SOFTWARE QUALITY

Software Quality refers to the degree to which software meets specified requirements and satisfies customer expectations.

High-quality software should be:

- Reliable
- Efficient
- Secure
- Maintainable
- User-friendly

QUALITY ASSURANCE (QA)

Quality Assurance is a systematic process used to ensure that software development activities follow established standards and procedures.

QA focuses on preventing defects rather than finding them.

DEFINITION OF QUALITY ASSURANCE

Quality Assurance is a planned and systematic set of activities implemented within a quality system to provide confidence that software meets quality requirements.

OBJECTIVES OF QUALITY ASSURANCE

1. Prevent defects.
2. Improve development process.
3. Ensure compliance with standards.
4. Enhance customer confidence.
5. Deliver high-quality software.

SOFTWARE QUALITY ASSURANCE (SQA)

Software Quality Assurance is a set of activities that ensures software processes and products conform to requirements, standards, and procedures.

SQA covers:

- Process improvement
- Reviews
- Audits
- Testing
- Documentation

QUALITY CONTROL (QC)

Quality Control focuses on identifying defects in the final product.

It is a product-oriented activity.

DIFFERENCE BETWEEN QA AND QC

Quality Assurance (QA)	**Quality Control (QC)**
Process-oriented	Product-oriented
Prevents defects	Detects defects
Proactive approach	Reactive approach
Focus on process improvement	Focus on product testing

SOFTWARE QUALITY FACTORS

According to software quality models, important quality factors include:

Correctness

Software performs required functions correctly.

Example

Calculator produces accurate results.

Reliability

Software performs consistently without failure.

Example

Banking software runs continuously without crashing.

Efficiency

Software uses resources effectively.

Example

Fast response time.

Integrity

Protects data from unauthorized access.

Example

Password protection systems.

Usability

Software should be easy to use.

Example

Simple and intuitive user interface.

Maintainability

Software should be easy to modify and update.

Example

Adding new features with minimal effort.

Portability

Software should run on different platforms.

Example

Application running on Windows and Linux.

SOFTWARE TESTING PRINCIPLES

The fundamental principles of software testing are:

Principle 1: Testing Shows Presence of Defects

Testing can show defects exist but cannot prove their absence.

Principle 2: Exhaustive Testing is Impossible

Testing every possible input and condition is impractical.

Principle 3: Early Testing

Testing should begin as early as possible.

Principle 4: Defect Clustering

Most defects are usually concentrated in a few modules.

Principle 5: Pesticide Paradox

Repeating the same tests may not find new defects.

Principle 6: Testing is Context Dependent

Different applications require different testing approaches.

Principle 7: Absence of Errors Fallacy

Software without defects may still fail if it does not meet user needs.

VERIFICATION AND VALIDATION:

Verification

Verification ensures that the software is being built correctly.

Question:

Are we building the product right?

Activities:

- Reviews
- Inspections
- Walkthroughs

Validation

Validation ensures that the correct software is being built.

Question:

Are we building the right product?

Activities:

- Testing
- User Acceptance Testing

DIFFERENCE BETWEEN VERIFICATION AND VALIDATION

Verification	**Validation**
Process-oriented	Product-oriented
Static activity	Dynamic activity
No code execution required	Code execution required
Prevents defects	Detects defects

SOFTWARE TESTING LIFE CYCLE (STLC)

STLC defines the phases involved in software testing.

PHASES OF STLC

Requirement Analysis

↓

Test Planning

↓

Test Case Development

↓

Test Environment Setup

↓

Test Execution

↓

Defect Reporting

↓

Test Closure

Requirement Analysis

Study software requirements.

Test Planning

Prepare testing strategy.

Test Case Development

Create test cases and test data.

Test Environment Setup

Prepare hardware and software environment.

Test Execution

Run test cases.

Defect Reporting

Record and report defects.

Test Closure

Evaluate testing completion.

