Software Testing

Part II

Introduction & Fundamentals

- What is Software Testing?
- Why testing is necessary?
- Who does the testing?
- What has to be tested?
- When is testing done?
- How often to test?

Most Common Software problems

- Incorrect calculation
- Incorrect data edits & ineffective data edits
- Incorrect matching and merging of data
- Data searches that yields incorrect results
- Incorrect processing of data relationship
- Incorrect coding / implementation of business rules
- Inadequate software performance

Most Common Software problems

- Confusing or misleading data
- Software usability by end users & Obsolete Software
- Inconsistent processing
- Unreliable results or performance
- Inadequate support of business needs
- Incorrect or inadequate interfaces with other systems
- Inadequate performance and security controls
- Incorrect file handling

Objectives of testing

- Executing a program with the intent of finding an ***error***.
- To check if the system meets the **requirements** and be **executed successfully** in the Intended environment.
- To check if the system is " **Fit for purpose**".
- To check if the system **does what it is expected to do**.

Objectives of testing

- **A good test case** is one that **has a probability of finding** an as yet **undiscovered error**.
- **A successful test** is one that uncovers a yet undiscovered error.
- **A good test** is not redundant.
- **A good test** should be "best of kind".
- A good test should **neither be too simple** nor **too complex**.

Objective of a Software Tester

- **Find bugs as early as possible** and make sure they get fixed.
- To understand the application well.
- Study the functionality in detail to find where the bugs are likely to occur.
- Study the code to ensure that each and every line of code is tested.
- Create test cases in such a way that testing is done to uncover the hidden bugs and also ensure that the software is usable and reliable

VERIFICATION & VALIDATION

Verification – typically involves reviews and meeting to evaluate documents, plans, code, requirements, and specifications. This can be done with checklists, issues lists, walkthroughs, and inspection meeting.

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

Validation and Verification process continue in a cycle till the software becomes defects free.

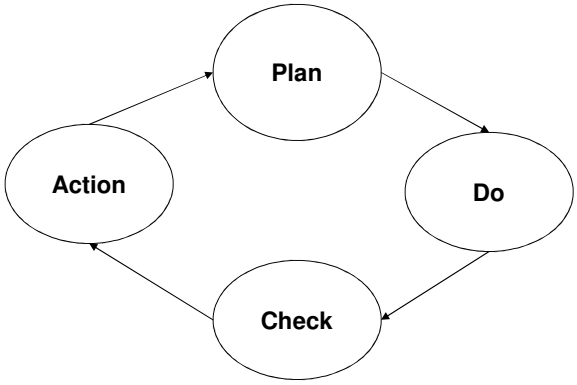
Summary

- **Testing**
  The execution of a program to find its faults
- **Verification**
  The process of proving the programs correctness.
- **Validation**
  The process of finding errors by executing the program in a real environment
- **Debugging**
  Diagnosing the error and correct it

Software Development

Process Cycle

Software Development Process Cycle



- **PLAN (P):** Device a plan. Define your objective and determine the strategy and supporting methods required to achieve that objective.
- **DO (D):** Execute the plan. Create the conditions and perform the necessary training to execute the plan.
- **CHECK (C):** Check the results. Check to determine whether work is progressing according to the plan and whether the results are obtained.
- **ACTION (A):** Take the necessary and appropriate action if checkup reveals that the work is not being performed according to plan or not as anticipated.

QUALITY PRINCIPLES

Quality - the most important factor affecting an organization's long-term performance.

Quality - the way to achieve improved productivity and competitiveness in any organization.

