T. Y. B. Sc.
COMPUTER SCIENCE
SEMESTER-VI

NEW SYLLABUS
CBCS PATTERN

SOFTWARE TESTING TOOLS

Ms. SMITA GHORPADE
Ms. KIRTI MORE

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

SOFTWARE TESTING TOOLS

For T.Y.B.Sc. Computer Science : Semester – VI

[Course Code CS 3610 : Credits – 2]

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Ms. Smita Ghorpade

M.C.S., M.Phil. (Comp. Sci.), SET

Assistant Professor, Department of Computer Science

M. V. P. Samaj's Commerce Management and Computer Science

(C.M.C.S.) College, Nashik

Ms. Kirti More

M.Sc. (Comp. Sci.), SET

Assistant Professor, Department of Computer Science

M. V. P. Samaj's Commerce Management and Computer Science

(C.M.C.S.) College, Nashik

Price ₹ 130.00

NIRALI PRAKASHAN

ADVANCEMENT OF KNOWLEDGE

N5952

SOFTWARE TESTING TOOLS **ISBN 978-93-5451-301-5**

First Edition : January 2022

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

DISTRIBUTION CENTRES

PUNE

Nirali Prakashan
(For orders outside Pune)
S. No. 28/27, Dhayari Narhe Road, Near Asian College
Pune 411041, Maharashtra
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

Nirali Prakashan
(For orders within Pune)
119, Budhwar Peth, Jogeshwari Mandir Lane
Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

MUMBAI

Nirali Prakashan
Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES

DELHI

Nirali Prakashan
Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

BENGALURU

Nirali Prakashan
Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

NAGPUR

Nirali Prakashan
Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhanshi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

KOLHAPUR

Nirali Prakashan
438/2, Bhosale Plaza, Ground Floor
Khasbag, Opp. Balgopal Talim
Kolhapur 416 012, Maharashtra
Mob : 9850046155
Email : kolhapur@niralibooks.com

JALGAON

Nirali Prakashan
34, V. V. Golani Market, Navi Peth,
Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395
Mob : 94234 91860
Email : jalgaon@niralibooks.com

SOLAPUR

Nirali Prakashan
R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

marketing@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks

Preface ...

We take an opportunity to present this Text Book on "**Software Testing Tools**" to the students of Third Year B.Sc. (Computer Science) Semester-VI as per the New Syllabus, June 2021.

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts. The book covers theory of Introduction to Test Case Design, Test Cases for Simple Programs, Test Cases and Test Plan, Defect Report and Testing Tools.

A special word of thank to Shri. Dineshbhai Furia, and Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Amar Salunkhe and Mrs. Prachi Sawant of M/s Nirali Prakashan for their excellent co-operation.

We also thank Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

Authors

Syllabus ...

1. **Introduction to Test Case Design** **(4 Lectures)**
   - How to Identify Errors, Bugs in the given Application.
   - Design Entry and Exit Criteria for Test Case, Design Test Cases in Excel.
   - Describe Feature of a Testing Method Used.
2. **Test Cases for Simple Programs** **(4 Lectures)**
   - Write Simple Programs Make use of Loops and Control Structures.
   - Write Test Cases for above Programs.
3. **Test Cases and Test Plan** **(4 Lectures)**
   - Write Test Plan for given Application with Resources required.
   - Write Test Case for given Application.
   - Prepare Test Report for Test Cases Executed.
4. **Defect Report** **(3 Lectures)**
   - Defect Life Cycle.
   - Classification of Defect.
   - Write Defect Report.
5. **Testing Tools** **(3 Lectures)**
   - How to make use of Automation Tools.
   - Types of Testing Tools.

Contents ...

CHAPTER 1

Introduction to Test Case Design

Objectives...

- To understand Test Case, Bug and Error
- To learn Design of Test Cases
- To study Design Entry and Exit Criteria of Test Cases
- To Design of Test Cases in Excel

1.0 INTRODUCTION

- A test case describes input, action and an expected result, in order to determine the functionality of a software application works correctly or not.
- A test case is a set of instructions on "how" to validate a particular test objective/target, which, when followed will tell us if the expected behavior of the system is satisfied or not.
- The purpose of a test case is to determine whether a software application is working as per the customer's requirements or not.
- A test case is a set of conditions and variables prepared to verify the compliance of an software application functionality against the specified requirements.
- A test case provides the description of inputs and their expected outputs to observe whether the software or a part of the software is working correctly.
- Institute of Electrical and Electronics Engineers (IEEE) defines test case as, "a set of input values, execution pre-conditions, expected results and execution post-conditions, developed for a particular objective or test condition such as to exercise a particular program path or to verify compliance with a specific requirement."
- Generally, a test case is associated with details of identifier, name, purpose, required inputs, test conditions and expected outputs.
- Software testing is the process of executing software or system with the intent of finding/detecting errors. The tools used for software testing are known as software testing tools.

- Test case designing includes preconditions, case name, input conditions, and expected result. Software testing tools are the tools which are used for the testing of software.
- Software testing tools often ensure that software products are robust, comprehensive and work according to requirements.
- A testing tool is a software product that enables software testers to define software testing tasks.
- Using a test tool an organization achieves greater speed, reliability and efficiency in their testing process.
- Testing tools support test activities such as requirements, planning, test execution, automation and defect logging.
- Tools from a software testing context can be defined as, a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.
- Demand for delivering better quality software products faster makes organizations search for test tools.
- Some popular software testing tools are JMeter, LoadRunner, Selenium, Appium, TestProject, Katalon, CloudTest, TestComplete AppliTools and so on.

1.1 BASICS FOR TEST CASE DESIGN

- A test case is a document which has a set of conditions or actions that are performed on the software application in order to verify the expected functionality of the feature requirements.
- A test case provides the description of inputs and their expected outcomes which is being observed to determine whether the software works correctly or not.
- Designing the test cases is the most challenging assignment of test engineers. Test cases have to be designed based on two criteria namely, reliability and validity.
  - A set of test cases is considered to be reliable if it detects all errors.
  - A set of test cases is considered as valid, if at least one test case reveals the errors.
- Test engineers' challenge lies in uncovering as many defects as possible with minimum number of test cases.
- A test case is a statement that defines what needs to be tested, how it will be tested, what is the precondition for that testing, and what is the expected output from that testing.
- Before we can write a test case, we need to think about how we should design the test cases to make the testing effective.
- Test case design techniques play an important role in software testing. Test case design techniques are standards of test designing that allow the creation of systematic and widely accepted test cases.

- An efficient test case design technique is necessary to improve the quality of the software testing process. It helps to improve the overall quality and effectiveness of the released software product.
- There are many ways to design test cases. The test case design techniques are shown in Fig. 1.1.

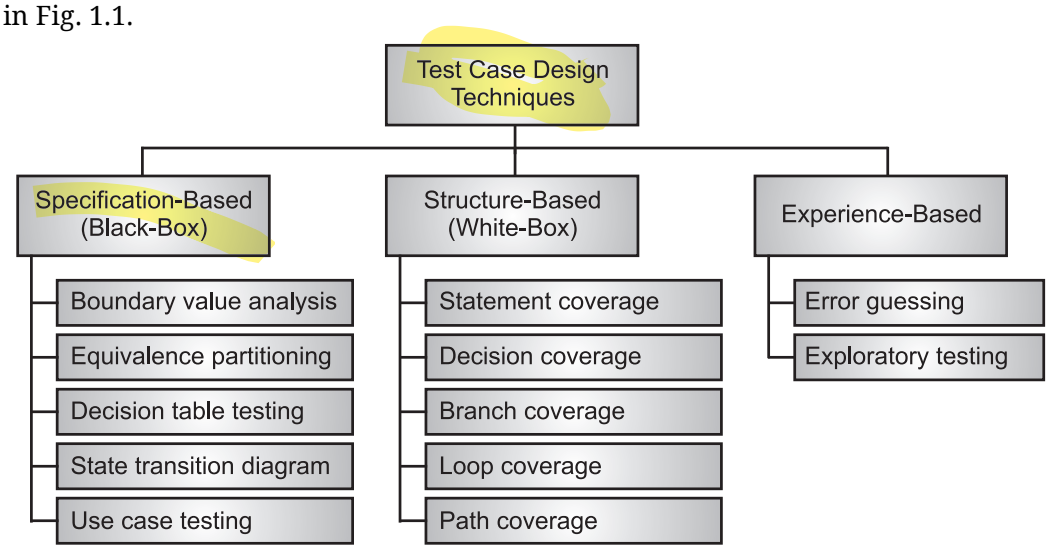


Fig. 1.1: Techniques for Test Case Design

- Test design is a process that describes "how" testing should be done. It includes processes for identifying test cases by enumerating steps of the defined test conditions.
- The test case design techniques are broadly classified into following three categories:

1. The test case design technique based on deriving test cases directly from a specification or a model of a system or proposed system, known as **specification-based** or **black-box techniques**. So black-box techniques are based on an analysis of the test basis documentation, including both functional and non-functional aspects. They do not use any information regarding the internal structure of the component or system under test.

   Specification-based test case design techniques can be used to design test cases in a systematic format. These use external features of the software such as technical specifications, design, client's requirements and more, to derive test cases. With this type of test case design techniques, testers can develop test cases that save testing time and allow full test coverage.

2. The test case design technique based on deriving test cases directly from the structure of a component or system, known as **structure-based** or **white-box techniques**. We will concentrate on tests based on the code written to implement a component or system, but other aspects of structure, such as a menu structure, can be tested in a similar way.

Structure-based test case design techniques are based on the internal structure of the software program and code. Developers go into minute details of the developed code and test them one by one.

3. The test case design technique based on deriving test cases from the tester's experience of similar systems and general experience of testing, known as **experience-based techniques**.

   Experienced-based techniques are highly dependent on tester's experience to understand the most important areas of the software. They are based on the skills, knowledge, and expertise of the people involved.

1.2 IDENTIFICATION OF ERRORS AND BUGS IN THE GIVEN APPLICATION

- Identification or finding errors and bugs in a software application is the most challenging work. Software testing includes the identification of error or bug in a software application.
- An error is an incorrect human action that produces an incorrect result. In short, human mistakes cause error.
- Errors are of following types:
  1. A **syntax error** occurs in the source code of a program and prevents the program from being properly compiled. This type of error is very common and typically occurs when there are one or more missing or incorrect characters in the code. For example, a single missing bracket could cause a syntax error.
  2. A **logic error** represents a mistake in the software flow and causes the software to behave incorrectly. This type of error can cause the program to produce an incorrect output or even hang or crash. Unlike syntax errors, logic errors will not prevent a program from compiling. A common logic error is the infinite loop. Due to poorly written code, the program repeats a sequence endlessly until it crashes.
- A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result in operation or to behave in unintended ways.
- Software bug is classified as follows:
  1. **Functional bugs** are associated with the functionality of a specific software component. For example, a Login button doesn't allow users to login.
  2. A **logical bug** disrupts the intended workflow of software and causes it to behave incorrectly. For example, assigning a value to the wrong variable.
- Fig. 1.2 shows relationship between errors and bugs. An error/mistake is leads to defect/fault/bug. The defect leads to failure.

- A bug is the alternative name of defects, which says that application or software isn't functioning as in line with the requirement.
- When we have some coding error, program can have breakdown or failure in execution, which is known as a bug.
- When the application is not working as per the requirement, it is knows as defects. It is specified as the deviation from the actual and expected result of the application or software.
- The Bug is the informal name of defects, which means that software or application is not working as per the requirement.

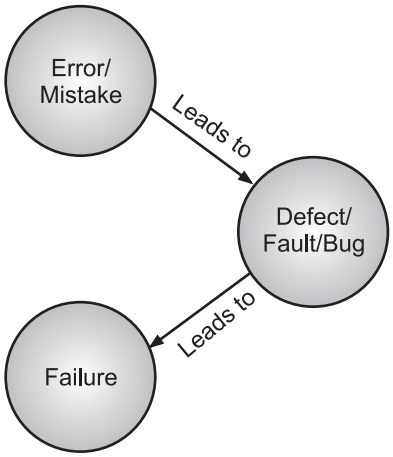


Fig. 1.2: Relation between Error, Bug and Failure

- Software testing is aimed at identifying any bugs, errors, faults or failures (if any) present in the software.
- Bug is defined as, a logical mistake which is caused by a software developer while writing the software code.
- Error is defined as, the measure of deviation of the output given by the software from the outputs expected by the user.
- Fault is defined as, the condition that leads to malfunctioning (caused due to several reasons such as change in the design, architecture or software code) of the software.
- The defect that causes an error in operation or a negative impact is called failure. Failure is defined as, that state of software under which it is unable to perform functions according to user requirements.
- Bugs, errors, faults and failures prevent software from performing efficiently and hence cause the software to produce unexpected output.
- In software testing, the bug can arise for reasons like Wrong coding, Missing coding and Extra coding.

1. **Wrong Coding:** Wrong coding means improper implementation. For example: Assume that if in Gmail application if we click on the "inbox" link, and it navigates

to the “draft” page instead of “inbox”, this is happening because of the wrong coding which is done by the developer, hence it is a bug.

2. **Missing Coding:** Here, missing coding means, we can say that the developer won't have developed the code only for that specific part of software. For an instance assume that if we take the above example and open the "inbox" link, we see that it is not there, this means that the feature is not developed or its coding is not done.
3. **Extra Coding:** Extra coding means that the developers add some extra features in the system which is not needed as per the requirements given by client. For example: Suppose we have one registration form wherein the Name field, the First name, and Last name textbox are needed to develop according to the client's requirement. But, here the developer write the code for "Middle name" textbox also and create it, but actually as per clients requirements it is not required as we can see in the Fig. 1.3.

   If we develop an additional functionality in the software which is not needed in the requirement, it leads to unnecessary added effort and it might also happen that adding up that additional functionality will affects the other part of the software.

