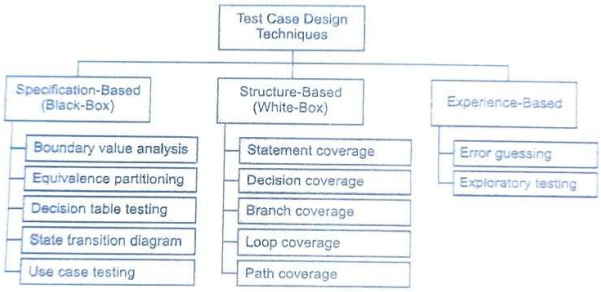
Chapter 1: Introduction to Test Case Design

- **Software testing**: software testing is the process of executing software or system with Intent of finding / detecting errors.
    - The tools used for software testing are known as software testing tools.
- **Test case**: test case is used to determine whether a software application is working as per customer's requirement or not.
    - It provides the description of inputs and their expected outputs.
    - It is considered to be reliable if it detects all errors.
    - It is considered as valid, if at least one test case reveals the errors.

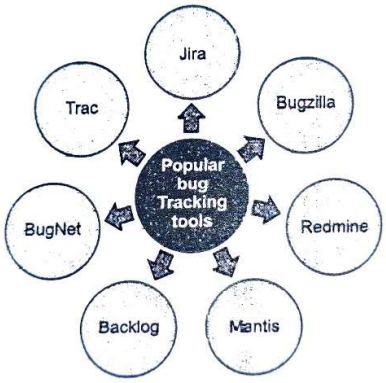


- **Testing tool**: testing tool is software product that enables Software testing tasks.
    - It provides great speed, reliability and efficiency.

Identification of Errors and Bugs in the given Application

- Software testing includes identification of error & bugs in software application.
- Error is an incorrect human action that produces an incorrect result.
- **Error Types**:
    1. **Syntax error**: It occurs in the source code of the program and prevents the program from being properly compiled.
    2. **Logical error**: It represents a mistake in software flow and causes software to behave incorrectly.
- **Bug**: It is an error, flaw, failure or fault in a computer program or system, that causes it to produce unexpected output.
- **Bug Types**:
    1. **Functional bugs**: Functional bugs are associated with functionality of specific software components.
    2. **Logical bugs**: Logical bugs descripts the intended workflow of software & cause it to behave incorrectly.
- **Reasons for arising a Bug**:
    1. **Wrong coding**: Wrong coding means improper implementation.
    2. **Missing coding**: Missing coding means, we can say that developer won't have developed the code only for that specific part of software.
    3. **Extra coding**: Extra coding means the developers add some extra features in the system, which is not needed as per requirements by client.

Common bug tracking tools



- **Jira**:
  - Most important open-source bug tracking, project management and issue tracking tool in manual testing. Comprises with different features like recording, reporting and workflow.
  - Features:
    - Provides complete set of reporting of the bug tracking.
    - Supports integration with development tools such as GitHub.
    - Workflow-based tool.
    - Used to manage the defects very effectively.
- **Bugzilla**:
  - Open-source bug tracking tool used by many organizations. It is used to assist customers and clients to keep track of the buts. It is also used as a test management tool.
  - Features:
    - Bug can be list in multiple formats.
    - Advanced searching capabilities.
    - Excellent security.
    - Time tracking.
- **Redmine**:
  - Open-source and web-based project management tool used to track the issues. Written in Ruby and compatible with databases like MySQL, Microsoft SQL and SQLite.
  - Characteristics:
    - Flexible role-based access control.
    - Time tracking functionality.
    - Flexible issue tracking system.
    - Multiple languages support.
- **Mantis**:
  - MantisBT stands for Mantis Bug Tracker. It is open-source and web-based bug tracking tool. It is used to track the software defects. It is executed in the PHP.
  - Characteristics:
    - Full-text search.
    - Revision control system integration.
    - Notifications.
    - Plug-ins.