TEST CASE

A Test Case is a set of conditions and inputs used to verify a particular functionality.

Components of Test Case

1. Test Case ID
2. Test Description
3. Input Data
4. Expected Result
5. Actual Result

6. Status (Pass/Fail)

EXAMPLE OF TEST CASE

Test Case ID	**TC01**
Function	Login
Input	Valid Username & Password
Expected Result	Login Successful
Actual Result	Login Successful
Status	Pass

TEST SUITE

A Test Suite is a collection of related test cases executed together.

TEST PLAN

A Test Plan is a document describing the testing scope, objectives, resources, schedule, and strategy.

BENEFITS OF SOFTWARE TESTING

1. Improved software quality.
2. Reduced maintenance cost.
3. Increased reliability.
4. Better security.
5. Higher customer satisfaction.
6. Early defect detection.

CHALLENGES IN SOFTWARE TESTING

1. Limited testing time.
2. Frequent requirement changes.
3. Complex software systems.
4. Resource constraints.
5. Incomplete documentation.

ROLE OF TEST ENGINEER

A Test Engineer is responsible for:

1. Creating test plans.
2. Designing test cases.

3. Executing tests.
4. Reporting defects.
5. Ensuring software quality.

UNIT – II

SOFTWARE TESTING TECHNIQUES:

INTRODUCTION

Software testing techniques are methods used to identify defects and verify that software functions correctly according to requirements. These techniques help testers design effective test cases and improve software quality.

Software testing techniques are mainly classified into:

1. Black Box Testing
2. White Box Testing
3. Gray Box Testing

BLACK BOX TESTING

Black Box Testing is a testing technique in which the internal structure, code, and implementation details of the software are not known to the tester. The tester focuses only on inputs and outputs.

The tester provides input data and checks whether the software produces the expected output.

Features of Black Box Testing

- No knowledge of programming is required.
- Testing is based on requirements and specifications.
- Focuses on functionality.
- Performed by testers and end users.

Advantages

- Easy to perform.
- Suitable for large systems.
- Independent of programming language.

Disadvantages

- Limited coverage of internal code.
- Difficult to identify hidden errors.

Example

Suppose a login page accepts a username and password.

Input:
Username = Pritam
Password = 12345

Expected Result:
Login successful.

If the application allows access, the test case passes.

TYPES OF BLACK BOX TESTING

Equivalence Partitioning

Equivalence Partitioning divides input data into groups called equivalence classes. Test cases are selected from each group.

The assumption is that all values within a partition behave similarly.

Example

Age field accepts values from 18 to 60.

Valid Partition:
18–60

Invalid Partition:
Less than 18
Greater than 60

Test Cases:

- 25 (Valid)
- 15 (Invalid)
- 65 (Invalid)

This reduces the number of test cases.

Advantages

- Reduces testing effort.
- Saves time.
- Improves efficiency.

Boundary Value Analysis (BVA)

Boundary Value Analysis focuses on testing values at the boundaries because defects often occur at boundary conditions.

Example

Age range = 18 to 60

Boundary Test Cases:

- 17
- 18
- 19
- 59
- 60
- 61

These values check the lower and upper limits.

Advantages

- Detects boundary-related defects.
- Provides better coverage.

Decision Table Testing

Decision Table Testing is used when software behavior depends on multiple conditions.

A decision table contains conditions and corresponding actions.

Example

Online Shopping Discount System

Membership	Purchase > ₹5000	Discount
Yes	Yes	20%
Yes	No	10%
No	Yes	5%
No	No	0%

Test cases are derived from each combination.

Advantages

- Useful for business applications.
- Covers all possible combinations.

State Transition Testing

State Transition Testing verifies system behavior for different states and transitions.

Example

ATM Card System

States:

- Active
- Blocked

If incorrect PIN is entered three times:

Active → Blocked

Testing verifies correct state transitions.

Advantages

- Suitable for real-time systems.
- Detects state-related errors.