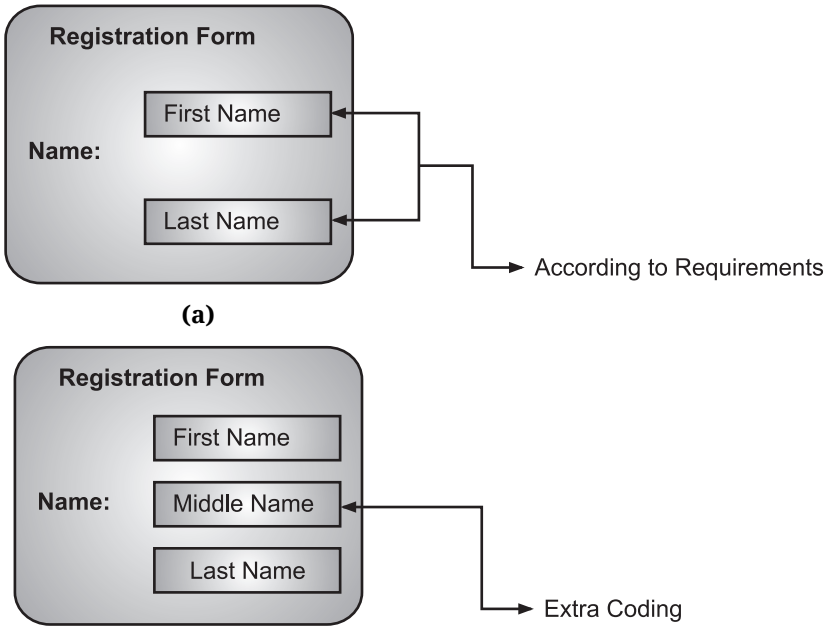


Fig. 1.3

- The defect/Bug tracking tool is used to monitor bug fixes and ensure the delivery of a quality application.
- This tool can help us to find the bugs in the testing stage so that we can get the defect-free data in the production server.

- With the help of these tools, the end-users can allow reporting the bugs and issues directly on their applications.

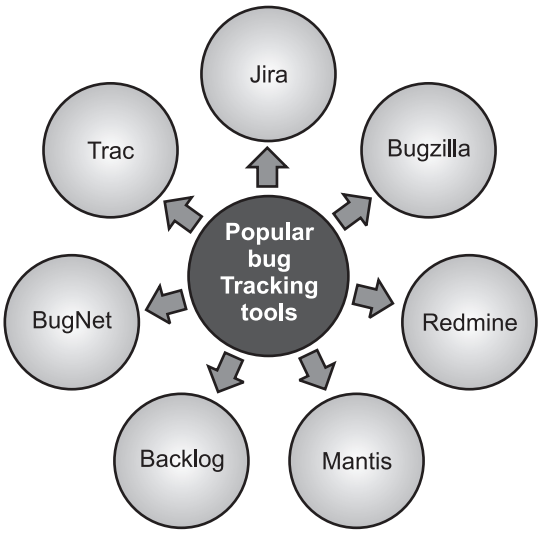


Fig. 1.4: Bug Tracking Tools

- Test case tracking is another concept which considers to the spreadsheet or database used to manage all the test cases in the test suites and how we track progress through that listing.
- Bug tracking has to do with the process which is used to manage the bugs. Since this systems form the principal communication channels inward to the own team, outward to other teams such as development and upward to the management, we should define all the things in the process well.
- We have various types of bug tracking tools available in software testing that helps us to track the bug, which is related to the software or the application.
- Some of the most usually used bug tracking tools are as below:

1. **Jira**: Jira is one of the most important bug tracking tools. Jira is an open-source tool that is used for bug tracking, project management, and issue tracking in manual testing. Jira comprises different features like recording, reporting, and workflow. We can monitor all kinds of bugs and issues, related to the software and generated by the test engineer using this tool Jira.

   Features of Jira are as follows:

   - It provides complete set of reporting of the bug tracking.
   - It supports integration with development tools such as GitHub.
   - It is a workflow-powered tool. Custom workflows can be created in Jira.
   - It is used to manage the defects very effectively and searching is very easy.

2. **Bugzilla:** Bugzilla is another important bug tracking tool, which is most widely used by many organizations to track the bugs. Bugzilla is an open source tool that is used to assist the customer, and the client to keep the track of the bugs. It is also used as a test management tool. Bugzilla supports several operating systems such as Linux, Windows, Mac.

   Features of Bugzilla are as follows:

   - A bug can be list in multiple formats.
   - Email notification controlled by user preferences.
   - Advanced searching capabilities.
   - Excellent security.
   - Time tracking.

3. **Redmine:** It is an open-source tool which is used to track the issues and web-based project management tool. Redmine tool is written in Ruby programming language and also compatible with multiple databases like MySQL, Microsoft SQL, and SQLite. While using the Redmine tool, users can also manage the various project and related subprojects.

   Some of the commonly known characteristics of Redmine tools are as follows:

   - Flexible role-based access control.
   - Time tracking functionality.
   - A flexible issue tracking system.
   - Feeds and email notification.
   - Multiple languages support (Albanian, Arabic, Dutch, English, Danish etc.).

4. **Mantis:** MantisBT stands for Mantis Bug Tracker. Along with the open source tool, it is also a web based bug tracking system. MantisBT is used to track the software defects. It is executed in the PHP programming language.

   Some of the commonly known characteristics of MantisBT tool are as follows:

   - Full-text search.
   - Audit trails of changes made to issues.
   - Revision control system integration.
   - Revision control of text fields and notes.
   - Notifications.
   - Plug-ins.
   - Graphing of relationships between issues.

5. **Backlog:** The backlog is widely used to manage the IT projects and track the bugs. It is primarily constructed for the development team for recording and reporting the bugs/defects with whole information of the problems and comments, updates and changes. It is a project management software tool.

Features of backlog tool are as follows:

- Gantt and burn down charts.
- It supports Git and SVN repositories.
- IP access control.
- Support Native iOS and Android apps.

6. **BugNet:** It is an open-source defect tracking and project issue management tool, which was written in ASP.NET and C# programming language and support the Microsoft SQL database. The goal of BugNet is to decrease the collaboration or dependency of the code that makes the deployment easy. It's advanced version is licensed to use for commercial purpose.

   Features of BugNet tool are as follows:

   - It will provide excellent security with simple navigation and administration.
   - BugNet supports various projects and databases.
   - With the help of this tool, we can get the email notification.
   - This has the capability to manage the Project and milestone.
   - This tool has an online support community.

7. **Trac:** Another defect/ bug tracking tool is Trac, which is also an open-source web-based tool written in Python. Various operating systems such as Windows, Mac, UNIX, Linux, etc. are supported by Trac. For tracking the issues for software development projects, Trac is useful. We can access it through code, view changes, and view history. Support of multiple projects is provided in this tool along with availability of wide range of plugins that provide many optional features, which keep the main system simple and easy to use.

1.3 DESIGN ENTRY AND EXIT CRITERIA FOR TEST CASE

- The entry and exit criteria for the test are closely related to the purpose and expected results for the test.
- Entry criteria and exit criteria in a test case are used to determine when a given test activity can start and when to stop.
- A test case consists of the test input, the entry criteria, the exit criteria and the expected output.
- After executing the test case, the expected result is compared with the actual result which indicates whether the software is functioning as desired/expected or not.
- In case of software testing, entry criteria defines the conditions to be satisfied in order for the testing to begin and exit criteria define the conditions that have to be satisfied in order to stop the testing.

- The **entry criteria** for a test are the requirements that need to be fulfilled before the test can run. For example, if the main web page has a textbox to fill out, then part of the entry criteria is filling out that textbox before clicking the Submit button.
- The **exit criteria** for a test case are a set of conditions based on which we can determine that the test case execution is finished. For example, after clicking the Submit button on the web page, if the web page navigates to the results page, then this is the exit criteria.

1.3.1 Concept and Design of Entry and Exit Criteria

- When we want to test a particular specification of the software, we have to give certain inputs to the software when it is being executed.
- For these test inputs, we expect some output (the expected result). But actually, the software will produce some output (the actual test result).
- Note that the expected result and the actual test result will be same if the software is working as per the design. Otherwise, it is fail.
- A test case defines design of entry criteria (before giving the test inputs) and exit criteria (after the testing)
- Fig. 1.5 shows a test case with entry criteria and exit criteria.
  - The entry criteria (the pre-conditions before the test inputs are given) i.e., test inputs.
  - The exit criteria (the post-conditions after the test inputs are executed) i.e., expected results.

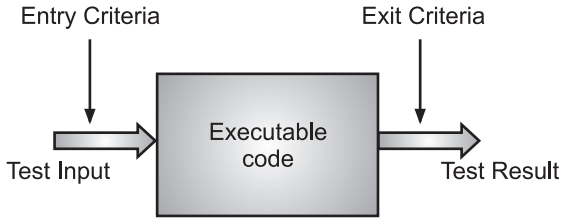


Fig. 1.5: Entry Criteria and Exit Criteria in a Test Case

- For example, suppose we need to test a program that takes a filename as input, opens the file, writes today's date into the file and closes the file. We can define a test case as follows:
  - **Test Input:** filename "xyz.dat".
  - **Entry Criteria:** The file xyz.dat should not exist in the current directory.
  - **Expected Output:** The file xyz.dat should be in the current directory, with the contents in today's date.
  - **Exit Criteria:** The program should terminate without errors. The file xyz.dat should be available in the current directory.

- We need to define a number of such test cases for testing all the specifications of the software.
- Fig. 1.6 shows a number of such test cases are selected, the program is executed and the results are compared with the estimated results.

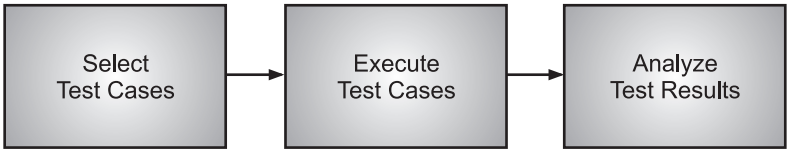


Fig. 1.6: Testing Process using Test Cases

1.3.2 Entry Criteria

- Entry criteria spell out what must happen to allow a system to move into a particular test phase. These criteria should address questions such as:
  - Are the necessary documentation, design, and requirements information available that will allow testers to operate the system and judge correct behavior?
  - Is the system ready for delivery, in whatever form is appropriate for the test phase in question?
  - Are the supporting utilities, accessories, and prerequisites available in forms that testers can use?
  - Is the system at the appropriate level of quality? This type of question usually indicates that some or all of a previous test phase has been successfully completed, with reference to the extent to which code overview troubles had been handled.
  - Is the test environment like lab, hardware, software, and system administration support, etc. ready?
- Following is an example of entry criteria that apply for a Java-based word processing package named SpeedyWriter, being written by Software Cafeteria, Inc.
  - SpeedyWriter has all the usual capabilities of a full-featured word processor, plus network file locking, Web integration, and public-key encryption.
  - SpeedyWriter includes various JavaBeans from other vendors.
  - **Example:** An example of a business driven set of System Test entry criteria for SpeedyWriter is given below.
  - **System Test for SpeedyWriter can begin when:**
    - Bug tracking and test tracking systems are in place.
    - All components are under formal, automated configuration and release management control.
    - The operations team has configured the System Test server environment, including all target hardware components and subsystems. The test team has been provided with appropriate access to these systems.

- The development teams have completed all features and bug fixes scheduled for release.
- The development teams have unit-tested all features and bug fixes scheduled for release.
- Less than 50 must-fix bugs (per Sales, Marketing, and Customer Service) are open against the first test release scheduled, (50 being the number of bug reports that can be effectively reviewed in a one-hour bug solving meeting.)
- The development teams provide software to the test team three business days prior to starting system test.
- The test team completes a three-day “smoke test” and reports on the results to the system test phase entry meeting.
- The project management team agrees in a system test phase entry meeting to proceed. The following topics will be resolved in the meeting:
  - Whether code is complete.
  - Whether unit-testing is complete.
  - Assign a target fix date for any known “must-fix" bugs (no later than one week after system test Phase Entry).

1.3.3 Exit Criteria

- Exit criteria address the issue of how to determine when testing has been completed. For example, one exit criterion can be - all the planned test cases and regression tests have been run.
- Another might be that project management deems your results “OK,” by whatever definition they use to decide such questions.
- In the case of system test exit criteria - provided system test is the last test phase on the project - these exit criteria often become the criteria by which the customer-ship or deployment decision is made.
  - **Example:** An example of a business driven set of system test exit criteria for SpeedyWriter is given below.
  - **System Test for SpeedyWriter will end when:**
    - No changes (design/code/features), except to address system test defects, occurred in the prior three weeks.
    - No panic, crash, halt, wedge, unexpected process termination, or other stoppage of processing has occurred on any server software or hardware for the previous three weeks.
    - No client systems have become inoperable due to a failed update during System Test.
    - The test team has executed all the planned tests against the GA-candidate software.

- The development teams have resolved all "must-fix" bugs per Sales, Marketing, and Customer Service.
- The test team has checked that all issues in the bug tracking system are either closed or deferred, and, where appropriate, verified by regression and confirmation testing.
- The test metrics indicate that we have achieved product stability and reliability that we have completed all planned tests and the planned tests adequately cover the critical quality risks.
- The project management team agrees that the product, as defined during the final cycle of system test, will satisfy the customer's reasonable expectations of quality.
- The project management team holds a system test phase exit meeting and agrees that we have completed system test.

1.4 DESIGN TEST CASES IN EXCEL

- A test case is a set of actions executed to verify a particular functionality of the software application.
- The primary goal of a test case is to ensure whether different features within an application are working as expected.
- The test case helps validate if software is free of defects and if it is working as per the expectations of the end users.
- Test case data can be managed in Excel. This requires an Excel spreadsheet which arranges the test data in rows and columns.
- Test designers can use MS Excel to design and build the tests. If we use MS Excel, it is best to keep everything in a single workbook and to include one test conditions spreadsheet, one test data spreadsheet and as many detailed test spreadsheets as are needed to adequately describe the environmental, pretest and post-test activities that are related to each test case.
- MS Excel functionalities are so powerful that we can write many similar test cases in very less time. For example, to write test cases for a long form with many fields, most of these test cases are repetitive with just the field name changed. We can use spreadsheet functions to write these test cases within few minutes. We can even import these test cases if we start using test management tool.