Design Entry & EXIT criteria for test case

- Entry criteria and exit criteria is the test case. determine when given test activity can. used to Start & Stop.
- Entry criteria for the test are the requirements that need to be fulfilled before the test case runs.
- Exit criteria for test cases are a set of conditions based on which we can determine that test case execution is finished.

Design test cases in Excel

- Test case data can be managed in Excel. This requires an Excel Spreadsheet which analyses the test data in rows and columns.
- Test designers can be used to design MS Excel and build the tests.
- Ms Excel functionalities are so powerful that we can write similar test cases in less time.

The common fields that are used in test case are:

1. Test Case ID
2. Test Case Name
3. Test Description
4. Step to Execute
5. Pre-Conditions
6. Execution Steps
7. Expected Results
8. Actual Results
9. Status
10. Comment

Features of Testing Method used

- Software testing methods are essential in building software.
- It develops to deal with different types of bugs.
- Testing methods like black-box testing, white-box testing applied to evaluate systems.
- Change in software that adds new functionality the existing the functionality is called Feature.

Several tests use for testing the software

1. High probability of detecting errors
2. No Redundancy
3. Choose most appropriate test
4. Moderate

Chapter 2: Test Cases for Simple Programs

- Types of test case design technique:
    1. **Specification-based test case design technique**: Used to design test cases based on an analysis of a model of the product without reference to its internal workings. Also known as the Black box test case design technique.
    2. **Structure-based test case design technique**: Used to design test cases based on an analysis of the internal structure of the test item. Also known as the White box test case design technique.
    3. **Experience-based test case design technique**: Experienced people with technology and business background with the help of their previous experience with similar systems know what must have gone wrong or will go wrong and make test cases on the basis of their experience. Design these test cases which are very effective.

- **Code coverage**: the percentage of program statements that can be invoked during testing is called code coverage.
  Formula = $\frac{\text{Number of lines of the code exercised}}{\text{Total number of lines of code}} \times 100$

- Code coverage testing techniques:
    1. Statement coverage
    2. Decision coverage
    3. Branch coverage
    4. Loop coverage
    5. Path coverage

- **Statement coverage testing**:
    - The simplest form of white box testing in which series of test cases are run such that each statement is executed at least once.
    - Used coz unless a statement is executed it is hard to determine if an error exists in that statement.
    - Statement coverage = $\frac{\text{number of executed statements}}{\text{Total number of statements}} \times 100$
    - Advantages:
        - Measures the quality of code.
        - Ensures every statement is tested.
    - Disadvantages:
        - Cannot test false conditions.
        - Does not understand logical operators.

- **Decision coverage testing**:
    - A white box testing technique which reports true or false outcomes of a Boolean expression.

- Covers all possible outcomes by each and every Boolean condition by using a control flow graph.

- decision coverage = $\frac{\text{number of decision outcomes exercised}}{\text{Total number of outcomes}} \times 100$
- Advantages:
  - Eliminate problems.
  - Where all branches are reached.
- Disadvantages:
  - Ignore branches under Boolean expression.

- **Branch coverage testing**:
  - Improvement over statement coverage testing.
  - Also known as edge coverage testing.
  - It is a white box testing that ensures that each decision condition from every branch is executed at least once.

Decision Coverage	Branch Coverage
Measures the coverage of conditional branches.	Measures coverage of conditional and unconditional branches.

  - branch coverage = $\frac{\text{number of executed branches}}{\text{total number of branches}} \times 100$
  - Advantages:
    - Validate all branches.
    - Remove tissues of statement coverage.
  - Disadvantages:
    - Costly
    - Time-consuming.

- **Loop coverage testing**:
  - White box testing checks whether the loop body executed 0 times, exactly one or more times.

1. Simple loop testing: while, do-while loops check entry exit conditions.
2. Nested loop: for, while, do-while loops inside loops.
3. Concatenated loop: series of loops or loop after loop.
4. Unstructured loop: a combination of nested and concatenated loop.