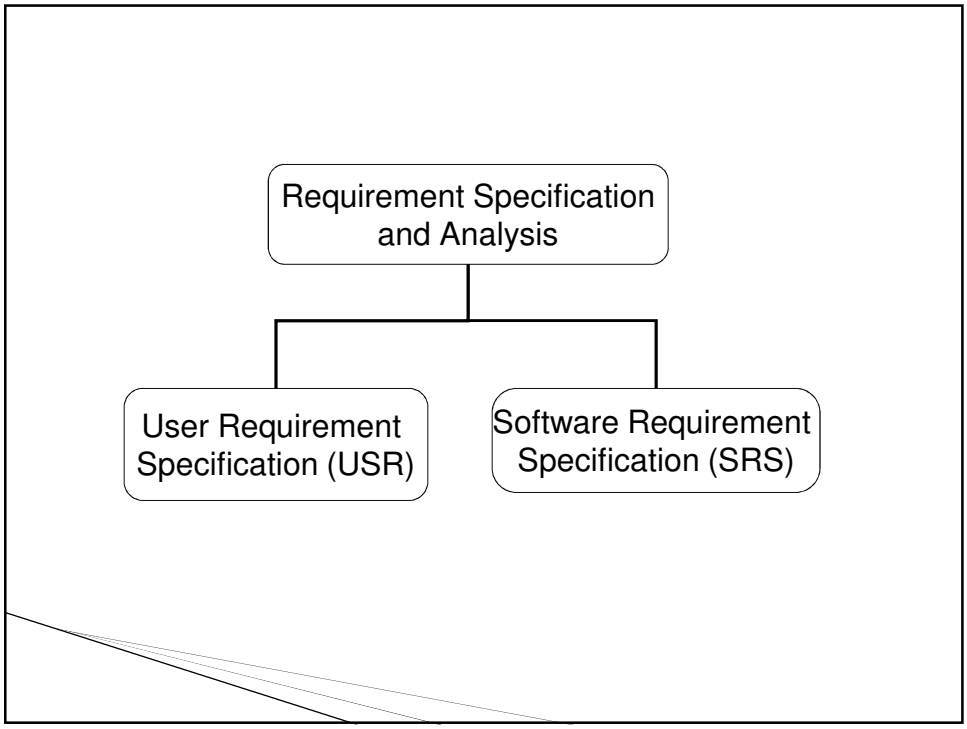
Quality - saves. It does not cost.

Quality - is the solution to the problem, not a problem.

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Phases of SDLC

- Requirement Specification and
- Analysis
- Design
- Coding
- Testing
- Implementation
- Maintenance



Design

The output of SRS is the input of design phase.

Two types of design -

High Level Design (HLD)
Low Level Design (LLD)

High Level Design (HLD)

- List of modules and a brief description of each module.
- Brief functionality of each module.
- Interface relationship among modules.
- Dependencies between modules (if A exists, B exists etc).
- Database tables identified along with key elements.
- Overall architecture diagrams along with technology details.

Low Level Design (LLD)

- Detailed functional logic of the module, in pseudo code.
- Database tables, with all elements, including their type and size.
- All interface details.
- All dependency issues
- Error message listings
- Complete input and outputs for a module.

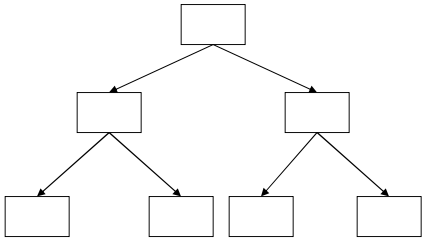
The Design process

Breaking down the product into independent modules to arrive at micro levels.

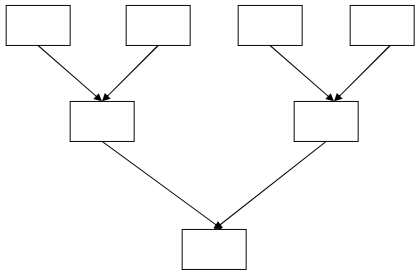
2 different approaches followed in designing –

Top Down Approach
Bottom Up Approach

Top-down approach



Bottom-Up Approach



Coding

Developers use the LLD document and write the code in the programming language specified.

Testing

The testing process involves development of a test plan, executing the plan and documenting the test results.

Implementation

Installation of the product in its operational environment.

Maintenance

After the software is released and the client starts using the software, maintenance phase is started.

3 things happen - **Bug fixing**, **Upgrade**, **Enhancement**

Bug fixing – bugs arrived due to some untested scenarios.