Standard Format of Test Cases Template:

- At the highest level, test cases are considered as consisting of a sequence of actions, each action having potentially some associated test data and some associated expected result.
- A test case template is a well-designed document for developing and better understanding of the test case data for a particular test case scenario.

- A good test case template maintains test artifact consistency for the test team and makes it easy for all stakeholders to understand the test cases.
- There are many different ways to document test cases. Myriad templates have been developed and used by various test teams.
- The industry standard, IEEE 829 template outline is shown in Fig. 1.7.

IEEE 829 Test Case Specification Template

Test Case Specification Identifier

Test Items

Describe features and conditions tested

Input Specifications

Data Names

Ordering

Values (with tolerances or generation procedures)

States

Timing

Output Specifications

Data Names

Ordering

Values (with tolerances or generation procedures)

States

Timing

Environmental Needs

Hardware

Software

Other

Special Procedural Requirements

Inter-Case Dependencies

Fig. 1.7: Template for Test Case

- Various templates are available, choose suitable template for the kind of system you are testing.
- Fig. 1.8 shows a basic test case template that can be used for either manual or automated testing.
- It complies with the IEEE 829 standard for test documentation, assuming that each test step includes the action to be taken, the associated data and the expected results. This template may vary according to organization.

F12

	A	B	C	D
1	**Test Case Name:**	Mnemonic identifier		
2	**Test ID:**	Five digit ID, XX.YYY: XX suite number, YYY test number.		
3	**Test Suite(s):**	The name of the test suite(s) that use this test case.		
4	**Priority:**	From coverage analysis		
5	**Hardware Required:**	List hardware in rows		
6	**Software Required:**	List software in rows		
7	**Duration:**	Elapsed dock time		
8	**Effort:**	Person-hours		
9	**Setup:**	List steps needed to set up the test		
10	**Teardown:**	List steps needed to return system under test to pretest state		
11				
12	**ID**	**Test Step/Substep**	**Result**	**Bug ID**
13	1	Major step		
14	1.001	Minor step(substep)		
15	1.002	Minor step(substep)		
16	2	Major step		
17				
18	**Execution Summary**	**Status**		
20		System Config ID		
21		Tester		
22		Date Completed		
23		Effort		
24		Duration		

Fig. 1.8: Basic Test Case Template

- The first 10 rows of the template identify and describe the test case, as the “header” section.
- It’s a good idea to name the test case with both mnemonic and numeric identifiers. The mnemonic name is a short description - for example, Stress, 1m Drop, or Network Performance.
- For the numeric identifier, Dewey decimal - style notation can be used. For example, a test case that is used in the fifth test suite and that is the second test case is assigned identifier 05.002.
- Alternatively, we can use a pure sequential numbering, to emphasize the many-to-many relationship between test cases and test suites.
- The next entry in the header lists the name of the test suite (or suites) in which the test case will be used.
- Because a given test case might be used in multiple suites, this entry could get a bit unwieldy.
- Most of the times practically, most test cases are used in only single test suite, so including the name of the suite provides some useful information with only rare confusion.

- Sometimes we can assign a priority to each test case. Prioritizing is most useful when we need to determine how many times a given test should be run.
- Here, priority is assigned based on intuition, the opinions of the sales, marketing, technical support, and development teams, coverage and quality risk analyses.
- Example for prioritizing is test coverage analysis, which allows us to allot the uppermost priority to those test cases that cover the most important quality risks, functions and requirements.
- The next entries in the header address resource requirements. For two of these entries, here listing is row by row, the hardware and software needed to run the test.
- We might want to restrict these lists to the nonobvious. For example, if we are testing a Windows based application and the standard test environment includes Microsoft Windows Me, Microsoft Office XP, Norton Antivirus, and LapLink, we needn't duplicate that listing for every test case.
- The entries for duration and effort specify how long it will take to run the test, in clock time and in person-hours, respectively.
- In creating these estimates, we have following two alternatives:
  1. We can assume that the test passes.
  2. We can make assumptions about the time impact of typical failures.
- The evaluation of person-hours shows the human resources needed to run the test. For example, do we need two people to run the test, single in front of each terminal?
- In the final two entries of the header, the setup procedures and the teardown procedures are specified.
- Sometimes there are none, but typically, two or three steps, such as installing or uninstalling an application, are needed at the beginning and end of a test.
- For example, after completion of the test, sample files those were created during the test needed to be deleted in order to return the system under test to its original state.
- A test case is fundamentally a sequence of actions, performed serially, in parallel, or in some combination, that creates the desired test conditions.
- It might involve the use of special test data, either entered as part of running the test step or prior to starting the test case.
- The test condition is associated with some expected result, which might be in an output, a system state, the timing or sequencing result or some other observable behavior.
- The template breaks down these actions into steps and sub-steps. Each step or sub-step has a numeric identifier.
- Use of Dewey decimal–style notation for numeric identifier is useful in bug reports and in discussions with testers.

- To the right of the list of steps are two columns that allow testers to record the results of their testing.
- The tester ran the test step or sub-step and observed,
  - the expected result,
  - the whole expected result,
  - nothing but the expected result.
- Following the execution of a test case, one of three statements will typically hold true for each step:
  - The test step or sub-step did not locate any bugs. The tester should record Pass in the Result column.
  - The tester ran the test step or sub-step, and the outcome was, to a greater or lesser extent, unexpected.
  - The test case was a success because the tester has identified some untoward behavior that can now be reported in your bug tracking database.
- How to classify the test case? If the unanticipated result was something along the lines of a CPU catching fire or a program crashing with a General Protection Fault or a system lockup, the tester should enter Fail in the Result column.
- However, what if the unexpected result is immaterial to the correct operation of the functionality under test? Development might see your team as unfair and alarmist if the tester throws this test case into the “failure” bucket.
- It is important, however, to keep track of the test case that did find a new bug, testers may not want to record it as a Pass.
- Entering Warn as a result is usually a good solution. A Warn entry can also cover some of the gray areas between complete failure and indirect failure - for example, if the functionality under test works correctly but causes an incorrectly spelled error message to be displayed.
- The tester did not run the test step or sub-step.
  - If running the step was impossible - for example, because the test was impeded by a known bug in the system or by the lack of essential resources - the tester should record Block in the Result column.
  - If the tester chose not to run the test step or sub-step, it should be marked Skip. In either case, the tester should explain this omission.
  - If a test step is marked Fail or Warn, the tester should also indicate, in the Bug ID column, the identifier for the bug report filed as a result of the observed failure. In all, we need a facility for recording bugs, and that each bug report so logged needs a unique identifier.

- At the bottom of the template is a summary section in which the tester indicates an overall assessment of the test case.
- The Status entry should be marked Pass, Warn, or Fail, depending on the success, Lack or Failure of the test case. (Remember, successful test cases find bugs.) The tester might also record Block or Skip if applicable.
- Since test cases consist of multiple steps, it is identified that a hierarchical rule is needed for assigning test case status based on test step status, such as, if any test step or sub-step is in progress, then the entire test case is "IP." Else, if any step or sub-step is blocked, then the entire test case is "Block." Else, if any step or sub-step failed, then the entire case is "Fail." Else, if any step or sub-step warned, then the entire case is "Warn." Else, if all steps pass, the entire case is "Pass." Here, we don't have a rule for "Skip" because generally this decision is made at the test case level.
- The tester should next note the specific system configuration used for the test. In the final part of the summary section, the tester should record his or her name or initials (depending on the custom), the date on which the test case was completed, the actual effort expended in terms of person-hours, and the duration. The latter three pieces of information allow us to track progress and understand variances from the plan that result from fast or slow test progress.
- Fig. 1.9 shows one more template for a test case in MS Excel.

	A	B	C	D	E	F	G	H	I
1	Test Case ID	Test Case Name	Description	Pre Conditions	Execution Steps	Expected Result	Actual result	Status	Comments
2	TC001	Feature_name OR Requirement number/Name as per SRS or client documents		1. 2.	1. 2. 3. 4.	As per requirements			
3									
4									

Fig. 1.9: Test Case template in MS Excel

- The common fields that are used in test case are explained below:
  1. **Test Case ID** field is defined by what type of system we are testing. Each test case should be represented by a unique ID. To indicate test types follow some convention like "TC_UI_01" indicating "User Interface Test Case#1."
  2. **Test Case Name** field contains name of the feature we are testing, Requirement number from the specifications and Requirement name as classified in client's document.
  3. **Test Description** field explains what type of feature we will test on which condition. This description should detail what unit, feature, or function is being tested or what is being verified.

4. **Steps To Execute** field contains the steps to be executed on the system being tested to get the expected results. Steps should be understandable and correct. They are written and executed according to a sequence.
5. **Pre-conditions** field must be fulfilled before the execution of the test case. Pre-conditions should be satisfied before the test case execution starts. List all the pre-conditions in order to execute this test case successfully.
6. **Execution Steps** field contains the steps to be performed on the system under test to get the desired results. Steps must be defined clearly and must be accurate. They are written and executed number wise.
7. **Expected Result** field contains the desired outputs from the execution steps performed. Results should be clearly defined for every step. It specifies what the specifications are and what we will get from a particular specification.
8. **Actual Result** field has the real/actual result after the performed execution steps on the system under testing. If the result matches with the expected result then we can write as expected.
9. **Status** field can state the test is Pass/Fail. If the result is showing according to the expected result, the test mark as pass and if not get the output according to the expected, result mark as fail. We can use color for status. Use the green color for Pass and red color for Fail.
10. **Comment** field column is for additional information for e.g. if status is set to "cannot be tested", then tester can give the reason in this column.

- Other fields for a test case are explained below:
  1. **Test priority (Low/Medium/High)** field is very useful during test execution.
  2. **Test Designed By** gives the Name of the Tester.
  3. **Test Designed Date** field gives the date when it was written.
  4. **Test Executed By** field gives name of the Tester who executed this test. To be filled only after test execution.
  5. **Test Execution Date** field gives the date when the test was executed.
  6. **Test Title/Name** field gives test case title. For example, verify the login page with a valid username and password.
  7. **Test Summary/Description** field describe the test objective in brief.
  8. **Post-condition** field gives the state of the system after executing the test case.
  9. **Test Scenario** field includes all the information about a test in the form of specific, detailed objectives that will help a tester perform a test accurately. It will not, however, include specific steps or sequences.
  10. **Test Data** field includes all the information and data that a test collects throughout the duration of the process. Use of test data as an input for the test case.

11. **Confirmation** field is the part of the process during which testers discuss and review whether or not a test was a success or a failure, based on the results.

- Following is an example of test case design in MS-Excel for Login functionality in Banking.

	A	B	C	D	E	F	G	H	I	J	K
1	**Test Case ID**		BU_001	**Test Case Description**		Test the Login Functionality in Banking					
2	**Created By**		Yogita	**Reviewed By**		Kiran		**Version**		4.1	
3											
4	**QA Tester's Log**		Review comments from Bill incorprate in version 2.1								
5											
6	**Tester's Name**		Amar	**Date Tested**		1-Jan-2017		**Test Case (Pass/Fail)**		Pass	
7											
8	**S #**	**Prerequisites:**				**S #**	**Test Data**				
9	1	Access to Chrome Browser				1	Userid = mg12345				
10	2					2	Pass = df12@434c				
11	3					3					
12	4					4					
13											
14	**Test Scenario**	Verify on entering valid userid and password, the customer can login									
15											
16	**Step #**	**Step Details**		**Expected Results**		**Actual Results**			**Pass / Fail / Not executed / Suspended**		
17											
18	1	Navigate to http://demo.nirali.com		Site should open		As Expected			Pass		
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass		
20	3	Click Submit		Cutomer is logged in		As Expected			Pass		
21	4										
22											
23											

- Following is another example of test case in MS-Excel for Facebook login functionality of the Web application:

Test Scenario ID	Login-1	**Test Case ID**	Login-1A
Test Case Description	Login – Positive test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

Test Execution Steps:

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**	**Test Comments**
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter correct Email & Password and hit login button	Email id : test@xyz.com Password: ******	Login success	Login success	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Login successful

1.5 FEATURE OF A TESTING METHOD USED

- Software testing methods are essential in building software. It helps developers deal with different types of bugs.
- As we all know, these bugs may range from a missing semicolon to a critical expected requirement.
- Testing methods like black-box testing, white-box testing applied to evaluate a system or a software with a purpose to find if it satisfies the given requirements.
- Change in software that adds new functionality or modifies the existing functionality is called "feature".
- Adding a feature plays a vital role in the Software Development Life Cycle (SDLC). Features are the ones that determine the functionality of the software.
- An effective and attractive developed feature requires testing to be done to maintain the quality of the software product.
- There are several tests used for testing the software. Each test has its own features. The following points, however, should be noted while doing testing:

1. **High Probability of Detecting Errors:** To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software can fail. For example, in a program to divide two numbers, the possible way in which the program can fail is when 2 and 0 are given as inputs and 2 is to be divided by 0. In this case, a set of tests should be developed that can demonstrate an error in the division operator.
2. **No Redundancy:** Resources and testing time are limited in software development process. Thus, it is not beneficial to develop several tests, which have the same intended purpose. Every test should have a distinct purpose.
3. **Choose the most Appropriate Test:** There can be different tests that have the same intent but due to certain limitations such as time and resource constraint, only few of them are used. In such a case, the tests, which are likely to find more number of errors, should be considered.
4. **Moderate:** A test is considered good if it is neither too simple, nor too complex. Many tests can be combined to form one test case. However this can increase the complexity and leave many errors undetected. Hence, all tests should be performed separately.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which is a process of executing a program or a system with the intent of finding/detecting defects/errors/bugs)?
   (a) Testing (b) Debugging
   (c) scheduling (d) None of the mentioned