- **Path coverage testing**:
  - Design test cases such that all linearly independent paths in the program are executed at least once.
  - The linearly independent path can be shown using the Control flow graph.
  - Path: It is a node and edge sequence from starting node to a terminal node of the control flow graph of the program code.
  - Linearly independent path: Is any path through the program that introduces at least one new edge that is not included in any other linearly independent path.

- Mc Cabe's Cyclometric Complexity: Quantitative measure of independent paths in a source code.
- Denoted as V(g)
  V(g) = E -N+2 (E= edge ,N= node)

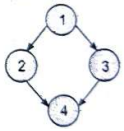
- Bounded area: Any region enclosed by nodes and edges is called bounded area.
  V(g) = no of bounded area + 1
  path coverage = $\frac{\text{total number path exercise}}{\text{the total number of the path in program}} \times 100$
- Node: The different number statements serve as a node.
- Edge: If the execution of the statement of a node can result in control transfer to another node.

Sequence:

```
1. a=5;
2. b=a*b-1;
```

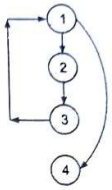
Selection:

```
1. if(a>b)then
2.     c=3;
3.  else c=5;
4.  c=c*c;
```



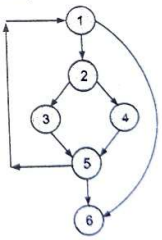
Iteration:

```
1. while(a>b){
2.      b=b*a;
3.      b=b-1;}
4.  c=b+d;
```



Example for CFG (Euclid's gcd Algorithm):

```
int f1(int x, int y){
1. while(x != y){
2.     if(x>y)then
3.     x=x-y;
4.  else y=y=x;
5. }
6. return x; }
```



Chapter 3: Test Cases and Test Plan

Test Case Definition:

- A test case in software testing is a document, which has a set of test data predictions, expected results and post-conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Plan:

- A test plan is a detailed document that outlines
    - **test strategy**
    - **objectives**
    - **test schedule**
    - required resources like **human resources**, **software resources** and **hardware resources**, **test estimation** and **test deliverables**.

- Determines the effort needed to validate the quality of application under test.
- Test plan serves as a blueprint to conduct software testing activities.

Creating a Test Plan:

1. **Product Analysis**
2. **Designing Test Strategy**
3. **Defining Objectives**
4. **Establish Test Criteria**
5. **Planning Resource Allocation**
6. **Planning Resource Allocation**
7. **Planning Setup of Test Env.**
8. **Determining Test Schedule and Estimation**
9. **Establish Test Deliverables**

1. **Product Analysis:**
    a. We need to have a deep understanding of the product or feature before we create a test plan.
    b. To understand the scope, objectives and functionality of the product, talking with the designer and developer helps.
        i. Review the product documentation, Scope of Work, project proposal or even the tasks in your project management tool.
        ii. Perform a product walkthrough to understand functionality, user flow, limitations.

2. **Designing Test Strategy:**
    Project obj. And how to achieve them.
    The amount of effort and cost required for testing.
    a. Test mgr. Develops the Test Strategy document which defines:
        i. **Scope of Testing:** Contains components that will and will not be tested. (hw, sw and mw)

ii. **Type of Testing:** Describes types of tests, necessary because each test identifies specific types of bugs.

iii. **Risks and Issues:** Describes all possible risks that may occur during testing: tight deadlines, insufficient management, inadequate or erroneous budget estimate - as well as the effect of these risks on the product or businesses.

iv. **Test Logistics:** Mentions the names of testers (or their skills) as well as tests to run by them. Also includes the schedule laid out for testing and required tools.

3. Defining Objectives:

- Defining the goals and expected results of the test execution. All testing intends to identify maximum defects in an application.
- Must include:
    - A list of software features - functionality, GUI, performance standards
    - Ideal result / benchmark. Actual results are compared to expected results.

