Test Planning and Design

Test Planning and Test Design are two critical stages in the software testing lifecycle.

They help ensure that testing is thorough, organized, and aligned with project goals.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Test Planning

What is Test Planning?

Test planning is the process of defining the scope, objectives, approach, and resources required for testing activities. It results in a Test Plan document, which acts as a blueprint for all testing efforts.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Key Components of a Test Plan:

Section Description

Test Objectives What is being tested and why.

Test Scope What will and won’t be tested.

Test Strategy Overall testing approach (manual, automated, black box, etc.).

Test Environment Hardware, software, and network setups required.

Test Schedule Timeline for each test activity.

Resources & Roles Who will perform what tasks (testers, developers, tools).

Entry & Exit Criteria Conditions to start and stop testing.

Risk Management Potential issues and mitigation strategies.

Deliverables What will be produced (test cases, bug reports, metrics).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example:

In a banking app project, the test plan may state:

"Functional testing will be performed on the login, account summary, and fund transfer modules using a black-box approach. Testing will begin on Sept 1 and end by Sept 20. Selenium will be used for automation."

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Test Design

What is Test Design?

Test design is the process of creating detailed test cases and test data based on the requirements and specifications defined in the planning phase.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Key Activities in Test Design:

Activity Description

Requirement Analysis Understand what needs to be tested.

Test Scenario Identification High-level descriptions of what to test.

Test Case Design Detailed steps, inputs, expected outcomes.

Test Data Creation Input values and environmental data required for tests.

Traceability Matrix Mapping test cases to requirements for full coverage.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example of a Test Case:

Field Example

Test Case ID TC\_Login\_001

Description Verify login with valid credentials

Steps 1. Open login page

2. Enter valid username and password

3. Click login

Test Data Username: user123

Password: pwd123

Expected Result User is redirected to the dashboard

Actual Result (Filled during testing)

Status Pass(Green Bar)/Fail(Red Bar)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Summary: Test Planning vs. Test Design

Aspect Test Planning Test Design

Goal Define what and how to test Create detailed test cases

Focus Strategy, resources, scheduling Test scenarios, steps, test data

Output Test Plan document Test cases, test scripts, test data

Performed By Test Manager / Lead QA Engineers / Testers

Test Execution in Software Testing

Test Execution is the phase where the actual testing takes place test cases are run, results are recorded, and any deviations (bugs or defects) are logged for fixing.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What is Test Execution?

Test execution is the process of running the test cases designed during the test design phase and comparing actual results with expected results to determine if the software behaves as intended.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Key Activities in Test Execution

Step Description

1. Test Environment Setup Make sure the hardware/software environment is ready (e.g., test server, database, network, external jar files to be imported).

2. Test Case Execution Run manual or automated test cases.

3. Record Results Log actual results and compare with expected outcomes.

4. Defect Reporting If the test fails, log a defect/bug in a tracking tool (e.g., Jira, Bugzilla).

5. Retesting and Regression Testing Once a defect is fixed, re-run test cases and check related functionality.

6. Test Status Reporting Track progress (e.g., % passed, failed, blocked) and report to stakeholders.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example:

Test Case ID Description Expected Result Actual Result Status Defect ID

TC\_Login\_001 Valid login credentials Redirect to dashboard Redirect to dashboard Pass –

TC\_Login\_002 Invalid password Show error message Login page reloads Fail BUG\_105

TC\_Logout\_003 Click logout button Redirect to login screen Redirect to login screen Pass –

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tools Commonly Used in Test Execution

Type Tools

Test Management TestRail, Zephyr, Xray, HP ALM

Bug Tracking Jira, Bugzilla, Mantis

Automation Selenium, Cypress, JUnit, TestNG

CI/CD Execution Jenkins, GitLab CI/CD

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Test Execution Status Terms

Status Meaning

Pass Test ran successfully and matched expectations.

Fail Test ran but didn’t meet the expected result.