2. Which is a software product that enables software testers to define software testing tasks?
   (a) Debugging tool (b) Testing tool
   (c) Both (a) and (b) (d) None of the mentioned
3. In software testing, which is the alternative name of defects and says that software application or software is not functioning as per specified/expected requirement?
   (a) test plan (b) test case
   (c) bug (d) None of the mentioned
4. While doing testing which of the following point is/are to be considered?
   (a) No redundancy (b) Choose the most appropriate test
   (c) High probability of detecting errors (d) All of the mentioned
5. It is a good idea to name the test case with,
   (a) only non-numeric identifiers (b) only numeric identifier
   (c) mnemonic and numeric identifiers (d) only mnemonic identifiers
6. A test case contains,
   (a) a set of test data (b) expected results
   (c) preconditions and post-conditions (d) All of the mentioned
7. Which describes input, action and an expected result, in order to determine the functionality of a software application works correctly or not?
   (a) a test plan (b) a test report
   (c) a test case (d) All of the mentioned
8. In a Dewey decimal - style notation, a test case that is used in the fifth test suite and that is the second test case is assigned identifier,
   (a) 05.0025 (b) 05.0012
   (c) 05.0021 (d) 05.002
9. Which is in a test case is most useful when we need to determine how many times a given test should be run.
   (a) Designing (b) Prioritizing
   (c) Planning (d) None of the mentioned
10. In the execution of a test case, which of the following statements will typically hold true for each step?
    (a) The test step or sub-step did not locate any bugs. The tester should record Pass in the result column.
    (b) The tester ran the test step or sub-step, and the outcome was, to a greater or lesser extent, unexpected.
    (c) The test case was a success because the tester has identified some untoward behavior that can now be reported in your bug tracking database.
    (d) All of the mentioned

11. Which is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly?
    (a) bound report (b) test document
    (c) bug report (d) test case
12. Which addresses the issue of how to determine when testing has been completed?
    (a) Entry criteria (b) Exit criteria
    (c) Both (a) and (b) (d) None of the mentioned
13. Which of the following is/are bug tracking tools?
    (a) Jira (b) Bugzilla
    (c) Backlog (d) All of the mentioned
14. Which of the following is an open-source defect tracking and project issue management tool, which was written in ASP.NET and C# programming language?
    (a) MantisBT (b) BugNet
    (c) Redmine (d) Trac
15. Test cases in software testing have to be designed based on,
    (a) reliability (if they detects all errors)
    (b) validity (if at least one test case reveals the errors)
    (c) Both (a) and (b) (d) None of the mentioned
16. The test case design technique based on deriving test cases directly from a specification of a system called as,
    (a) experience-based techniques (b) structure-based techniques
    (c) specification-based techniques (d) None of the mentioned
17. Which is the measure of deviation of the outputs given by the software from the outputs expected by the user?
    (a) error (b) teat plan
    (c) test case (d) None of the mentioned
18. Which software is used for design and builds test cases?
    (a) MS Word (b) MS Excel
    (c) MS Access (d) MS PowerPoint
19. Features of software testing methods include,
    (a) Choose the most appropriate test
    (b) High probability of detecting errors
    (c) No redundancy (d) All of the mentioned

Answers

1. (a)	2. (b)	3. (c)	4. (d)	5. (c)	6. (d)	7. (c)	8. (d)	9. (b)	10. (d)
11. (d)	12. (b)	13. (d)	14. (b)	15. (c)	16. (c)	17. (a)	18. (b)	19. (d)	

Q. II Fill in the Blanks:

1. When the application is not working as per the requirement, it is known as _____.
2. Trac is an open-source web-based tool written in _____.
3. _____ coding means improper implementation in which functionality is not correctly linked.
4. MantisBT is used to track the software defects and it is executed in the _____ programming language.
5. _____ spell out what must happen to allow a system to move into a particular test phase.
6. If system test is the last test phase on the project then _____ often become the criteria by which the customer-ship or deployment decision is made.
7. In test case template the entries for _____ specify how long it will take to run the test, in clock time and in person-hours, respectively.
8. If the tester chose not to run the test step or sub-step, it should be marked _____.
9. In the test case template, the _____ entry should be marked Pass, Warn or Fail, depending on the success, lack or failure of the test case.
10. The _____ testing tool is mainly, used for finding the bugs present in the software and catch that bugs for further processing.
11. Test case _____ techniques are standards of test designing that allow the creation of systematic and widely accepted test cases.
12. Software _____ is the process of executing software or system with the intent of finding errors.
13. The test case design technique based on deriving test cases from the tester's experience like skills, knowledge etc., of similar systems and general experience of testing, known as _____ techniques.
14. Software testing _____ are the tools which are used for the testing of software.
15. A _____ provides the description of inputs and their expected outputs to observe whether the software or a part of the software is working correctly or not.
16. _____ also refer to human actions that results in software containing a defect or fault.

Answers

1. defects	2. Python	3. Wrong	4. PHP
5. Entry criteria	6. exit criteria	7. Duration and Effort	8. Skip
9. Status	10. Jira	11. design	12. testing
13. experience-based	14. tools	15. test case	16. Error

Q. III State True or False:

1. Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified/expected requirements or not and to identify the defects to ensure that the software product is defect-free in order to produce a quality product.
2. A test case consists of the test input, the entry criteria, the exit criteria and the expected output.
3. A software bug is termed as error, flaw, failure or fault in computer program or system and in terms gives incorrect and unexpected results in operations.
4. Entry Criteria and Exit Criteria are not part of test plan template.
5. The defect/bug tracking tool is used to monitor bug fixes and ensure the delivery of a quality application.
6. A testing tool is a software product that enables software testers to define software testing tasks.
7. Test case data can be managed in MS Access (uses spreadsheet which arranges the test data in rows and columns).
8. MantisBT stands for Mantis Bug Tracker.
9. Redmine tool is written in ASP.NET and C# programming language and is not compatible with multiple databases.
10. Test design is a process that describes “how” testing should be done.
11. To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software should not fail.
12. Using a test tool an organization achieves greater speed, reliability and efficiency in their testing process.
13. The entry criteria (the conditions that must be met before we can start the test) and exit criteria (the conditions that must be met before the test is completed) for the test are closely related to the purpose and expected results for the test.
14. The test case design technique based on deriving test cases directly from the structure of a software system or a system known as structure-based techniques.

Answers

1. (T)	2. (T)	3. (T)	4. (F)	5. (T)	6. (T)	7. (F)	8. (T)	9. (F)	10. (T)
11. (F)	12. (T)	13. (T)	14. (T)						

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define software testing.
2. What is testing tool?
3. Define test case.

4. Define is error.
5. Define defect.
6. Define bug.
7. What is meant by missing coding?
8. What is extra coding?
9. What is wrong coding.
10. Define test case tracking?
11. Enlist any two features of Bugzilla tool.
12. List any two the characteristics of Redmine tool.
13. What is the purpose of Entry and Exit Criteria?
14. List out the characteristics of MantisBT tool. Any two.
15. What is test suite?
16. Enlist any two features of Backlog tool.
17. Which are different characteristics of BugNet tool?
18. List techniques for test case design.
19. “MS Excel is used for design a test case”. State true or false.

(B) Long Answer Questions:

1. What is test case? How to design it? Which techniques are used for designing a test case?
2. What is software testing? Why it needed? Also explain its purpose.
3. What is test tool? What is its purpose? Also give name of popular testing tools.
4. Which are the different features to be considered while doing software testing, explain it?
5. Draw and explain in short, IEEE 829 Test Case Specification Template outline.
6. Explain the various fields of a basic test case template in excel that can be used for either manual or automated testing.
7. Explain Entry Criteria by giving an example.
8. Which are different bug tracking tools? Explain any two.
9. Elaborate Exit Criteria with an example.
10. How to identify errors, bugs in the given application.
11. How to design test cases in MS Excel? Describe with example.

❖❖❖

CHAPTER 2

Test Cases for Simple Programs

Objectives...

- To write Test Cases for Simple Programs
- To learn Test Cases for Control Statements
- To write Test Cases for Loop Statements

2.0 INTRODUCTION

- Testing is concerned with errors, faults, failures, and incidents. A test is the act of exercising software with test cases.
- A test has two distinct goals namely, to find failures or to demonstrate correct execution.
- A test case has an identity and is associated with program behavior. A test case also has a set of inputs and a list of expected outputs.
- The essence of software testing is to determine a set of test cases for the item to be tested.
- The main goal of test case design techniques is to test the functionalities and features of software with the help of effective test cases.
- The test case design techniques are broadly classifies as:
  1. **Specification-based Test Case Design Techniques:** The specification-based test case design techniques are used to design test cases based on an analysis of the description or model of the product without reference to its internal workings. These techniques are also known as black-box test case design techniques.

     These techniques leverage external description of software to test cases such as Technical specifications, Designs, Client's requirements and so on.
  2. **Structure-based Test Case Design Techniques:** The structural test case design techniques are used to design test cases based on an analysis of the internal structure of the test item. These techniques are also known as white-box test case

design techniques. Traditionally, the internal structure has been interpreted as the structure of the code.

In these techniques the test cases are designed based on internal design such as Software code, Internal structure, Design and so on.

3. **Experience-based Techniques:** Experience-based test case design techniques are used to complement black-box and white- box techniques. In experience-based test techniques, people's knowledge, skills and background are a prime contributor to the test conditions, test cases and test data. The experience of both technical and business people is required, as they bring different perspectives to the test analysis and design process. Due to previous experience with similar systems, they may have insights into what could go wrong, which is very useful for testing. Both structural and behavioral insights are used to design experience-based tests.

   All experience-based test case design techniques have the common characteristic that they are based on human knowledge and experience. Test cases are therefore derived in a less systematic way, but may be more effective. Experience-based test case design techniques are highly dependent on tester's experience, (Skills, Knowledge, Expertise and so on).

- White-box testing is a way of testing the external functionality of the code by examining and testing the program code that realizes the external functionality. White-box testing is also known as clear box or glass box or open box testing.
- White-box testing takes into account the program code, code structure, and internal design flow.
- A number of defects come about because of incorrect translation of requirements and design into program code. Some other defects are created by programming errors.
- White-box testing is further has two types namely, Static testing and Structural testing.
  1. **Static testing** is a type of testing which requires only the source code of the product, not the binaries or executables.
  2. **Structural testing** takes into account the code, code structure, internal design, and how they are coded.
- White-box testing verifies the designs and codes involved in the development of a software product.

2.1 WRITE SIMPLE PROGRAMS MAKE USE OF LOOPS AND CONTROL STRUCTURES WITH TEST CASES

- During white-box testing, test inputs are chosen to invoke as many program statements as possible.
- The percentage of program statements that can be invoked during this phase of testing is called code coverage.

- Code coverage is white-box testing because it requires us to have full access to the code to view what parts of the software we pass through when we run the test and what parts of the software fail.
- The primary purpose of code coverage testing is to ensure that the testing activity covers as much code as possible.
- Code coverage testing is a technique that involves measuring the percentage of a system's code that is being tested by a test suite.
- Coverage is measured based on the items tested within a selected structure, for example the code statements, the decisions, the interfaces, the menu structure or any other identified structural element.
- Structural testing examines source code and analyses what is present in the code. Structural testing cannot expose errors of code omission but can estimate the test suite adequacy in terms of code coverage, i.e., execution of components by the test suite or its fault-finding ability.
- Code coverage includes creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once).
- Code coverage testing is determining how much code is being tested. It can be calculated using the formula:

$$\text{Code Coverage} = \frac{\text{Number of lines of the code exercised}}{\text{Total number of lines of the code}} \times 100$$

- Code coverage helps in determining how much code of the source is tested which helps in accessing quality of test suite and analyzing how comprehensively a software is verified.
- Actually simple code coverage refers to measure of the degree to which the source code of the software code has been tested. This code coverage is considered as one of the form of white-box testing.
- Code coverage testing is a dynamic white-box testing where testing is done by executing the test and the tester has complete access to the code.
- The tester (he/she) is also aware of the parts of the software that have passed and the parts that have failed.
- Code based testing involves detecting errors by following execution oriented methods, is also known as white-box testing.
- Here, a tester is expected to know and understand the low level design and identify the code-based approach and to apply the techniques in the code that is written.
- Commonly used code coverage testing techniques are Statement coverage testing, Decision coverage testing, Branch coverage testing, Loop coverage testing and Path coverage testing.

Statement Coverage Testing:

- Statement coverage testing is the simplest form of white-box testing, whereby a series of test cases are run such that each statement is executed at least once.
- Statement coverage testing's achievement is insufficient to provide confidence in a software product's behavior.
- Often, a CASE (Computer Aided Software Engineering) tool keeps track of how many times each statement has been executed.
- A weakness of statement coverage testing is that there is no guarantee that all outcomes of branches are properly tested.

  Example:

```
if(s > 1 && t==0)
    x = 9;
```

  Test case: s = 2; t = 0;

  Here, the programmer has made a mistake. As per requirements the compound conditional (s>1 && t==0) should have been (s>1 || t==0). The chosen test data, however, allow the statement x = 9 to be executed without the fault being detected.
- Statement coverage is a white-box testing technique in which all the executable statements in the source code are executed at least once.
- Statement coverage is used for calculation of the number of statements in source code which have been executed.
- The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.
- Statement coverage refers to writing test cases that execute each of the program statements.
- The principal idea governing the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement.
- Unless a statement is executed, it is very difficult to observe whether it causes failure due to some illegal memory access, wrong result computation, etc.
- However, executing some statement once and observing that it behaves properly for that input value is no guarantee that it will behave correctly for all input values.
- In most of the programming languages, the program construct may be a sequential control flow, a two-way decision statement like if-then-else, a multi-way decision statement like switch or even loops like while, do, repeat until and for.
- In a **sequential structure**, the program has a series of instructions or statements that are executed consecutively in the sequence, they are written.
- For a set of sequential statements (i.e., with no conditional branches), test cases can be designed to run through from top to bottom.