WHITE BOX TESTING

White Box Testing is a testing technique where the tester has complete knowledge of the program's internal structure and source code.

The tester examines the code, logic, conditions, loops, and paths.

Features

- Requires programming knowledge.
- Tests internal logic.
- Ensures code quality.

Advantages

- Detects hidden errors.
- Improves code quality.
- Covers internal paths.

Disadvantages

- Time-consuming.
- Requires skilled testers.

Example

A program contains:

```
if(marks >= 40)
System.out.println("Pass");
else
System.out.println("Fail");
```

The tester checks both paths:

marks = 50 → Pass

marks = 30 → Fail

TYPES OF WHITE BOX TESTING

Statement Coverage

Statement Coverage ensures every statement in the program executes at least once.

Example

int a = 10;

int b = 20;

int c = a + b;

Testing should execute all statements.

Formula

\text{Statement Coverage}=\frac{\text{Executed Statements}}{\text{Total Statements}}\times100

Branch Coverage

Branch Coverage ensures every branch or decision outcome is executed.

Example

if(age >= 18)

System.out.println("Eligible");

else

System.out.println("Not Eligible");

Both true and false branches must be tested.

Formula

\text{Branch Coverage}=\frac{\text{Executed Branches}}{\text{Total Branches}}\times100

Condition Coverage

Condition Coverage verifies every logical condition.

Example

if(a>0 && b>0)

Test Cases:

- a=5, b=5
- a=-5, b=5
- a=5, b=-5

All conditions are tested.

PATH TESTING

Path Testing ensures every possible execution path is tested.

Example

```
if(a>b)
{
if(a>c)
largest=a;
}
else
largest=b;
```

Different paths are executed to verify correctness.

Advantages

- Provides thorough testing.
- Detects logical errors.

LOOP TESTING

Loop Testing focuses on validating loops.

Types

1. Simple Loop
2. Nested Loop
3. Concatenated Loop

Example

```
for(int i=1;i<=5;i++)
{
System.out.println(i);
}
```

Test Cases:

- Zero iteration
- One iteration
- Maximum iteration
- Invalid iteration

CONTROL FLOW TESTING

Control Flow Testing examines the sequence of execution within a program.

It uses:

- Control Flow Graph (CFG)
- Nodes
- Edges

Advantages

- Detects logical errors.
- Improves program reliability.

GRAY BOX TESTING

Gray Box Testing is a combination of Black Box Testing and White Box Testing.

The tester has partial knowledge of the internal structure.

Features

- Partial access to source code.
- Focuses on both functionality and design.
- Useful for integration testing.

Advantages

- Better coverage.
- Detects interface defects.
- Improves overall quality.

Disadvantages

- Requires technical knowledge.
- More complex than black box testing.

Example

A tester knows the database structure but tests the application through the user interface.

COMPARISON OF TESTING TECHNIQUES

Feature	Black Box	White Box	Gray Box
Knowledge of Code	No	Yes	Partial
Focus	Functionality	Internal Logic	Both
Programming Knowledge	Not Required	Required	Partial
Performed By	Tester	Developer	Tester/Developer

TEST CASE DESIGN TECHNIQUES

Test Case Design Techniques help create effective test cases.

Major techniques are:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing

- Path Testing

These techniques improve test coverage and increase the probability of detecting defects.

BENEFITS OF TESTING TECHNIQUES

- Early defect detection.
- Improved software quality.
- Better reliability.
- Reduced maintenance cost.
- Increased customer satisfaction.

UNIT – III

LEVELS OF TESTING, INTEGRATION TESTING, SYSTEM TESTING, ACCEPTANCE TESTING, REGRESSION TESTING AND TEST MANAGEMENT

INTRODUCTION:

Software testing is performed at different stages of software development. Each stage focuses on finding different types of defects. These stages are known as Levels of Testing.

The main testing levels are:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

These testing levels ensure that software works correctly from individual components to the complete system.