Upgrade – Upgrading the application to the newer versions of the software.

Enhancement - Adding some new features into the existing software.

Software Testing

Life Cycle

SOFTWARE TESTING LIFECYCLE - PHASES

- Requirements study
- Test Case Design and Development
- Test Execution
- Test Closure
- Test Process Analysis

Requirements study

- Testing Cycle starts with the study of client's requirements.
- Understanding of the requirements is very essential for testing the product.

Analysis & Planning

- Test objective and coverage
- Overall schedule
- Standards and Methodologies
- Resources required, including necessary training
- Roles and responsibilities of the team members
- Tools used

Test Case Design and Development

- Component Identification
- Test Specification Design
- Test Specification Review

Test Execution

- Code Review
- Test execution and evaluation
- Performance and simulation

Test Closure

- Test summary report
- Project Documentation

Test Process Analysis

Analysis done on the reports and improving the application's performance by implementing new technology and additional features.

Testing Levels

- Unit testing
- Integration testing
- System testing
- Acceptance testing

Unit testing

- The most 'micro' scale of testing.
- Tests done on particular functions or code modules.
- Requires knowledge of the internal program design and code.
- **Done by Programmers** (not by testers).

Unit testing

Objectives	• To test the function of a program or unit of code such as a program or module • To test internal logic • To verify internal design • To test path & conditions coverage • To test exception conditions & error handling
When	• After modules are coded
Input	• Internal Application Design • Master Test Plan • Unit Test Plan
Output	• Unit Test Report

Who	•Developer
Methods	•**White Box** testing techniques •Test Coverage techniques
Tools	•Debug •Re-structure •Code Analyzers •Path/statement coverage tools
Education	•Testing Methodology •Effective use of tools

Incremental integration testing

- Continuous testing of an application as and when a new functionality is added.
- Application's functionality aspects are required to be independent enough to work separately before completion of development.
- **Done by programmers or testers**.

Integration Testing

- Testing of combined parts of an application to determine their functional correctness.
- 'Parts' can be
  - code modules
  - individual applications
  - client/server applications on a network.

Types of Integration Testing

- Top Down Integration testing
- Bottom Up Integration testing

Integration testing

Objectives	• To technically verify proper interfacing between modules, and within sub-systems
When	• After modules are unit tested
Input	• Internal & External Application Design • Master Test Plan • Integration Test Plan
Output	• Integration Test report

Who	•Developers
Methods	•White and Black Box techniques •Problem / Configuration Management
Tools	•Debug •Re-structure •Code Analyzers
Education	•Testing Methodology •Effective use of tools

System Testing

Objectives	• To verify that the system components perform control functions • To perform inter-system test • To demonstrate that the system performs both functionally and operationally as specified • To perform appropriate types of tests relating to Transaction Flow, Installation, Reliability, Regression etc.
When	• After Integration Testing
Input	• Detailed Requirements & External Application Design • Master Test Plan • System Test Plan
Output	• System Test Report

Who	•Development Team and Users
Methods	•Problem / Configuration Management
Tools	•Recommended set of tools
Education	•Testing Methodology •Effective use of tools

Systems Integration Testing

Objectives	• To test the co-existence of products and applications that are required to perform together in the production-like operational environment (hardware, software, network) • To ensure that the system functions together with all the components of its environment as a total system • To ensure that the system releases can be deployed in the current environment
When	• After system testing • Often performed outside of project life-cycle
Input	• Test Strategy • Master Test Plan • Systems Integration Test Plan
Output	• Systems Integration Test report

Who	•System Testers
Methods	•White and Black Box techniques •Problem / Configuration Management
Tools	•Recommended set of tools
Education	•Testing Methodology •Effective use of tools

Acceptance Testing

Objectives	• To verify that the system meets the user requirements
When	• After System Testing
Input	• Business Needs & Detailed Requirements • Master Test Plan • User Acceptance Test Plan
Output	• User Acceptance Test report

Who	Users / End Users
Methods	•Black Box techniques •Problem / Configuration Management
Tools	Compare, keystroke capture & playback, regression testing
Education	•Testing Methodology •Effective use of tools •Product knowledge •Business Release Strategy

TESTING METHODOLOGIES

AND TYPES

Testing methodologies

- Black box testing
- White box testing
- Incremental testing

- **Black box testing**
  - No knowledge of internal design or code required.
  - Tests are based on requirements and functionality
- **White box testing**
  - Knowledge of the internal program design and code required.
  - Tests are based on coverage of code statements, branches, paths, conditions.