4. Establishing Test Criteria:

- Denotes standards
- Two main test criteria:
    - Suspension Criteria: Type of Testing: Describes types of tests, necessary because each test identifies specific types of bugs.
    - Exit Criteria: Defines the benchmarks that signify completion of success in a test phase or project. The exit criteria are the expected results of tests and must be met before moving on to the next stage of development.

5. Planning Resource Allocation:

- Includes all type of resources required and their breakdown
- Includes human effort, equipment, infrastructures, etc.
- Helps managers to calculate schedule and estimates.

6. Planning Setup of Test Env.:

- SW and HW setup on which software testing team runs their tests.
- Real devices are used ideally.

7. Establish Test Derivable:

- Implies list of documents, tools and equipment needed for the testing activities.
- List of Deliverables:
    - Deliverables required before testing: includes test plan and test design
    - Deliverables required during testing: test scripts, simulators or emulators, test data, error and execution logs.
    - Deliverables required after testing: test results, defect reports and release notes.

Test Plan Template

- Fig. 3.2 shows typical test plan template used in software testing.

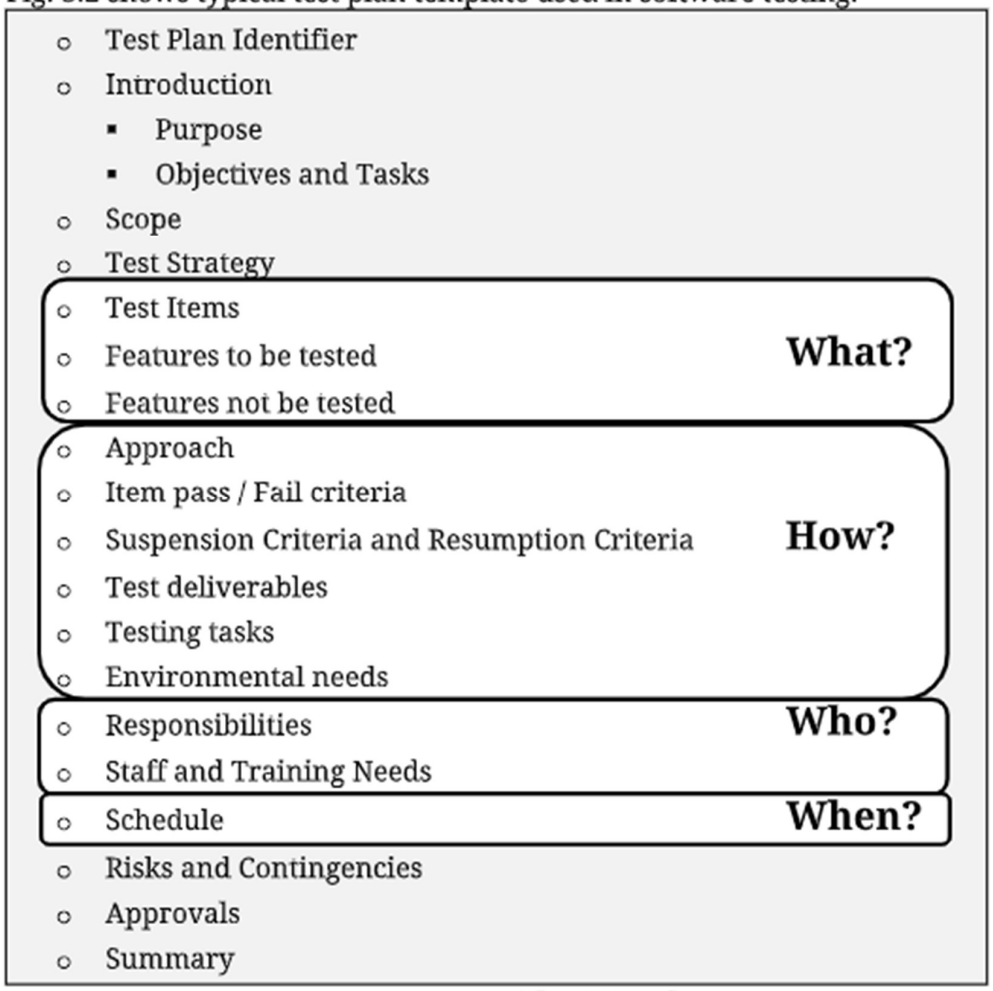


Fig. 3.2: Test Plan Template

- Document that outlines test strategies, goals, timetables, estimates, deliverables and resources needed for testing.
- Helps determine how much work is needed to confirm quality of application.

Definitions:

1. **Test Plan Identifiers**: identifies test plans; includes versions and release numbers.
2. **Introduction**: intro about the project
3. **Purpose**: purpose of the test plan
4. **Objectives and Tasks**: describes goals of project, resource factors
5. **Scope**: functional / non-functional requirements are tested.
6. **Test Strategies**: overall approach
7. **Test Items**: application / feature under the test
8. **Software Risk Issues**: project risk and assumptions
9. **Features to be Tested**
10. **Features not to be tested**
11. **Approach**: details about overall testing
12. **Item Pass/Fail Criteria**
13. **Suspension/Resumption Criteria**: Suspension and Resumption Criteria

14. **Test Deliverables**: delivered as a part of the test process. Ex: Test Process, Plans, Specifications and Summary Reports.
15. **Remaining Test Tasks**
16. **Environment Needs**: HW, SW, OS, Network Config., Tools
17. **Staffing and Training Needs**
18. **Responsibilities**: Lists roles and responsibilities of the team members.
19. **Schedule**: important dates and milestones