- The following program is purely sequential structure:
  1. Read X
  2. Read Y
  3. Z = X + Y

  In above program code, the computer would execute those three statements in sequence, so it would read (input) a value into X (this is just a name for a storage location), then read another value into Y, and finally add them together and put the answer into Z.
- A **decision or selection control structure** consists of a test condition and one or more blocks of code.
- The results of the test determine which of these blocks are to be executed. It has simple if, if then else, nested if or switch structures.
- In case of an if-then-else statement, if we want to cover all the statements then we should also cover the "then" and "else" parts of the 'if' statement.
- This means that we should have, for each if-then-else, at least one test case to test the "then" part and at least one test case to test the "else" part.
- The multi-way, switch statement can be reduced to multiple two-way if statements. Thus, to cover all possible switch cases, there would be multiple test cases.

Example:

```
1. IF P > 10
2. THEN
3.       X = X + Y
4. ELSE
5.       X = X - Y
6. ENDIF
```

In above program code, we ask the computer to evaluate the condition P > 10, which means compare the value that is in location P with 5. If the value in P is greater than 10 then the condition is true. If not, the condition is false. The computer then selects which statement to execute next. If the condition is true it will execute the part labelled THEN, so it executes line 3. Similarly if the condition is false it will execute line 5. After it has executed either line 3 or line 5 it will go to line 6, which is the end of the selection [IF THEN ELSE] structure. From there it will continue with the next line in sequence.

There may not always be an ELSE part, as below:

```
1. IF P > 10
2. THEN
3.       X = X + Y
4. ENDIF
```

In above program code, the computer executes line 3 if the condition is true, or moves on to line 4 [the next line in sequence] if the condition is false.

- In a **loop structure**, the program uses while, do while or for loops. The codes inside the block of loops are repeated again and again, till they reach to a condition from where the loop has to terminate.

  Example: Iteration structures are called loops. The most common loop is DO WHILE loop and is given below:

  1. X = 10
  2. Count = 0
  3. WHILE X < 20 DO
  4. X = X + 1
  5. Count = Count + 1
  6. END DO

  As with the selection structures there is a decision. In above program code, the condition that is tested at the decision is X < 20. If the condition is true the program 'enters the loop' by executing the code between DO and END DO. In this case the value of X is increased by one and the value of Count is increased by one. When this is done the program goes back to line 3 and repeats the test. If X < 20 is still true the program 'enters the loop' again. This continues as long as the condition is true. If the condition is false the program goes directly to line 6 and then continues to the next sequential instruction. In the program fragment above the loop will be executed five times before the value of X reaches 20 and causes the loop to terminate. The value of Count will then be 5.
- Statement coverage is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
- The formula for calculating statement coverage is given below:

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

- Generally, in the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc.
- Some statements from code are executed and some may not be executed, depending on the input given to the program.
- The aim of statement coverage technique is to cover all of the statements whose execution is possible and path lines in the code.

Example 1:

Source Code Structure:

- Take input of two values like values for x and y.
- Find the sum of these two values.

- If the sum is greater than 0, then print "This is the positive result."
- If the sum is less than 0, then print "This is the negative result."

Code for the given structure is: (C programming is considered here),

```
1. input (int x, int y) {
2.    sum= x+y;
3.    if (sum>0)
4.       printf(This is positive result);
5.    else
6.       printf(This is negative result);
7. }
```

- Let us consider two different test case scenarios and find the percentage of statement coverage for given source code.

Test Case 1:

If x = 6, y = 2

```
1. input (int x, int y){
2.    int sum= x+y;
3.    if (sum>0)
4.       printf(This is positive result);
5.    else
6.       printf(This is negative result);
7. }
```

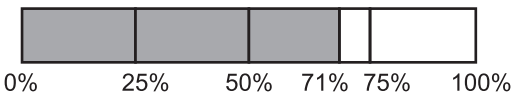
- In Test Case 1, it is observed that the value of sum will be 8 which is greater than 0 and as per the condition result will be "This is a positive result."
- For calculating statement coverage, here total count of statements is 7 and count of statements used in execution is 5.
- Total number of statements = 7.
- Number of executed statements = 5.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

$$\text{Statement coverage} = 5/7 \times 100$$

$$= 500/7$$

$$= 71\%$$



Test Case 2:

If x = -4, y = -3

```
1. input (int x, int y){
```

```
2.    int sum= x+y;
3.    if (sum>0)
4.       printf(This is positive result);
5.    else
6.       printf(This is negative result);
7.  }
```

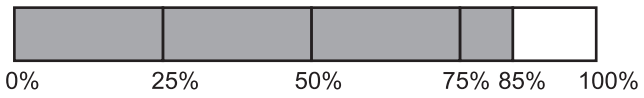
- In Test Case 2, it is observed that the value of sum will be -7 which is less than 0 and as per the condition, result will be "This is a negative result."
- For calculating statement coverage, here total count of statements is 7 and count of statements used in execution is 6.
- Total number of statements = 7.
- Number of executed statements = 6.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

$$\text{Statement coverage} = 6/7 \times 100$$

$$= 600/7$$

$$= 85\%$$



Example 2: Consider the following code:

```
input (int X, int Y){
   int Z = ((X + Y) / 200 )* 100;
   if(Z>50)
      printf("PASS");
    else
      printf("FAIL");
}
```

Let us consider two different test case scenarios and find the percentage of statement coverage for given source code.

Test Case 1:

If X=20 , Y = 30

```
1.  input (int X, int Y){
2.  int Z = ((X + Y) / 200 )* 100;
3.  if (Z>50)
4.     printf("PASS");
```

```
5. else
6.     printf(“FAIL”);
7. }
```

In Test Case 1, it is observed that the value of Z will be 25 which is less than 50 and as per the condition result will be "Fail".

For calculating statement coverage, here total count of statements is 7 and count of statements used in execution is 6.

Total number of statements = 7.

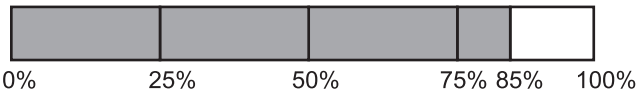
Number of executed statements = 6

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

$$\text{Statement coverage} = 6/7 \times 100$$

$$= 600/7$$

$$= 85\%$$



Test Case 2:

If X = 100, Y = 75

```
1. input (int X, int Y){
2. int Z = ((X + Y) / 200 )* 100;
3. if (Z>50)
4.     printf(“PASS”);
5. else
6.     printf(“FAIL”);
7. }
```

In Test Case 2, it is observed that the value of Z will be 87.5 which is greater than 50 and as per the condition result will be "Pass".

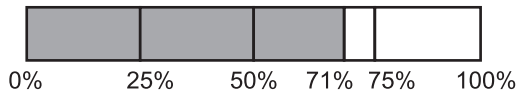
For calculating statement coverage, here total count of statements is 7 and count of statements used in execution is 5.

Total number of statements = 7.

Number of executed statements = 5.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

Statement coverage = 5/7 × 100
= 500/7
= 71%



Example 3: Consider the following Euclid's GCD computation algorithm,

Consider the Euclid's GCD computation algorithm:

```
int compute_gcd(x, y)
int x, y;
{
1.  while (x! = y)
2.  {
3.  if (x>y) then
4.      x= x – y;
5.  else y= y – x;
6.  }
7.  return x;
8.  }
```

By choosing the test set {(x=3, y=3), (x=4, y=3), (x=3, y=4)}, we can exercise the program such that all statements are executed at least once. For each input we can calculate statement coverage.

We can try out many more test cases by passing different input values including negative values or by passing 0 or not passing any value at all.

Advantage of Statement Coverage:

1. It verifies what the written code is expected to do and not to do
2. It measures the quality of code written
3. It checks the flow of different paths in the program and it also ensures that whether those path are tested or not.

Disadvantage of statement coverage:

1. It cannot test the false conditions.
2. It does not report that whether the loop reaches its termination condition.
3. It does not understand the logical operators.

Decision Coverage Testing:

- Decision coverage is a white-box testing technique which reports the True or False outcomes of each Boolean expression of the source code.

- Decision coverage testing criteria requires sufficient test cases for each condition in a program decision to take on all possible outcomes at least once.
- Decision coverage testing differs from branch coverage only when multiple conditions must be evaluated to reach a decision.
- Multi-condition coverage requires sufficient test cases to exercise all possible combinations of conditions in a program decision.
- The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.
- Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (control flow statements), it is considered as decision point as outcome is either True or False from two possible outcomes.
- Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.
- Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two.
- The percentage of decision coverage can be found by dividing the number of exercised outcome with the total number of outcomes and multiplied by 100.

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$$

Example 1: Consider the pseudo code to apply on decision coverage technique:

```
1. Test (int x)
2. {  if(x>4)
3.     x=x*3
4.     print (x)
5. }
```

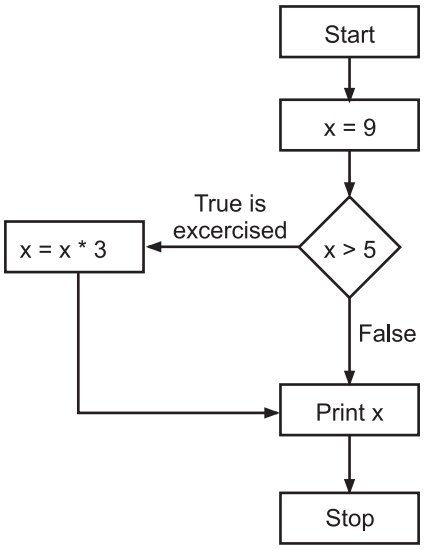
Test Case 1:

Value of x is 9 (x=9)

```
1. Test (int x=9)
2. {  if (x>5)
3.     x=x*3
4.     print (x)
5. }
```

The outcome of this code is "True" if condition (x>5) is checked.

For the above example, following is flowchart or flowgraph when the value of x is 9.



Calculation of Decision Coverage percentage:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$$

Decision coverage = ½ × 100 (Only "True" is exercised)

= 100/2

= 50

Decision coverage is 50%.

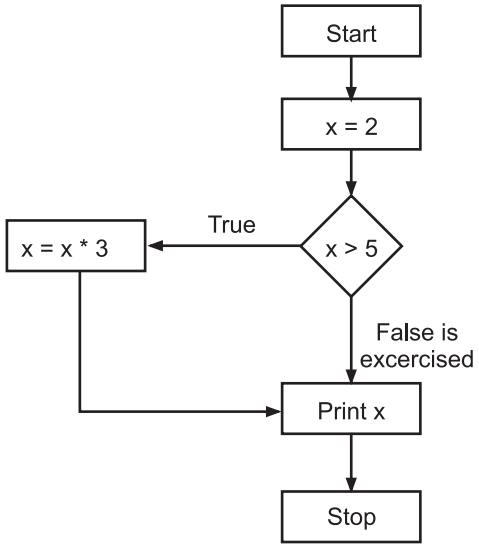
Test Case 2:

Value of x is 2 (x=2)

```
1. Test (int x=2)
2. {  if (x>5)
3.      x=x*3
4.      print (x)
5. }
```

The outcome of this code is “False” if condition (x > 5) is checked.

For the above example the following is the flowchart or flow graph when the value of x is 2.



Calculation of decision coverage percentage:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$$

$= ½ \times 100$ (only "False" is exercised)

$= 100/2$

$= 50$

Decision coverage is 50%.

Result table of decision coverage:

Test Case	Value of x	Output	Decision Coverage
1	9	27	50%
2	2	2	50%

Example 2: The below sample is for validating the age of the patient, and determining if the person is a senior citizen or not.

Consider the pseudo code to apply on decision coverage technique:

```
1. Test_age (int x)
2. {
3. if(x>=60)
4.    print (“Senior Citizen”);
5. else
6.    print (“Not a Senior Citizen”);
7. }
```

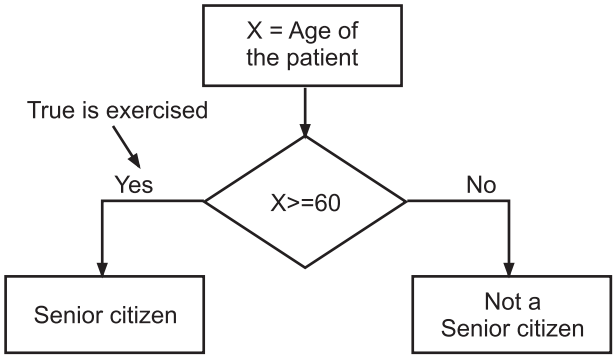
Test Case 1:

Value of x is 76 (x=76)

```
1. Test (int x=76)
2. {
3. if (x>=60)
4.    print (“Senior Citizen”);
5. else
6.    print (“Not a Senior Citizen”);
7. }
```

The outcome of this code is "True" with message “Senior Citizen” if condition (x>=60) is checked and results in success.

For the above example the following is the flowchart or flowgraph when the value of x is 76.



Calculation of decision coverage percentage:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$$

$= ½ \times 100$ (Only "True" is exercised)

$= 100/2$

$= 50$

Decision Coverage is 50%.

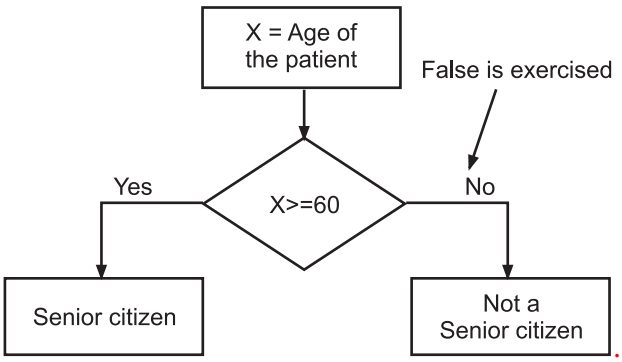
Test Case 2:

Value of x is 45 (x=45)

```
1. Test (int x=45)
2. {
3. if (x>=60)
4.    print (“Senior Citizen”);
5. else
6.    print (“Not a Senior Citizen”);
7. }
```

The outcome of this code is "False" with message “Not a Senior Citizen” if condition (x>=60) is checked and results in fail.

For the above example the following is the flowchart or flowgraph when the value of x is 45.