UNIT TESTING

Unit Testing is the process of testing individual modules or components of a software application independently.

It is usually performed by developers during coding.

The objective of unit testing is to verify that each unit performs its intended function correctly.

A unit may be:

- Function
- Method
- Class
- Module

Objectives of Unit Testing

- Verify correctness of individual modules.
- Detect coding errors.
- Simplify debugging.
- Improve software quality.

Advantages

- Early defect detection.
- Easy debugging.
- Reduces development cost.
- Improves code quality.

Example

Suppose a function calculates the sum of two numbers.

```
int add(int a,int b)

{

return a+b;
```

}

Test Cases:

Input: 10,20

Expected Output: 30

Input: 5,8

Expected Output: 13

If the output matches the expected result, the test case passes.

INTEGRATION TESTING

Integration Testing is performed after Unit Testing.

In this testing, multiple modules are combined and tested together to verify their interaction.

The main objective is to identify defects in interfaces and communication between modules.

Need for Integration Testing

Even if individual modules work correctly, errors may occur when modules interact with each other.

Example:

Login Module → Database Module

If data transfer fails, integration defects occur.

Types of Integration Testing

1. Big Bang Integration Testing
2. Incremental Integration Testing

BIG BANG INTEGRATION TESTING

All modules are integrated simultaneously and tested as a complete system.

Advantages

- Simple approach.
- Suitable for small projects.

Disadvantages

- Difficult defect identification.
- Time-consuming debugging.

Example

All modules of an Online Shopping System are integrated together and tested.

INCREMENTAL INTEGRATION TESTING

Modules are integrated gradually and tested step by step.

Advantages

- Easier defect detection.

- Better control.

Types

1. Top-Down Testing
2. Bottom-Up Testing
3. Sandwich Testing

TOP-DOWN INTEGRATION TESTING

Testing starts from top-level modules and gradually moves to lower-level modules.

Lower-level modules not yet developed are replaced using Stubs.

Stub

A Stub is a temporary program used to simulate lower-level modules.

Advantages

- Early testing of major functions.
- User interface tested first.

Disadvantages

- Many stubs required.

BOTTOM-UP INTEGRATION TESTING

Testing begins with lower-level modules and gradually moves upward.

Higher-level modules are replaced using Drivers.

Driver

A Driver is a temporary program used to simulate higher-level modules.

Advantages

- Low-level modules thoroughly tested.
- No stubs required.

Disadvantages

- User interface tested late.

SANDWICH TESTING

Sandwich Testing combines Top-Down and Bottom-Up approaches.

Testing proceeds simultaneously from top and bottom.

Advantages

- Faster testing.
- Better coverage.

Disadvantages

- Complex management.

SYSTEM TESTING

System Testing is performed after Integration Testing.

The complete integrated software system is tested against specified requirements.

The objective is to evaluate the behavior of the entire system.

Characteristics

- Tests complete application.
- Performed in real environment.
- Focuses on functional and non-functional requirements.

Objectives

- Verify system functionality.
- Validate requirements.
- Detect environment-related issues.

Example

Testing a complete Banking Application including:

- Login
- Account Management
- Fund Transfer
- Report Generation

TYPES OF SYSTEM TESTING

Functional Testing

Verifies software functions according to requirements.

Performance Testing

Measures speed, responsiveness, and stability.

Security Testing

Identifies vulnerabilities and security weaknesses.

Usability Testing

Evaluates user friendliness.

Compatibility Testing

Checks software operation on different platforms.

Reliability Testing

Measures software stability over time.

PERFORMANCE TESTING

Performance Testing evaluates system performance under different workloads.

Objectives

- Measure response time.
- Measure throughput.
- Evaluate stability.

Types

Load Testing

Tests behavior under expected user load.

Example:

1000 users accessing a website simultaneously.

Stress Testing

Tests behavior beyond normal limits.

Example:

5000 users accessing a website simultaneously.