20. **Planning Risks and Contingencies**
21. **Approvals**: Captures all approvers of documents, their titles, signs, dates.
22. **Glossary**: Summary or definitions of terms used in testing.

Writing Test Case for Given Application:

Parameter	Description
Test Case ID	Each test case should be represented by a unique ID. It's good practice to follow some naming convention for better understanding and discrimination purposes.
Test Case Description	Pick test cases properly from the test scenarios.
Pre-conditions	Conditions that need to meet before executing the test case. Mention if any preconditions are available.
Test Steps	To execute test cases, you need to perform some actions. So write proper test steps. Mention all the test steps in detail and in the order how it could be executed from the end-user's perspective.
Test Data	You need proper test data to execute the test steps. So gather appropriate test data. The data which could be used an input for the test cases.
Expected Result	The result which we expect once the test cases were executed. It might be anything such as Home Page, Relevant screen, Error message, etc.,
Post-Conditions	Conditions that need to achieve when the test case was successfully executed.
Actual Result	The result which system shows once the test case was executed. Capture the result after the execution. Based on this result and the expected result, we set the status of the test case.
Status	Finally set the status as Pass or Fail based on the expected result against the actual result. If the actual and expected results are the same, mention it as Passed. Else make it as Failed. If a test fails, it has to go through the bug life cycle to be fixed.
Comments	If there are any special conditions to support the above fields, which can't be described above or if there are any questions related to expected or actual results then mention them here.

(Refer Examples for better understanding)

Test Case Template

Test Case Template 1:

Project Name - Module name - Created by - Creation Date - Reviewed by - Reviewed Date -											Company Logo
Test Scenario ID and Description	**Test Case ID**	**Test Case Description**	**Pre-conditions**	**Test Data**	**Post-conditions**	**Expected Result**	**Actual Result**	**Status**	**Test Case Executed By**	**Executed Date**	**Comments (if any)**

Test Case Template 2:

Your Company LOGO	**Project Name:**		**Test Designed by:**	
	Module Name:		**Test Designed date:**	
	Release Version:		**Test Executed by:**	
			Test Execution date:	
Pre-condition				
Dependencies:				
Test Priority				
Test Case#	**Test Title**	**Test Summary**	**Test Steps**	**Test Data**