Calculation of decision coverage percentage:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$$

= ½*100 (Only "False" is exercised)

= 100/2

= 50

Decision coverage is 50%.

Result table of decision coverage:

Test Case	Value of x	Output	Decision Coverage
1	76	Senior Citizen	50%
2	45	Not a Senior Citizen	50%

Advantages of Decision Coverage:

1. Validate that all the branches in the code are reached.
2. Ensure that no branches lead to any abnormality of the program's operation.
3. It eliminates problems that occur with statement coverage testing.

Disadvantages of Decision Coverage:

1. This metric ignores branches within Boolean expressions which occur due to short-circuit operators.

Branch Coverage Testing:

- Branch coverage is an improvement over statement coverage, in that a series of tests are run to ensure that all branches are tested at least once.
- Branch coverage is also sometimes referred to as all edge coverage.
- Branch coverage requires sufficient test cases for each program decision or branch to be executed so that each possible outcome occurs at least once.
- Branch coverage is considered to be a minimum level of coverage for most software products, but decision coverage alone is insufficient for high-integrity applications.
- Branch coverage is a white-box testing method in which every outcome from a code module (statement or loop) is tested.
- The purpose of branch coverage testing is to ensure that each decision condition from every branch is executed at least once.
- It helps to measure fractions of independent code segments and to find out sections having no branches.
- Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
- However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two.

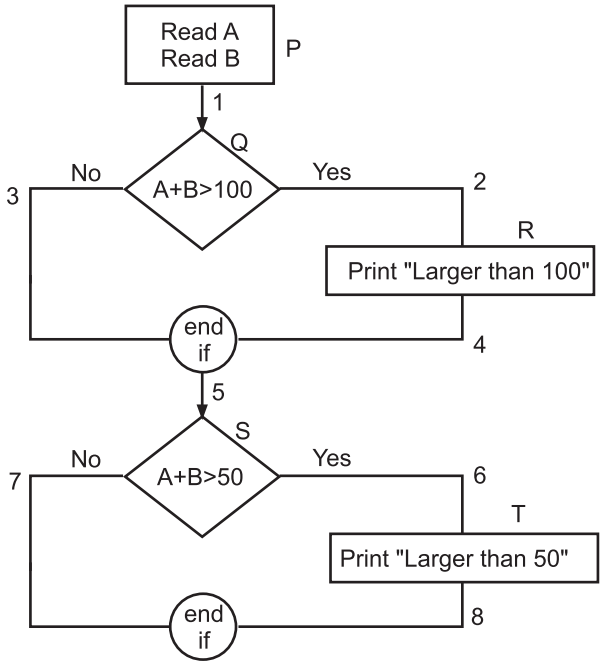
- Decision coverage technique measures the coverage of conditional branches whereas branch coverage measures the coverage of conditional and unconditional branches. So we can say, branch coverage considers decision point and branch coverage edges.
- Like decision coverage, it also uses a control flow graph to calculate the number of branches.
- There are several methods to calculate Branch coverage, but pathfinding is the most common method.
- In this method, the number of paths of executed branches is used to calculate Branch coverage.
- Sometimes, branch coverage technique can be used as the alternative of decision coverage as both are closely related an at 100% coverage they give exactly the same results.

Example 1: Consider the following code:

1. Read A
2. Read B
3. If A+B>100 THEN
4. Print "Larger than 100"
5. ENDIF
6. If A+B>50 THEN
7. Print "Larger than 50"
8. ENDIF

This is the basic code structure where we took two variables A and B and two conditions. If the first condition is true, then print "Larger than 100" and if it is false, then go to the next condition. If the second condition is true, then print "Larger than 50".

For the above example we draw the flowchart or flow graph as shown right side.



In the above flow chart, control flow graph of code is depicted. In the first case traversing through "Yes" decision, (for A=100 and B=10) the path is P1-Q2-R4-5-S6-T8 and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path.

To cover these edges, we have to traverse through "No" decision. In the case of "No" decision, (for A=100 and B=10) the path is P1-Q3-5-S7, and the number of covered edges is 1, 3, 5 and 7. So by traveling through these two paths, all branches have covered.

- Path 1: P1-Q2-R4-5-S6-T8
- Path 2: P1-Q3-5-S7
- Branch Coverage (BC) = Number of paths = 2.

To calculate percentage branch coverage, one has to find out the minimum number of paths which will ensure that all the edges are covered.

In this case there is no single path which will ensure coverage of all the edges at once. The aim is to cover all possible true/false decisions.

The formula for finding branch coverage percentage is given below:

$$\text{Branch coverage} = \frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100$$

Branch Coverage will consider unconditional branch as well.

For above example:

Branch coverage for result Yes:

Branch Coverage = 5/8 × 100 = 62.5%.

Branch coverage for result No:

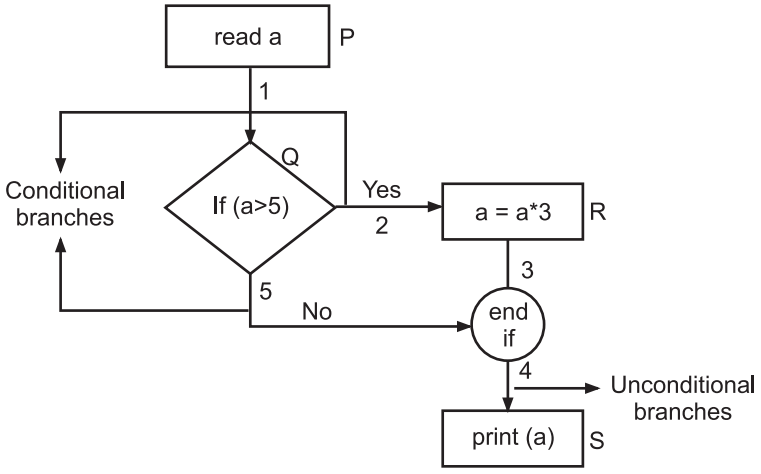
Branch Coverage = 3/8 × 100 = 37.5%.

Case	Covered Branches (Conditional + Unconditional)	Path	Branch Coverage
Yes	2, 4, 5, 6, 8 (from conditions)	P1-Q2-R4-5-S6-T8	62.5 %
No	3, 5, 7 (from conditions)	P1-Q3-5-S7	37.5%

Example 2: Consider the following code

```
1. read a;
2. if (a> 5)
3.   a=a*3;
4. end if
5.   print (a);
```

For the above example the following is the flowchart or flow graph:



$$\text{Branch coverage} = \frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100$$

Branch coverage for result Yes:

Branch Coverage = 3/5 × 100 = 60%.

Branch coverage for result No:

Branch Coverage = 2/5 × 100 = 40%.

Case	Covered Branches (Conditional + Unconditional)	Path	Branch Coverage
Yes	2, 3, 4 (from conditions)	P1-Q2-R3-4S	60%
No	5, 4 (from condition)	P1-Q5-4S	40%

Advantages of Branch Coverage:

1. It allows to validate-all the branches in the code.
2. It helps to ensure that no branched lead to any abnormality of the program's operation.
3. Branch coverage method removes issues which happen because of statement coverage testing.
4. It allows finding those areas which are not tested by other testing methods.
5. It allows finding a quantitative measure of code coverage.

Disadvantages of Branch coverage:

1. This metric ignores branches within Boolean expressions which occur due to short-circuit operators.
2. It is costly.
3. It is take more time for performing this task.

Loop Coverage Testing:

- Loop coverage testing criteria requires sufficient test cases for all program loops to be executed for zero, one, two, and many iteration covering initialization, typical running and termination (boundary) conditions.
- Loop testing is a type of white box testing technique to validate the loops. It is one of the type of control structure testing.
- Loop testing is a test coverage criteria which checks whether loop body executed zero times, exactly once or more than once.
- The goals of loop coverage testing are given below:
  1. To fix the infinite loop repetition problem.
  2. To know the performance.
  3. To identify the loop initialization problems.
  4. To determine the uninitialized variables.

Types of Loop Testing:

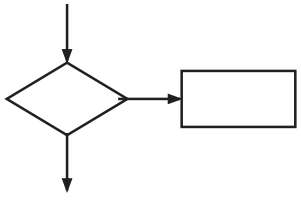
- Loop testing is classified on the basis of the types of the loops:

1. Simple Loop Testing:

- Testing performed in a simple loop is known as simple loop testing.
- Simple loop is basically a normal “for”, “while” or “do-while” in which a condition is given and loop runs and terminates according to True and False occurrence of the condition respectively.
- Simple loop testing is a type of testing is performed basically to test the condition of the loop whether the condition is sufficient to terminate loop after some point of time.

  Example:

```
while(condition)
{
   /// statement(s);
}
```

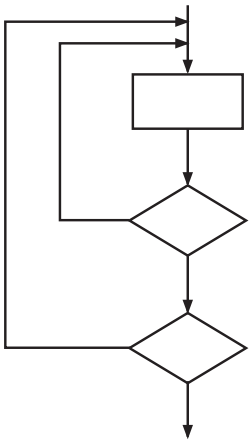


2. Nested Loop Testing:

- Testing performed in a nested loop in known as nested loop testing. Nested loop is basically one loop inside another loop.
- In nested loop there can be finite number of loops inside a loop and there a nest is made. It may be either of any of three loops i.e., for, while or do-while.

Example:

```
while(condition 1)
{
   while(condition 2)
   {
     // statement(s);
   }
}
```

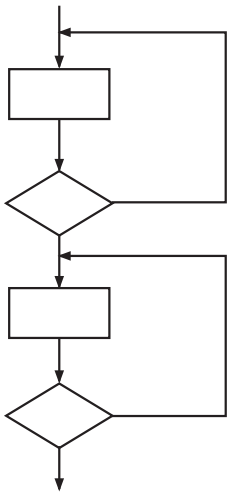


3. Concatenated Loop Testing:

- Testing applied on a concatenated loop is recognized as concatenated loop testing. Concatenated loops are loops after the loop.
- Concatenated loop is a series of loops. Nested loop means loop inside the loop and concatenated means loop is after the loop.

Example:

```
while(condition 1)
{
   // statement(s);
}
while(condition 2)
{
   // statement(s);
}
```

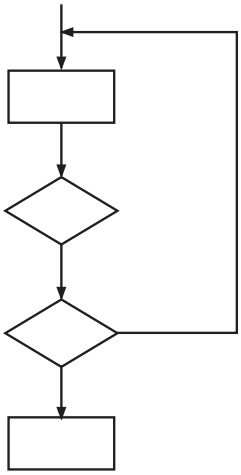


4. Unstructured Loop Testing:

- Testing applied on an unstructured loop is known as unstructured loop testing.
- Unstructured loop is the combination of concatenated and nested loops.
- In other words, it is a group of loops that are in no particular order.

Example:

```
while()
{
   for()
   {
   }
   while()
   {
   }
}
```



Example 1: In order to have loop coverage, testers should exercise the tests given as follows:

- **Test 1:** Design a test in which loop body should not execute at all (i.e. zero iteration).
- **Test 2:** Design a test in which loop control variable be negative (Negative number of iterations).
- **Test 3:** Design a test in which loop iterates only once.
- **Test 4:** Design a test in which loop iterates twice.
- **Test 5:** Design a test in which loop iterates certain number of times, say n where n < maximum number of iterations possible.
- **Test 6:** Design a test in which loop iterates one less than the maximum number of iterations.
- **Test 7:** Design a test in which loop iterates the maximum number of iterations.
- **Test 8:** Design a test in which loop iterates one more than the maximum number of iterations.

Consider the below Java code example which applies all the conditions specified:

```
public class SimpleTest
{
/* Compute total of  positive numbers in the array names as numItems and
store it in  total. For testing different count of array elements is
passed to find total*/
private int[] numbers = {5,-7,8,-1,4,1,-20,6,2,10};
public int calSum(int numItems)
{
   int total = 0;
   if (numItems<= 10)
   {
      for (int i=0; i<numItems; i = i + 1)
      {
         if (numbers[i] > 0)
         {
            total = total + numbers[i];
         }
      }
   }
   return total;
```

```
    }
    }
    public class TestPass
    {
       public void testmethod() throws Exception
       {
          SimpleTest s = new SimpleTest();
          assertEquals(0, s.calSum(0));    //Test 1
          assertEquals(0, s.calSum(-1));   //Test 2
          assertEquals(5, s.calSum(1));    //Test 3
          assertEquals(5, s.calSum(2));    //Test 4
          assertEquals(17, s.calSum(5));   //Test 5
          assertEquals(26, s.calSum(9));   //Test 6
          assertEquals(36, s.calSum(10));  //Test 7
          assertEquals(0, s.calSum(11));   //Test 8
       }
    }
```

So here in above code different type of test data is passed as input and loop is tested. For loop testing use of software or automation can be more suitable as number of iterations can be more and testing it manually is time consuming.

Example 2: Consider the below C code example to be tested, it displays first n numbers.

```
#include<stdio.h>
int main()
{
int i=1,n;
   printf("Enter n:");
   scanf("%d", &n);
   while(i<=n)
   {
      printf("%d \n",i);
      i++;
   }
   return 0;
}
```

Some Test Cases with Test Data:

Loop Coverage	Test Case	n	Output
while(i<=n)	1	4	1 2 3 4
	2	6	1 2 3 4 5 6
	3	-2	No output
	4	0	No output

Path Coverage Testing:

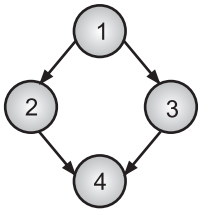
- The path coverage-based testing strategy requires us to design test cases such that all linearly independent paths in the program are executed at least once.
- A linearly independent path can be defined in terms of the Control Flow Graph (CFG) of a program. A CFG describes how the control flows through the program.
- In other words, it expresses the sequence in which the different instructions of a program get executed.
- In order to draw the control flow graph of a program, we need to first number all the statements of a program. The different numbered statements serve as the nodes of the control flow graph.
- An edge from one node to another node exists if the execution of the statement representing the first node can result in the transfer of control to the other node.
- Following image shows CFG for Sequence, Selection and Iteration respectively.