Black box / Functional testing

- Based on requirements and functionality
- Not based on any knowledge of internal design or code
- Covers all combined parts of a system
- Tests are data driven

White box testing / Structural testing

- Based on knowledge of internal logic of an application's code
- Based on coverage of code statements, branches, paths, conditions
- Tests are logic driven

Black Box

Testing Techniques

Black Box - testing technique

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Performance errors
- Initialization and termination errors

Functional testing

- Black box type testing geared to functional requirements of an application.
- Done by testers.

System testing

- Black box type testing that is based on overall requirements specifications; covering all combined parts of the system.

End-to-end testing

- Similar to system testing; involves testing of a complete application environment in a situation that mimics real-world use.

Sanity testing

- Initial effort to determine if a new software version is performing well enough to accept it for a major testing effort.

Regression testing

- Re-testing after fixes or modifications of the software or its environment.

Acceptance testing

- Final testing based on specifications of the end-user or customer

Load testing

- Testing an application under heavy loads.
- Eg. Testing of a web site under a range of loads to determine, when the system response time degraded or fails.

Stress Testing

- Testing under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database etc.

Performance testing

- Testing how well an application complies to performance requirements.

Install/uninstall testing

- Testing of full, partial or upgrade install/uninstall process.

Recovery testing

- Testing how well a system recovers from crashes, HW failures or other problems.

Compatibility testing

- Testing how well software performs in a particular HW/SW/OS/NW environment.

Exploratory testing / ad-hoc testing

- Informal SW test that is not based on formal test plans or test cases; testers will be learning the SW in totality as they test it.

Comparison testing

- Comparing SW strengths and weakness to competing products.

Alpha testing

- Testing done when development is nearing completion; minor design changes may still be made as a result of such testing.

Beta-testing

- Testing when development and testing are essentially completed and final bugs and problems need to be found before release.

Mutation testing

- To determining if a set of test data or test cases is useful, by deliberately introducing various bugs.
- Re-testing with the original test data/cases to determine if the bugs are detected.

White Box

Testing Techniques

White Box - testing technique

- All independent paths within a module have been exercised at least once
- Exercise all logical decisions on their *true* and *false* sides
- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

Loop Testing

This white box technique focuses on the validity of loop constructs.

4 different classes of loops can be defined

- **simple loops**
- **nested loops**
- **concatenated loops**
- **Unstructured loops**

Other White Box Techniques

Statement Coverage
+ execute all statements at least once

Decision Coverage
+ execute each decision direction at least once

Condition Coverage
+ execute each decision with all possible outcomes at least once

Decision / Condition coverage
+ execute all possible combinations of condition outcomes in each decision.

Multiple condition Coverage
+ invokes each point of entry at least once.

Incremental Testing

- A disciplined method of testing the interfaces between unit-tested programs as well as between system components.
- Involves adding unit-testing program module or component one by one, and testing each result and combination.

There are two types of incremental testing

Top-down

- testing form the top of the module hierarchy and work down to the bottom. Modules are added in descending hierarchical order.

Bottom-up

- testing from the bottom of the hierarchy and works up to the top. Modules are added in ascending hierarchical order.

Testing Levels/ Techniques	White Box	Black Box	Incremental
Unit Testing	X		
Integration Testing	X		X
System Testing		X	
Acceptance Testing		X	

Major Testing Types

Major Testing Types

- Stress / Load Testing
- Performance Testing
- Recovery Testing
- Conversion Testing
- Usability Testing
- Configuration Testing

Stress / Load Test

- Evaluates a system or component at or beyond the limits of its specified requirements.
- Determines the load under which it fails and how.