Test Case Template 3:

TEST CASE TEMPLATE				
Title		**Req ID**		
Author		**Test Case ID**		
Date		**Test Type**		
Objective	**Objective Description**			
#	Enter context of Objective # here			
#	Enter context of Objective # here			
#	Enter context of Objective # here			
#	Enter context of Objective # here			
#	Enter context of Objective # here			
Test Objective #	Enter Test Objective			
Test Input	Enter Input for test			
Test Script Step(s)	Enter Test Script Step(s) here			
Expected Results	Enter Expected Results of test			
Actual Results	Enter Actual Results of test			
Data Location	Enter Location of Test Data here			
	Name	**Test Date**	**Test Status**	**Defect No.**
Tested By				
Reviewed By				
Approved By				
Test Objective #	Enter Test Objective			
Test Input	Enter input for test			
Test Script Step(s)	Enter Test Script Step(s) here			
Expected Results	Enter Expected Results of test			
Actual Results	Enter Actual Results of test			
Data Location	Enter Location of Test Data here			
	Name	**Test Date**	**Test Status**	**Defect No.**
Tested By				
Reviewed By				
Approved By				

(Refer examples for better understanding)

Writing Test Case

Steps for writing Test Cases:

Step 1: Review the Requirements
Step 2: Write a Test Plan
Step 3: Identify the Test Suite
Step 4: Name the Test Cases
Step 5: Write Test Case Description and Objectives
Step 6: Create the Test Cases
Step 7: Review the Test Cases
(insert General Theory into these points, read once from the textbook)

Preparing Test Cases for the Tests Executed

- Test reporting is essential for making sure a software application is achieving an acceptable level of quality.
- A test report is any description, explanation or justification of the status of a test project.
- A comprehensive test report is all of those things together.
- Testers need to demonstrate the ability to develop testing status reports. These reports should show the status of the testing based on the test plan.
- Reporting should document what tests have been performed and the status of those tests.

Test Report Template is given below:

Module	Scenarios	Sub Levels	Complexity	Tester	Execution Date	Status	Defect_Id	Severity

(Refer one example for better understanding of what data to fill in the columns)

Test Summary Report

IEEE Std. 829-1998 for Software Test Documentation
Template for Test Summary Report

Contents

1. Test Summary Report Identifier
2. Summary
3. Variances
4. Comprehensive Assessment
5. Summary of Results
   - 5.1 Resolved Incidents
   - 5.2 Unresolved Incidents
6. Evaluation
7. Recommendations
8. Summary of Activities
9. Approvals

Fig. 3.4: IEEE Std. Test Summary Report Template

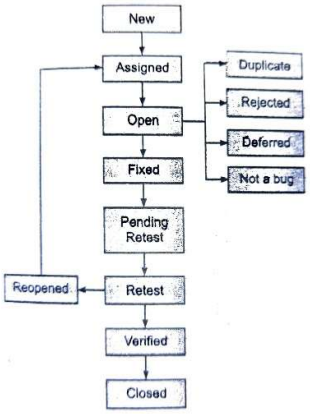
(Refer Flipkart Login Page Example)

Chapter 4: Defect Report

- A software defect arises when the expected result doesn't match with the actual results.
- Defect is defined as "an inconsistency in the behaviour of the software."
- A defect is an error or a bug in the application.
- Root causes of defects are:
    - Requirements are not defined clearly.
    - Designs are wrong and not implemented properly.
    - People are not trained for work in collecting requirements.
    - Processes used for product development and testing are not capable.

Defect Life Cycle

- Also known as Bug Life Cycle.
- Life cycle starts as soon as any new defect is found by a tester.