Sequence:

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

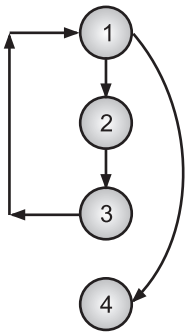
Selection:

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



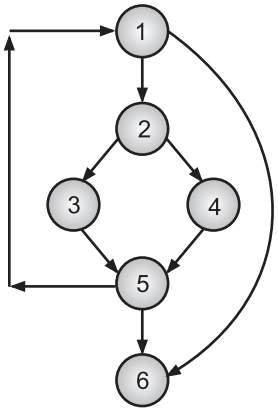
Iteration:

```
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; }
```



Path:

- A path through a program is a node and edge sequence from the starting node to a terminal node of the control flow graph of a program.
- There can be more than one terminal node in a program.
- Writing test cases to cover all the paths of a typical program is impractical.
- For this reason, the path-coverage testing does not require coverage of all paths but only coverage of linearly independent paths.

Linearly Independent Path:

- A 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 paths.
- If a path has one new node compared to all other linearly independent paths, then the path is also linearly independent.
- This is because any path having a new node automatically implies that it has a new edge.
- Thus, a path that is sub-path of another path is not considered to be a linearly independent path. The path-coverage testing does not require coverage of all paths but only coverage of linearly independent paths.

McCabe's Cyclomatic Complexity:

- Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program.
- For more complicated programs it is not easy to determine the number of independent paths of the program.
- McCabe's cyclomatic complexity defines an upper bound for the number of linearly independent paths through a program.
- This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.
- There are three different ways to compute the Cyclomatic complexity. The answers computed by the following three methods are guaranteed to agree:

  Method 1: Given a control flow graph G of a program, the Cyclomatic complexity V(G) can be computed as, V(G) = E – N + 2 where N is the number of nodes of the control flow graph and E is the number of edges in the control flow graph. For the CFG of example shown above for Euclid's gcd algorithm E = 7 and N = 6. Therefore, the Cyclomatic complexity = 7 – 6 + 2 = 3.

  Method 2: An alternative way of computing the Cyclomatic complexity of a program from an inspection of its control flow graph is as follows:

  V(G) = Total number of bounded areas + 1

  In the program's control flow graph G, any region enclosed by nodes and edges can be called as a bounded area.

  For the CFG example shown above for Euclid's gcd algorithm, from a visual examination of the CFG the number of bounded areas is 2. Therefore the cyclomatic complexity, computing with this method is also 2 + 1 = 3.

  Method 3: This is an easy way to determine the McCabe's Cyclomatic complexity. The Cyclomatic complexity of a program can also be easily computed by computing the number of decision statements of the program. If N is the number of decision statement of a program, then the McCabe's metric is equal to N + 1.

  The number of decision statements in example above for Euclid's GCD algorithm is 2, therefore Cyclomatic complexity will be 2 + 1 = 3. We can see here that all the three methods will generate same result.

Derivation of Test Cases for Path Testing:

Number of independent paths: 3

1, 6 test case (x = 1, y = 1)

1, 2, 3, 5, 1, 6 test case (x = 1, y = 2)

1, 2, 4, 5, 1, 6 test case (x = 2, y = 1)

- Path coverage is determined by assessing the proportion of execution of paths through a program exercised by the set of proposed test cases. The 100% path coverage is where every path in the program is visited by at least one test.
- Path coverage corresponds to visiting the paths through the graph. The 100% coverage is where all the paths are taken by the test cases.

$$\text{Path coverage} = \frac{\text{Total path exercised}}{\text{Total number of paths in the program}} \times 100$$

- There are three paths in above flowgraph of Euclid's GCD algorithm. So, if we consider each individual path from it then,

  Path coverage = 1/3 × 100 = 33.33%.
- It is possible to have 100% branch coverage without 100% path coverage. All branch edges are visited, without visiting all paths.
- It is possible to have 100% statement coverage without 100% branch coverage. If 100% path coverage then there can be 100% branch coverage. If 100% branch coverage is achieved the there can be 100% statement coverage.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which is a document, has a set of test data, preconditions, expected results and post-conditions?
   (a) test plan (b) test case
   (c) test strategy (d) None of the mentioned
2. The goals of test case design techniques is to,
   (a) test the functionalities of software using effective test cases.
   (b) test the features of software with the help of effective test cases.
   (c) Both (a) and (b) (d) None of the mentioned
3. Experience-based testing techniques are highly dependent on tester's,
   (a) Skills (b) Knowledge
   (c) Expertise (d) All of the mentioned
4. Which is a way of testing the external functionality of the code by examining and testing the program code that realizes the external functionality?
   (a) White-box testing (b) Black-box testing
   (c) Both (a) and (b) (d) None of the mentioned
5. Which is a statistic in software testing that reflects how much testing a collection of tests has performed?
   (a) Test case (b) Test coverage
   (c) Test report (d) Test template

6. Which of the following testing is type of white-box testing?
   (a) Specification testing (b) Structural testing
   (c) Design based testing (d) None of the mentioned
7. White-box testing techniques are,
   (a) Statement coverage testing (b) Decision coverage testing
   (c) Branch coverage testing (d) All of the mentioned
8. Which is a type of testing which requires only the source code of the product, not the binaries or executables?
   (a) Source testing (b) Static testing
   (c) Structural testing (d) Al of the mentioned
9. Which of the following is not another name of white box testing?
   (a) Clear box testing (b) Behavioral testing
   (c) Glass box testing (d) None of the mentioned
10. Which of the following is correct formula for Statement Coverage (SC)?
   (a) SC=number of executed statements/total number of statements * 100
   (b) ST=number of non-executable statements/total number of statements * 100
   (c) ST=number of non-executable statements/number of executed statements *100
   (d) None of the mentioned
11. Which of the following is correct formula for Decision Coverage (DC)?
   (a) SC=number of executed statements/total number of statements * 100
   (b) ST=number of non-executable statements/total number of statements * 100
   (c) DC=number of decision outcomes exercised/total number of decision outcomes * 100
   (d) None of the mentioned
12. Which of the following is correct statement?
   (a) Branch coverage technique measures the coverage of conditional branches whereas Decision coverage technique measures the coverage of conditional and unconditional branches.
   (b) Decision coverage technique measures the coverage of conditional branches whereas branch coverage measures the coverage of conditional and unconditional branches.
   (c) Both (a) and (b)
   (d) None of the mentioned
13. Testing applied on an unstructured loop is known as,
   (a) Structured loop testing (b) Nested loop testing
   (c) Concatenated loop testing (d) Unstructured loop testing

14. The Cyclomatic complexity in software testing is a testing metric used for measuring the _____ of a software program.
    (a) Robustness (b) Complexity
    (c) Validity (d) None of the mentioned
15. Which testing involves designing and executing teat cases and finding out the percentage (%) of the code that is covered by testing?
    (a) Code coverage testing (b) Statement coverage testing
    (c) Branch coverage testing (d) None of the mentioned

Answers

1. (b)	2. (c)	3. (d)	4. (a)	5. (b)	6. (b)	7. (d)	8. (c)	9. (b)	10. (a)
11. (c)	12. (b)	13. (d)	14. (b)	15. (a)					

Q. II Fill in the Blanks:

1. _____ testing is also known as clear box, or glass box or open box testing.
2. _____ testing takes into account the code, code structure, internal design, and how they are coded.
3. _____ coverage testing technique is used to cover all branches of the control flow graph.
4. _____testing is coverage criteria which checks whether loop body executed zero times, exactly once or more than once.
5. _____ coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing.
6. _____ coverage testing is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
7. _____ coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.
8. Testing applied on a _____ loop is recognized as concatenated loop testing.
9. In software development a number of defects come about because of _____ translation of requirements and design into program code.
10. A _____ independent path is any path through the program that introduces at least one new edge that is not included in any other linearly independent paths.
11. Path coverage ensures that every path is traversed at least _____.
12. _____ complexity is a metric that quantifies the complexity of a program.

Answers

1. White-box	2. structural	3. Branch	4. Loop
5. Code	6. Statement	7. Decision	8. concatenated
9. incorrect	10. linearly	11. once	12. Cyclomatic

Q. III State True or False:

1. In structural testing, tests are actually run on the computer with system to be tested.
2. Branch coverage considers decision point and branch coverage edges.
3. Loop testing is a type of software testing type that does not check the loops.
4. Testing performed in a simple loop is known as nested loop testing.
5. Path finding is the one of the methods to calculate branch coverage.
6. Specification-based testing technique leverages external description of software to test cases and also known as White box testing.
7. Experience-based testing techniques are highly dependent on tester's experience.
8. A path through a program is a node and edge sequence from the starting node to a terminal node of the control flow graph of a program.
9. The path coverage needs to design test cases such that all linearly independent paths in the program are executed infinite number of times.
10. McCabe's Cyclomatic complexity defines an upper bound for the number of linearly independent paths through a program.
11. Statement coverage ensures whether each and every line of the code is executed at least once during testing.
12. Loop coverage is a white-box testing technique, which involves the execution of all the statements at least once in the source code.

Answers

1. (T)	2. (T)	3. (F)	4. (F)	5. (T)	6. (F)	7. (T)	8. (T)	9. (F)	10. (T)
11. (T)	12. (F)								

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Why to use test case design techniques?
2. Differentiate between specification and structural test case design techniques.
3. What is code coverage testing?
4. Enlist the various code coverage techniques.
5. List the goals of loop coverage testing.
6. What is simple loop testing?
7. What is the difference between nested loop testing and concatenated loop testing?
8. What is the formula to find coverage in Decision Coverage (DC) testing?
9. Give the formula to find coverage in Statement Coverage (SC) testing?
10. What is Cyclomatic complexity?
11. Define path coverage.

12. Define branch coverage.
13. Define statement coverage testing.
14. Define code coverage testing.
15. Define coverage in white-box testing.

(B) Long Answer Questions:

1. What is meant by test case design techniques? Enlist them.
2. What is decision coverage testing? Explain with an example.
3. Explain statement coverage testing with an example.
4. Explain in brief, branch coverage testing.
5. Give the classification of loop testing. Explain any one in detail.
6. What is unstructured loop testing?
7. Write a note on path coverage testing.
8. Apply statement coverage testing for the following code by considering some test cases:

```
public class IfelseEx
{
   public void check_evenodd(int number)
   {
      //Check if the number is divisible by 2 or not
      if(number%2==0)
      {
         System.out.println("even number");
      }
      else
      {
         System.out.println("odd number");
      }
   }
}
```

9. Apply loop testing to following code by considering some test data:

```
class TestClass
{
   public display_no(String args[])
    {   // initialization expression
        int i = 1;
        // test expression
```

```
            while (i < 6)
            {
               System.out.println("Hello World");
               // update expression
               i++;
            }
      }
   }
```

10. Apply decision coverage testing for the following code by considering some test cases:

```
example (int x)
{
   if (x < 20)
        print("x < 20")
    else
       print("x >= 20")
}
```

11. Apply branch coverage testing for the following code:

```
READ username
READ password
IF count (username) < 8
   PRINT “Enter a valid username.”
ENDIF
IF count (Password) < 5
   PRINT “Enter a valid password”
ENDIF
IF count(username & password) < 1
   PRINT “Please Fill the Username & Password Fields”
ENDIF
ELSE
   PRINT “Login Successfully”
```

❖❖❖

CHAPTER 3

Test Cases and Test Plan

Objectives...

- To understand Test Case
- To learn Test Plan
- To prepare Test Report

3.0 INTRODUCTION

- A test case in software testing is a document, which has a set of test data, preconditions, expected results and post-conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.
- Test planning, the most important activity to ensure that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project. Test planning also defines the size of the test effort.
- IEEE standard for software test documentation defines test planning as, "a document describing the scope, approach, resources and schedule of intended testing activities".
- Test planning involves scheduling and estimating the system testing process, establishing process standards and describing the tests that should be carried out.
- The main purpose of test planning is to show whether software works correctly as per requirements.
- The goal of test planning is to take into account the important issues of testing strategy, resource utilization, responsibilities, risk and priorities.
- A test plan is a document that outlines the test strategy, objectives, timetable, estimation, deliverables, and resources needed to accomplish software testing.
- The test plan assists us in determining the amount of work required to confirm the quality of the application being tested.
- The test plan is a blueprint for conducting software testing operations as a defined procedure, which the test manager closely monitors and controls.
- A test plan is a document detailing the scope, strategy, resources and timetable of expected test activities.

- A test plan is a document which is considered to be a pre-planned blueprint of everything that is required to make the testing process complete, successful and executed on time.

3.1 TEST PLAN FOR GIVEN APPLICATION WITH RESOURCES REQUIRED

- A test plan is a detailed document that outlines the test strategy, objectives, test schedule, required resources like human resources, software resources, and hardware resources, test estimation and test deliverables.
- The test plan helps us to determine the effort needed to validate the quality of the application under test.
- The test plan serves as a blueprint to conduct software testing activities as a defined process that is fully monitored and controlled by the testing manager.

3.1.1 How to Create a Test Plan?

- Fig. 3.1 shows steps for creating a test plan in software testing:

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

Fig. 3.1: Creation of Test Plan

- Let us see steps in writing test plan in detail:
  1. **Analyze the Product or Feature to be Tested:** Before we can start creating a test plan, we need to have a deep understanding of the product or feature. For example, let's say we have just gone through a website redesign and want to test it before launch. What information do we need?

     To understand the scope, objectives, and functionality of product, talk with the designer and developer
     - Review the project documentation, (such as the Scope Of Work (SOW), project proposal or even the tasks in your project management tool).
     - Perform a product walkthrough to understand the functionality, user flow and limitations.

2. **Designing Test Strategy:** Test manager develops the Test Strategy document which defines the following:
   - Project objectives and how to achieve them.
   - The amount of effort and cost required for testing.