Performance Test

- Evaluate the compliance of a system or component with specified performance requirements.
- Often performed using an automated test tool to simulate large number of users.

Recovery Test

- Confirms that the system recovers from expected or unexpected events without loss of data or functionality.

  Eg.
  - Shortage of disk space
  - Unexpected loss of communication
  - Power out conditions

Conversion Test

- Testing of code that is used to convert data from existing systems for use in the newly replaced systems

Usability Test

- Testing the system for the users to learn and use the product.

Configuration Test

- Examines an application's requirements for pre-existing software, initial states and configuration in order to maintain proper functionality.

Test Plan

TEST PLAN

Objectives

- To create a set of testing tasks.
- Assign resources to each testing task.
- Estimate completion time for each testing task.
- Document testing standards.

- **A document that describes the**
  - scope
  - approach
  - resources
  - schedule

  …**of intended test activities**.

- **Identifies the**
  - test items
  - features to be tested
  - testing tasks
  - task allocation
  - risks requiring contingency planning.

Purpose of preparing a Test Plan

- Validate the acceptability of a software product.
- Help the people outside the test group to understand 'why' and 'how' of product validation.
- A Test Plan should be
  - thorough enough (Overall coverage of test to be conducted)
  - useful and understandable by the people inside and outside the test group.

Scope

- The areas to be tested by the QA team.
- Specify the areas which are out of scope (screens, database, mainframe processes etc).

Test Approach

- Details on how the testing is to be performed.
- Any specific strategy is to be followed for testing (including configuration management).

Entry Criteria

Various steps to be performed before the start of a test i.e. Pre-requisites.

E.g.

- Timely environment set up
- Starting the web server/app server
- Successful implementation of the latest build etc.

Resources

List of the people involved in the project and their designation etc.

Tasks/Responsibilities

- Tasks to be performed and responsibilities assigned to the various team members.

Exit Criteria

- Contains tasks like
  - ✓Bringing down the system / server
  - ✓Restoring system to pre-test environment
  - ✓Database refresh etc.

Schedule / Milestones

- Deals with the final delivery date and the various milestones dates.

Hardware / Software Requirements

- ➢Details of PC's / servers required to install the application or perform the testing
- ➢Specific software to get the application running or to connect to the database etc.

Risks & Mitigation Plans

- ➢List out the possible risks during testing
- ➢Mitigation plans to implement incase the risk actually turns into a reality.

Tools to be used

- List the testing tools or utilities
- Eg. WinRunner, LoadRunner, Test Director, Rational Robot, QTP.

Deliverables

- Various deliverables due to the client at various points of time i.e. Daily / weekly / start of the project end of the project etc.
- These include test plans, test procedures, test metric, status reports, test scripts etc.

Sign-off

- Mutual agreement between the client and the QA Team.
- Both leads/managers signing their agreement on the Test Plan.

Good Test Plans

- Developed and Reviewed early.
- Clear, Complete and Specific
- Specifies tangible deliverables that can be inspected.
- Staff knows what to expect and when to expect it.

Good Test Plans

- Realistic quality levels for goals
- Includes time for planning
- Can be monitored and updated
- Includes user responsibilities
- Based on past experience
- Recognizes learning curves

Test Cases

TEST CASES

Test case is defined as

- A set of test inputs, execution conditions and expected results, developed for a particular objective.
- Documentation specifying inputs, predicted results and a set of execution conditions for a test item.

- Specific inputs that will be tried and the procedures that will be followed when the software tested.
- Sequence of one or more subtests executed as a sequence as the outcome and/or final state of one subtests is the input and/or initial state of the next.
- Specifies the pretest state of the AUT and its environment, the test inputs or conditions.
- The expected result specifies what the AUT should produce from the test inputs.

Test Cases

Contents

- Test plan reference id
- Test case
- Test condition
- Expected behavior

Good Test Cases

Find Defects

- Have high probability of finding a new defect.
- Unambiguous tangible result that can be inspected.
- Repeatable and predictable.