Volume Testing

Tests large amounts of data.

Example:

Database containing millions of records.

SECURITY TESTING

Security Testing identifies vulnerabilities that may lead to unauthorized access.

Objectives

- Protect data.
- Ensure confidentiality.
- Verify authentication mechanisms.

Example

Testing login security against SQL Injection attacks.

COMPATIBILITY TESTING

Compatibility Testing verifies software operation across different environments.

Example

Testing application on:

- Windows
- Linux
- Android
- iOS

USABILITY TESTING

Usability Testing evaluates how easy the software is to learn and use.

Factors Evaluated

- User Interface
- Navigation
- Accessibility
- Ease of Use

ACCEPTANCE TESTING

Acceptance Testing is the final level of testing performed before software release.

It determines whether the software satisfies business requirements and customer expectations.

Objectives

- Validate requirements.
- Gain customer approval.
- Decide product release.

Types of Acceptance Testing

1. Alpha Testing
2. Beta Testing

ALPHA TESTING

Alpha Testing is performed by developers or testers at the developer's site.

Characteristics

- Conducted before Beta Testing.
- Controlled environment.
- Internal users involved.

Advantages

- Early defect identification.
- Improves software quality.

BETA TESTING

Beta Testing is performed by actual users in real environments.

Characteristics

- Conducted after Alpha Testing.
- Real-world usage.
- Customer feedback collected.

Advantages

- Identifies practical issues.
- Improves customer satisfaction.

Example

Testing a mobile application by releasing it to selected users before official launch.

REGRESSION TESTING

Regression Testing ensures that modifications or bug fixes have not introduced new defects into existing functionality.

Need for Regression Testing

Whenever software is modified:

- New features added.
- Bugs fixed.
- Performance improved.

Existing functionality must be retested.

Advantages

- Prevents side effects.
- Ensures system stability.
- Improves reliability.

Example

After fixing login bugs, all login-related functionalities are retested.

SMOKE TESTING

Smoke Testing verifies whether the major functionalities of a software build work correctly.

It is performed immediately after receiving a new build.

Example

Checking:

- Login
- Dashboard
- Database Connection

before detailed testing.

SANITY TESTING

Sanity Testing verifies specific changes or bug fixes.

Example

Testing only password reset functionality after fixing a password-related defect.

TEST MANAGEMENT

Test Management involves planning, monitoring, controlling, and reporting testing activities.

It ensures testing is completed efficiently and effectively.

Objectives

- Improve testing quality.
- Optimize resources.
- Track progress.

TEST PLANNING

Test Planning is the process of defining testing scope, objectives, resources, schedule, and strategy.

Components of Test Plan

- Testing objectives
- Scope
- Resources
- Schedule
- Risks
- Deliverables

TEST EXECUTION

Test Execution is the process of running test cases and comparing actual results with expected results.

Activities

- Execute test cases.
- Record results.
- Report defects.

DEFECT MANAGEMENT

Defect Management is the process of identifying, documenting, tracking, and resolving defects.

Defect Life Cycle

New

↓

Assigned

↓

Open

↓

Fixed

↓

Retest

↓

Closed

Defect Report Contents

- Defect ID
- Description
- Severity
- Priority
- Status
- Assigned Developer

TEST METRICS

Test Metrics are measurements used to evaluate testing effectiveness.

Examples:

- Number of test cases executed.
- Number of defects found.
- Defect density.
- Test coverage.

Defect Density Formula

\text{Defect Density}=\frac{\text{Number of Defects}}{\text{Size of Software}}

BENEFITS OF TEST MANAGEMENT

- Better planning.
- Efficient resource utilization.
- Improved product quality.
- Faster defect resolution.
- Better project control.

UNIT – IV

TEST AUTOMATION, TEST TOOLS, CONFIGURATION MANAGEMENT, SOFTWARE QUALITY MODELS, SOFTWARE REVIEWS AND AUDITS:

INTRODUCTION

As software systems become larger and more complex, manual testing alone becomes difficult, time-consuming, and expensive. To improve testing efficiency and software quality, organizations use automated testing tools, quality models, reviews, audits, and configuration management techniques.