Defect Classification

1. **Requirement Defects**: These defects arise in a product when one fails to understand what is required by the customer. These may arise when the customer is unbale to define his requirements.
    a. Functional Defects
    b. Interface Defects
2. **Design Defects**: These defects generally refer to the way of design creation or its usage while creating a software product.
    a. Algorithmic Defects
    b. Module Interface Defects
    c. User Interface Defects
    d. System Interface Defects
3. **Coding Defects**: These defects may arise when designs are implemented wrongly. If there is absence of development/coding standards then it may lead to coding defects.
    a. Variable Declaration/Initialization Defects
    b. Commenting/Documentation Defects
    c. Database Related Defects
4. **Testing Defects**: These defects are introduced in a
    a. Test Design Defects
    b. Test Tool Defects
    c. Test environment Defects

Write Defect Report

- Defects reports provide information about defects that are related to a test plan, the status of those defects, and trends in those defects over time.
- Defects are founded by testers and written in defect report which is also known as bug report.
- It describes defects in software and documentation. It has a priority rating that indicates the impact the problem has on the customer.
- The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.
- **Defect Report Attributes:**
    1. Defect ID
    2. Defect Name
    3. Project Name
    4. Module/Sub-module Name
        a. Login Page
        b. Home Page
        c. Settings
    5. Phase Introduced
    6. Phase Found
    7. Defect Type
    8. Severity
        a. S1 = Critical
        b. S2 = Major
        c. S3 = Minor
        d. S4 = Trivial
    9. Priority
        a. P1 = High
        b. P2 = Medium
        c. P3 = Low
    10. Summary
    11. Defect Description
    12. Status of a Defect
        a. New
        b. Open
        c. Fixed
        d. Close
        e. Rejected
        f. Can't Reproduce
    13. Reported By/Reported On
        a. Test Engineer
        b. Test Lead
        c. Developer
    14. Version
    15. Date Raised
    16. Detected By
    17. Fixed By
    18. Date Closed

Chapter 5: Testing Tools

- Manual testing is performed by execution of test cases, compares actual test results and expected results.
- It is the oldest and most rigorous type of testing.
- Manual testing is a testing process that is carried out manually in order to find defects without the usage of automation tools or automation scripting.
- **Limitations of Manual Testing:**
    1. Manual testing is slow and costly.
    2. They do not scale well.
    3. It is not consistent or repeatable.
    4. Lack of training is a common problem.
    5. Testing is difficult to manage.
- Automation testing is a process of changing any manual test case into the test scripts by using automation testing tools and scripting or programming language.
- We could define automation as the technique of performing tasks without human intervention.
- **Benefits of Automated Testing:**
    1. Faster feedback cycle
    2. Increased efficiency
    3. Reliable
    4. Repeatable
    5. Programmable
    6. Comprehensive
    7. Reusable
    8. Better quality software
    9. Fast
    10. Cost reduction
    11. Improved accuracy
    12. Higher test coverage

Use of Testing Tools

- **There are two types of testing tools:**
    1. Static Test Tools: They do not test the actual execution of the software. These tools include coverage analysers, flow analysers, flow tests and so on.
    2. Dynamic Test Tools: They test the software system with “live” data. Dynamic test tools include Mutation analysers, emulators, test drivers, test beds and so on.
- **Usage of Automation Testing Tools commonly applied steps:**
    1. Select test tool
    2. Define scope of automation
    3. Planning, Designing and Development of test
    4. Execution of test
    5. Maintenance
- **Characteristics of Test Tools:**
    1. Easy to use.
    2. Provide complete code coverage.
    3. Use one or more testing strategy.
    4. Provide a clear and correct report on test case.
    5. Support GUI-based test preparation.
    6. Able to adopt the underlying hardware.