Blocked Test couldn’t be executed due to an external issue (e.g., network down).

Not Run Test case wasn’t executed yet.

Skipped Test was intentionally skipped (e.g., not relevant for this cycle).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Best Practices for Test Execution

• Prioritize critical test cases first.

• Log defects clearly with steps, screenshots, and logs.

• Communicate regularly with the development team.

• Continuously update test status and reports.

• Track defect fixes and retest promptly.

Defect Management in Software Testing

Defect Management is the structured process of identifying, documenting, prioritizing, tracking, and resolving defects (bugs) found during the software development and testing life cycle.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What Is a Defect (Bug)?

A defect is a flaw or error in a software product that causes it to behave incorrectly or unexpectedly, not meeting the requirements or specifications.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Defect Management Process Steps

Defect Life Cycle (also called the Bug Life Cycle):

Step Description

1. Defect Detection A tester or user finds unexpected behavior in the software.

2. Defect Logging The issue is logged in a bug tracking tool (e.g., Jira, Bugzilla) with details.

3. Defect Triage Team analyzes, validates, and assigns priority and severity.

4. Defect Assignment The bug is assigned to a developer to fix.

5. Defect Fixing The developer investigates and resolves the issue.

6. Retesting The tester re-executes the failed test cases to verify the fix.

7. Closure If the fix works, the defect is marked Closed. If not, it's Reopened.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Defect Life Cycle (Status Flow)

New → Assigned → Open → Fixed → Retest → Verified → Closed

↘ Reopen ↖

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Key Defect Attributes

Attribute Description

Defect ID Unique identifier

Title Short summary

Description Detailed explanation with steps to reproduce

Severity Impact on system (e.g., Critical, Major, Minor)

Priority Urgency to fix (e.g., High, Medium, Low)

Status Current state (e.g., New, Open, Closed)

Environment Where it occurred (e.g., Browser, OS)

Attachments Screenshots, logs, video evidence

-----

Unit Testing 🡪 System Testing

Moving from unit testing to system testing in the software testing lifecycle involves a structured progression through different testing levels. Each level increases in scope and complexity.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Unit Testing

• Goal: Test individual units (functions/methods) of code in isolation.

• Who performs it: Developers.

• Tools: JUnit, NUnit, PyTest, etc.

• Key Focus: Correctness of logic, inputs/outputs, edge cases.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Integration Testing

• Goal: Verify interactions between units/modules.

• Types:

o Top-down

o Bottom-up

o Big bang

o Incremental

• Who performs it: Developers or testers.

• Tools: TestNG, JUnit (with mocks), Postman (for APIs), etc.

• Key Focus: Data flow, interfaces, communication between modules.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. System Testing

• Goal: Validate the complete and fully integrated application against requirements.

• Who performs it: QA/Testers.

• Environment: Close to production.

• Types of System Testing:

o Functional Testing

o Non-functional Testing (Performance, Security, Usability)

• Tools: Selenium, JMeter, TestRail, etc.

• Key Focus: End-to-end workflows, user scenarios, overall behavior.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Steps in Transition:

Step Description

1. Complete Unit Testing Ensure all functions/classes are tested with high code coverage.

2. Set Up Continuous Integration (CI) Automate running unit tests on each commit or build.

3. Identify Integration Points Understand where modules interact; define interfaces.

4. Conduct Integration Testing Test data exchange, API calls, database interactions, etc.

5. Prepare Test Environment Set up system test environment resembling production.

6. Review Requirements & Specs Use functional requirements to derive system test cases.

7. Create System Test Plans & Scripts Include end-to-end test cases, performance checks, etc.

8. Perform System Testing Execute full application tests covering all features and workflows.

9. Log and Track Bugs Report and retest defects found during system testing.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Summary Flow:

Unit Testing → Integration Testing → System Testing

↓ ↓ ↓

Test Code Test Module Test Entire App

In Isolation Interactions End-to-En