This unit focuses on methods that help improve software quality and maintain consistency throughout the software development process.

TEST AUTOMATION

Test Automation is the process of using software tools and scripts to execute test cases automatically without human intervention.

Automated testing helps reduce repetitive manual work and improves testing efficiency.

Objectives of Test Automation

- Reduce testing time.
- Improve accuracy.
- Increase test coverage.
- Minimize human errors.
- Support continuous testing.

Advantages of Test Automation

- Faster execution of test cases.
- Reusable test scripts.
- Better accuracy.
- Reduced testing cost.
- Supports regression testing.

Disadvantages of Test Automation

- High initial cost.
- Requires skilled professionals.
- Tool maintenance is necessary.
- Not suitable for all test cases.

Example

A login test executed automatically 100 times using automation software instead of manual execution.

AUTOMATION FRAMEWORK

An Automation Framework is a set of guidelines and standards used to develop and execute automated test scripts.

Types of Automation Frameworks

1. Linear Framework
2. Modular Framework
3. Data-Driven Framework
4. Keyword-Driven Framework
5. Hybrid Framework

Data-Driven Framework

Test data is stored separately from test scripts.

Advantages:

- Easy maintenance.
- Better reusability.

Keyword-Driven Framework

Uses predefined keywords to perform testing actions.

Examples:

- Login
- Click
- Submit

Advantages:

- Easy for non-programmers.
- Improves readability.

TEST AUTOMATION TOOLS

Automation tools are software applications used to automate testing activities.

Selenium

Selenium is one of the most popular open-source automation testing tools.

Features:

- Supports multiple browsers.
- Supports multiple programming languages.
- Suitable for web application testing.

Advantages:

- Free and open source.
- Large community support.
- Platform independent.

TestComplete

TestComplete is a commercial automation testing tool.

Features:

- Record and playback testing.
- Supports desktop, web, and mobile applications.

QTP/UFT

QTP (Quick Test Professional), now called UFT (Unified Functional Testing), is developed by Micro Focus.

Features:

- Automated functional testing.
- GUI testing support.

JUnit

JUnit is a testing framework for Java applications.

Uses:

- Unit testing.
- Test automation.

Example:

```
@Test
public void testAddition()
```

{

assertEquals(10,5+5);

}

TestNG

TestNG is an advanced Java testing framework inspired by JUnit.

Features:

- Parallel testing.
- Test grouping.
- Flexible reporting.

CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) is the process of controlling and managing changes made to software products throughout their life cycle.

It ensures that all software components remain consistent and traceable.

Objectives of SCM

- Control software changes.
- Maintain software integrity.
- Manage versions.
- Improve project coordination.

Need for Configuration Management

Software projects undergo frequent modifications.

Without proper control:

- Files may be lost.
- Versions may conflict.
- Errors may increase.

SCM prevents these issues.

CONFIGURATION ITEM (CI)

A Configuration Item is any software component managed under configuration control.

Examples:

- Source code
- Documents
- Test cases
- Design documents

ACTIVITIES OF CONFIGURATION MANAGEMENT

Configuration Identification

Identifying software components to be controlled.

Configuration Control

Managing and approving changes.

Configuration Status Accounting

Maintaining records of changes.

Configuration Auditing

Verifying configuration correctness.

VERSION CONTROL

Version Control is the process of managing multiple versions of software.

Advantages

- Tracks changes.
- Supports collaboration.
- Allows rollback to previous versions.

Popular Version Control Tools

- Git
- GitHub
- GitLab
- Bitbucket

Example

Version 1.0 → Initial Release

Version 1.1 → Bug Fixes

Version 2.0 → New Features

SOFTWARE QUALITY MODELS

Software Quality Models define characteristics used to evaluate software quality.