Good Test Cases

- Traceable to requirements or design documents
- Push systems to its limits
- Execution and tracking can be automated
- Do not mislead
- Feasible

Defect Life Cycle

Defect Life Cycle

What is Defect?

- A defect is a variance from a desired product attribute.

Two categories of defects are

- Variance from product specifications
- Variance from Customer/User expectations

Variance from product specification

- Product built varies from the product specified.

Variance from Customer/User specification

- A specification by the user not in the built product, but something not specified has been included.

Defect categories

Wrong

- The specifications have been implemented incorrectly.

Missing

- A specified requirement is not in the built product.

Extra

- A requirement incorporated into the product that was not specified.

Defect Log

1. Defect ID number
2. Descriptive defect name and type
3. Source of defect – test case or other source
4. Defect severity
5. Defect Priority
6. Defect status (e.g. New, open, fixed, closed, reopen, reject)

Defect Log (Cont..)

7. Date and time tracking for either the most recent status change, or for each change in the status.
8. Detailed description, including the steps necessary to reproduce the defect.
9. Component or program where defect was found
10. Screen prints, logs, etc. that will aid the developer in resolution process.
11. Stage of origination.
12. Person assigned to research and/or corrects the defect.

Severity Vs Priority

Severity

Factor that shows how bad the defect is and the impact it has on the product

Priority

Based upon input from users regarding which defects are most important to them, and be fixed first.

Severity Levels

- Critical
- Major / High
- Average / Medium
- Minor / low
- Cosmetic defects

Severity Level – Critical

- An installation process which does not load a component.
- A missing menu option.
- Security permission required to access a function under test.
- Functionality does not permit for further testing.

- Runtime Errors like JavaScript errors etc.
- Functionality Missed out / Incorrect Implementation (Major Deviation from Requirements).
- Performance Issues (If specified by Client).
- Browser incompatibility and Operating systems incompatibility issues depending on the impact of error.
- Dead Links.

Severity Level – Major / High

- Reboot the system.
- The wrong field being updated.
- An updated operation that fails to complete.
- Performance Issues (If not specified by Client).
- Mandatory Validations for Mandatory Fields.

- Functionality incorrectly implemented (Minor Deviation from Requirements).
- Images, Graphics missing which hinders functionality.
- Front End / Home Page Alignment issues.
- Severity Level – Average / Medium
  Incorrect/missing hot key operation.

Severity Level – Minor / Low

- Misspelled or ungrammatical text
- Inappropriate or incorrect formatting (such as text font, size, alignment, color, etc.)
- Screen Layout Issues
- Spelling Mistakes / Grammatical Mistakes
- Documentation Errors

- Page Titles Missing
- Alt Text for Images
- Background Color for the Pages other than Home page
- Default Value missing for the fields required
- Cursor Set Focus and Tab Flow on the Page
- Images, Graphics missing, which does not, hinders functionality

Testing Standards

IEEE STANDARDS

Institute of Electrical and Electronics Engineers designed an entire set of standards for software and to be followed by the testers.

IEEE – Standard Glossary of Software Engineering Terminology

IEEE – Standard for Software Quality Assurance Plan

IEEE – Standard for Software Configuration Management Plan

IEEE – Standard for Software for Software Test Documentation

IEEE – Recommended Practice for Software Requirement Spec.

IEEE – Standard for Software Unit Testing

IEEE – Standard for Software Verification and Validation

IEEE – Standard for Software Reviews

IEEE – Recommended practice for Software Design descriptions

IEEE – Standard Classification for Software Anomalies

IEEE – Standard for Software Productivity metrics

IEEE – Standard for Software Project Management plans

IEEE – Standard for Software Management

IEEE – Standard for Software Quality Metrics Methodology

Other standards…..

ISO – International Organization for Standards

Six Sigma – Zero Defect Orientation

SPICE – Software Process Improvement and Capability Determination

NIST – National Institute of Standards and Technology