- **Test Automation Framework**
    - A test framework is a set of guidelines which can be followed to create test cases and related processes.
    - A test framework usually contains internal libraries and reusable code modules which provide a foundation for test automation.
- **Four types of test automation frameworks**
    1. Data driven automation framework: Test data and test scripts are separated which makes it easier to maintain and update the test data at any point of time without affecting the test scripts.
    2. Keyword driven automation framework: The test logic is divided into keywords and functions. A sequence of keywords is used to define the test scripts and these keywords are further defined as functions to implement the desired behaviour.
    3. Modular automation framework: The application can be divided into different modules which can be tested independently.
    4. Hybrid automation framework: More than one framework is used to achieve the desired objectives.
- **Test Automation Frameworks Benefits:**
    1. Flexibility to run selective test cases
    2. Reduced manual interactions
    3. Efficiency
    4. Easy maintenance

Types of Testing Tools

- Tools from a software testing context is a product that supports one or more test activities right form planning, requirements, creating a build, test execution, defect logging and test analysis.
- **Testing Tools Parameters:**
    1. Purpose of the tools.
    2. Activities that are supported within the tool.
    3. Type/level of testing.
    4. Kind of licensing.
    5. Technology used.
- **Popular Open-Source Automation Testing Tools:**

1. Katalon Studio
    - It is test automation tool that enables us to test web, apps, mobile app and desktop APIs.
    - It is powerful in enabling cross-functional operations for product development team at scale.
    - It is easy to use, robust to expand, yet contains the necessary component for advanced needs with built-in keywords and project templates.
2. Appium
    - It is an open-source test automation framework for mobile apps.
    - It is built on client/server architecture.
    - It automates the applications that are created for iOS and Android.
    - It is a well-liked mobile automation testing tool.

3. Robotium
    - It is an open-source tool that acts as a test automation framework which is mainly intended for Android UI testing.
    - It supports graybox UI testing, system testing, functional testing and user acceptance testing for both native and hybrid android-based applications.
4. Marathon
    - It is an open-source test automation framework, test Java-based GUI applications.
    - Mainly intended for acceptance testing, allows us to record and replay the tests and generates test reports as well.
    - Use marathon if we are testing a small project and if your application screen size is limited to 10 screens.

Selenium Tool

- It is a popular open-source software testing tool. It allows user to write scripts in lot of different languages, including Java, C#, Python, Perl and Ruby.
- It also runs in several OS such as Windows, Linux, Solaris, Macintosh and also supports iOS, Windows mobile and Android and browsers like Google Chrome, Mozilla Firefox, Internet Explorer, Safari and so on.
- Selenium IDE:
    - It is a completed IDE framework for selenium tests. It allows for recording, editing and debugging of tests.
    - It is a Chrome and Firefox plugin that can log natural interactions in the browser and generate its code in C#, Java, Python and Ruby as well as Selenese.
    - It is a record/run tool that a test case developer uses to develop selenium test cases.
    - Allows user to create the test cases and test suites and edit it later as per their requirements.
- Selenium WebDriver:
    - It is a programming interface that can be used to create and execute test cases.
    - It allows us to test across all the major programming languages, browsers and OS.
- Selenium Grid:
    - It allows us to run multiple tests at the same time on machines.
- **Steps for using Selenium to test the application:**
    1. Create a new project through File > New > Java Project.
    2. Enter details:
        a. Project Name
        b. Location to save project
        c. Select an execution JRE
        d. Select layout project option
        e. Click on finish button
    3. In this step,
        a. Right click on the newly created project and
        b. Select New > Package, and name that package as “newpackage”
        c. Enter the name of the package
        d. Click on finish button

4. Create a new Java class by selecting New > Class and then name it as “MyClass” and enter following details:
    a. Name of the class
    b. Click on finish button.

5. Now set selenium .jar files into Java build path. It will open a pop-up window
    a. Right-click on “newproject” and select properties.
    b. On the properties dialog, click on “Java Build Path”.
    c. Click on the libraries tab, and then
    d. Click on “Add External JARs”.
6. Finally, click OK and we are done importin selenium libraires into our project.
    a. There are different types of browser drivers according to types of browsers.
    b. This driver is needed while testing the web application. So, download the driver for browser of our choice.
    c. Write the test script code and execute the application and observer the output.