These models help organizations measure and improve software quality.

McCall's Quality Model

Introduced by Jim McCall.

The model classifies quality factors into:

1. Product Operation
2. Product Revision
3. Product Transition

Product Operation Factors

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability

Product Revision Factors

- Maintainability
- Flexibility
- Testability

Product Transition Factors

- Portability
- Reusability
- Interoperability

Advantages

- Comprehensive quality evaluation.
- Widely accepted model.

BOEHM QUALITY MODEL

Barry Boehm proposed a hierarchical software quality model.

Main characteristics:

- Reliability
- Efficiency
- Human Engineering
- Testability
- Portability
- Maintainability

Advantages:

- Better quality assessment.
- Supports software improvement.

ISO 9126 QUALITY MODEL

ISO 9126 is an international standard for software quality evaluation.

Quality characteristics include:

1. Functionality
2. Reliability

3. Usability
4. Efficiency
5. Maintainability
6. Portability

Functionality

Ability to satisfy stated requirements.

Reliability

Ability to perform consistently.

Usability

Ease of learning and operation.

Efficiency

Optimal resource utilization.

Maintainability

Ease of modification.

Portability

Ability to operate on different platforms.

SOFTWARE REVIEWS

A Software Review is a systematic examination of software artifacts to identify defects and improve quality.

Reviews are considered static testing techniques because code execution is not required.

Objectives of Reviews

- Detect defects early.
- Improve quality.
- Reduce development cost.
- Ensure standards compliance.

TYPES OF SOFTWARE REVIEWS

Informal Review

A simple review without a formal process.

Advantages:

- Quick and inexpensive.

Walkthrough

The author explains the software product to reviewers.

Advantages:

- Knowledge sharing.
- Early defect identification.

Technical Review

Experts evaluate technical aspects of software.

Advantages:

- Improves design quality.

Inspection

A formal review technique introduced by Michael Fagan.

Advantages:

- Highly effective defect detection.

SOFTWARE INSPECTION

Inspection is the most formal review method.

Participants include:

- Moderator
- Author
- Reviewer
- Recorder

Inspection Process

Planning

↓

Preparation

↓

Inspection Meeting

↓

Rework

↓

Follow-Up

Advantages

- High defect detection rate.
- Reduced testing effort.
- Improved software quality.

SOFTWARE AUDIT

A Software Audit is an independent examination of software processes, products, and activities.

Audits verify whether software development follows established standards and procedures.

Objectives of Audit

- Ensure compliance.
- Improve quality.
- Identify process weaknesses.
- Reduce project risks.

TYPES OF AUDITS

Internal Audit

Conducted within the organization.

External Audit

Conducted by external agencies.

Process Audit

Evaluates development processes.

Product Audit

Evaluates final software products.

DIFFERENCE BETWEEN REVIEW AND AUDIT

Review	**Audit**
Detects defects	Ensures compliance
Conducted by project team	Conducted by independent authority
Improves quality	Verifies standards
Technical focus	Process focus

QUALITY ASSURANCE PLAN

A Quality Assurance Plan defines procedures, standards, responsibilities, and activities required to ensure software quality.

Components

- Quality objectives
- Testing procedures
- Review procedures
- Audit procedures
- Reporting mechanisms

SOFTWARE QUALITY METRICS

Quality Metrics measure software quality and process effectiveness.

Examples:

- Defect Density
- Test Coverage
- Failure Rate
- Mean Time Between Failures (MTBF)

Mean Time Between Failures

MTBF measures system reliability.

Formula:

MTBF=\frac{\text{Total Operating Time}}{\text{Number of Failures}}

Higher MTBF indicates better reliability.

BENEFITS OF QUALITY ASSURANCE ACTIVITIES

- Improved software quality.
- Reduced defects.
- Better customer satisfaction.
- Lower maintenance cost.
- Increased reliability.
- Better compliance with standards.