   More specifically, the document must detail out:
   - **Scope of Testing:** Contains the software components (hardware, software, middleware) to be tested and also those that will not be tested.
   - **Type of Testing:** Describes the types of tests to be used in the project. This is necessary since each test identifies specific types of bugs.
   - **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 business.
   - **Test Logistics:** Mentions the names of testers (or their skills) as well as the tests to be run by them. This section also includes the schedule laid out for testing and required tools.
3. **Defining Objectives:** In this step, we define the goals and expected results of test execution. Since all testing intends to identify maximum defects in an application. Test objective is the overall goal and achievement of the test execution. The objects must include:
   - A list of all software features – functionality, GUI, performance standards- that must be tested.
   - The ideal result or benchmark for every aspect of the software that needs testing. This is the standard to which all actual results will be compared with expected results.
4. **Establish Test Criteria:** Test criteria denote to standards or rules governing all activities in a testing project. The two main test criteria are:
   - **Suspension Criteria**: Defines the benchmarks for suspending all tests. For example, if QA team members find that 50% of all test cases have failed, then all testing is suspended until the developers resolve all of the bugs that have been identified so far.
   - **Exit Criteria**: Defines the benchmarks that signify the successful completion of 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. For example, 80% of all test cases must be marked successful before a particular feature or portion of the software can be considered suitable for public use.
5. **Planning Resource Allocation:** Resource planning includes all types of resources required to complete project. This step creates a detailed breakdown of all required resources for project completion. Resources include human effort,

equipment, and all infrastructures required for accurate and comprehensive testing. This also helps test managers to formulate a correctly calculated schedule and estimation for the project.

6. **Planning Setup of Test Environment:** This step refers to software and hardware setup on which software testing team run their tests. Ideally, test environments should be real devices so that testers can monitor software behavior in real user conditions. Whether it is manual testing or automation testing, nothing beats real devices, installed with real browsers and operating systems are non-negotiable as test environments.
7. **Determining Test Schedule and Estimation:** Break down the project into smaller tasks and allocate time and effort required for each activity. Then, prepare a schedule to complete these tasks in the designated time with the specific amount of effort. Creating the schedule, however, does require input from multiple perspectives:
   - Employee availability, number of working days, project deadlines, daily resource availability.
   - Risks associated with the project which has been evaluated in an earlier stage.
8. **Establish Test Deliverables:** Test deliverables imply a list of documents, tools and other equipment that must be created, provided, and maintained to support testing activities in a project.

   A different set of deliverables is required before, during, and after testing. These deliverables are given below:
   - **Deliverables required before testing** include Test Plan and Test Design.
   - **Deliverables required during testing** include Test Scripts, Simulators or Emulators (in early stages), Test Data, Error and execution logs.
   - **Deliverables required after testing** include Test Results, Defect Reports and Release Notes.

3.1.2 Test Plan Template

- The test plan template is a document that outlines the testing strategy, goals, timetable, estimates, and deliverables, as well as the resources needed for testing.
- The Test Plan assists us in determining the amount of work required to confirm the quality of the application being tested.
- The test plan is a blueprint for conducting software testing operations as a defined process that the test team closely monitors and controls.
- A test plan is a document describing the scope, approach, objectives, resources, and schedule of a software testing effort.

- A test plan identifies the items to be tested, items not be tested, who will do the testing, the test approach followed, what will be the pass/fail criteria, training needs for the testing team,
- A test plan can be defined as, a detailed document that describes the scope, approach, schedule, resources etc., of intended software product testing activities.
- Fig. 3.2 shows typical test plan template used in software testing.

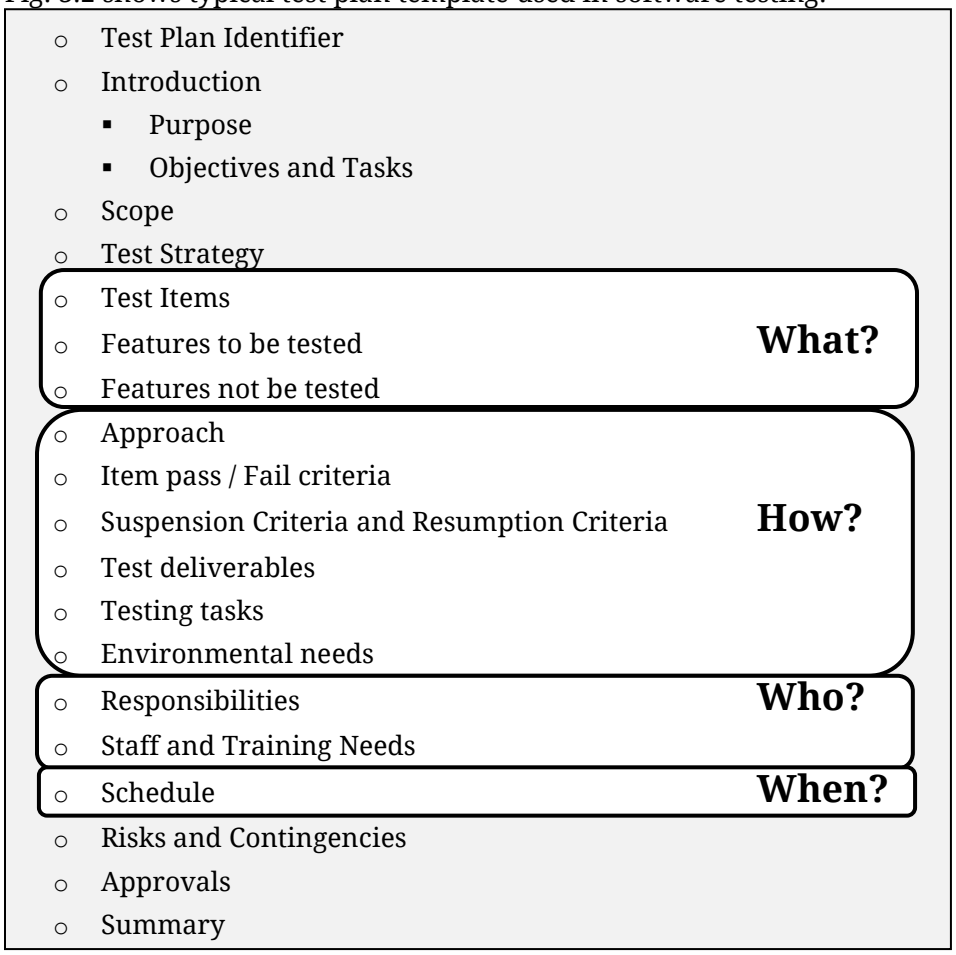


Fig. 3.2: Test Plan Template

- Test plan format/template has following fields:
  1. **Test Plan Identifier:** Uniquely identifies the test plan and may include version/release number of the document.
  2. **Introduction:** A brief introduction about the project and to the document.
  3. **Purpose:** It defines the purpose of the test plan.
  4. **Objectives and Tasks:** Objectives describes the objectives of test plan like goal of project, resource factors, project scheduling constraints, quality and cost factors. Tasks list all tasks identified by the test plan like problem reporting, post-testing etc.

5. **Scope:** It defines the features, functional or non-functional requirements of the software that will be tested.
6. **Testing Strategy:** It describes overall approach to testing.
7. **Test Items:** A test item is a software item that is the application under test.
8. **Software Risk Issues:** Software project risks and assumptions.
9. **Features to be tested:** A feature that needs to tested on the test ware.
10. **Features not to be tested:** Identify the features and the reasons for not including as part of testing.
11. **Approach:** It shows details about the overall approach to testing.
12. **Item Pass/Fail Criteria:** Documented whether a software item has passed or failed its test.
13. **Suspension Criteria and Resumption Requirements:** It contains suspension criteria and resumption requirements.
14. **Test Deliverables:** The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports.
15. **Remaining Test Tasks:** All remaining tasks for planning and executing the testing.
16. **Environmental Needs:** Defining the environmental requirements such as hardware, software, OS, network configurations, tools required.
17. **Staffing and Training Needs:** Captures the actual staffing requirements and any specific skills and training requirements.
18. **Responsibilities:** Lists the roles and responsibilities of the team members.
19. **Schedule:** States the important project delivery dates and key milestones.
20. **Planning Risks and Contingencies:** Planning for risks and contingencies.
21. **Approvals:** Captures all approvers of the document, their titles and the sign off date. All the individuals and parties directly and indirectly associated with the testing process give their agreements in the form of signed approvals.
22. **Glossary:** Summary or definitions of used in testing.

3.2 WRITE TEST CASE FOR GIVEN APPLICATION

- A test case is a set of actions executed to verify a particular feature or functionality of the software application.
- A test case comprises test steps, test data, precondition, post-condition developed for specific test scenario to verify any requirement.
- The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

- For the test execution, test case acts as the starting point, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point which is also known as execution post-condition.
- Test cases help guide the tester through a sequence of steps to validate whether a software application is free of bugs, and working as required by the end user.
- The components of a test case are a particular test case format, usually, comprises of following parameters:

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.

3.2.1 Test Case Template

- A test case is a sequence of actions and observations that are used to verify the desired functionality.
- A good test case helps to identify problems in the requirements or design of an software application.
- The test case templates are a set of conditions or variables made according to the requirements provided; after the execution of these test cases, a tester will decide whether a system satisfies the requirements or works correctly.
- A test case template is a well-designed document for developing and better understanding of the test case data for a particular test case scenario.
- As per organization, test case template can vary. Some sample test case templates are given below:

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				

Example 1: Let us say that we need to check an input field that can accept maximum of 20 characters. While developing the test cases for the above scenario, the test cases are documented the following way. In the below example, the first case is a Pass scenario while the second case is a Fail.

Scenario	Test Step	Expected Result	Actual Outcome
Verify that the input field that can accept maximum of 20 characters.	Login to application and key in 20 characters.	Application should be able to accept all 20 characters.	Application accepts all 20 characters.

Contd...

Verify that the input field that can accept maximum of 21 characters.	Login to application and key in 21 characters.	Application should NOT accept all 21 characters.	Application accepts all 21 characters.

If the expected result doesn't match with the actual result, then we log a defect. The defect goes through the defect life cycle and the testers address the same after fix.

Example 2:

Project Name	Exam portal								
Module Name	Login Functionality								
Created By	Ajay								
Created Date	dd-mm-yyyy								
Executed By	ABC								
Executed Date	dd-mm-yyyy								
Test Case Id	**Test Case Description**	**Pre Steps**	**Test Step**	**Precoditions**	**Test Data**	**Expected Resul**	**Actual Result**	**Status**	**Comments**
Test the login Functionality in Exam Portal	Verify login functionality with valis username & password		Navigate to login page			Appears Home Page	As excpected	Pass	
			Enter valid username	Valid username	abc78@gmail.com	Credential can be entered	As excpected	Pass	
			Enter valid password	Valid password	india@123	User logged	As excpected	Pass	
Test the login Functionality in Exam Portal	Verify login functionality with valis username & invalid password		Navigate to login page	Valid username	abc78@gmail.com	Credential can be entered	As excpected	Pass	
			Enter valid username	InValid password	ind@123	Credential can be entered	As excpected	Pass	
			Enter invalid password			User logged	Unsuccessful Login	Fail	

Example 3: ATM Machine Test Cases. Primary and generic positive and negative test cases for the ATM machine. While testing ATM machine we have to keep following things in mind:

- Performance
- Accuracy
- Reliability

If we want to achieve all the above points for ATM Machine, then we have to test each separate component, after that integration testing after assembling all the elements and finally need to perform performance testing.

ATM Machine UI Testing:

- Check in the screen all labels button, links & images are appearing correctly.
- Check whatever written on the screen is visible.
- Check the application UIO is responsive.

- Check the ATM Machine is a full touch screen, or it also supports Keyboard and Touch screen both the functionality.

ATM Machine Positive Test Cases:

- Check the ATM card is as per the specification document.
- After entering the Debit/Credit card in the card, reader users should be able to select language and operation like withdrawal, language change, mini statement, and other options.
- When an ATM card is entered in the card reader, it should verify the card.
- Check during any transaction the ATM Machine accepts card and Pin details.
- Check after successfully enter the pin and complete the process, the user should be able to take out the money.
- After taking out the money, the money receipt should also print.
- After successfully withdraw the amount, the user should be log out from the sessions.
- Check if the user wants to print the money receipt (mini statement), it should be done by following the menu options.
- If the user enters more amounts then the account balance, then the user should get an error message.
- The ATM should have a waiting period between user session log out and active another account.
- In the ATM Machine, the user should be able to Use a Master card, Visa card, and Rupay Card.
- Verify after the transaction, the printed slip has the correct information or not.
- Verify the entered pin is encrypted or not.
- Verify the touch functionality is working correctly or not.
- Check whether the ATM is providing all types of accounts to do operations like Savings and the correct account.

Negative Test Cases:

- Check the ATM Machine can accept the cards and pin or not.
- If a user enters an invalid pin, then an error message should be displayed.
- If allowed bank ATM card is entered, then only the user able to do the operations.
- Check is the ATM Machine can find the wrong pin or not.
- Check if the card is entered in the wrong way, the machine should find that and display an error message.
- Check if the user three times the wrong password, then the account should be locked.
- If the user has a lack of money, then he should receive a warning message.

- If the user inserted an expired card, then the user should not be able to perform any action.
- If the user inserted less than 100 amount, then the amount should not out from the ATM Machine.
- If the ATM Machine has dispatched the money, then the money should not again enter into the ATM.
- The machine does not accept either Visa or MasterCard or both debit/credit cards.
- If the user enters the wrong denominations, then a warning message should display.
- If the user has entered more than the daily limit amount, then the transaction should be canceled, and a warning message should be displayed on the screen.
- The transaction should be canceled if the user clicks on the Cancel button.
- Check an error message is a display or not when the ATM does have the currency on it.
- Check whether an error message is displayed or not when there is some network issue.
- Check after the money release it is asking or not for the user confirmation to print the transaction receipt.
- Check during the transaction if there is some power failure or network issue comes then the transaction should be marked as nulled and no amount should be dispatched.

Example 4: Test Case for SMS (Short Message Service). Here are the test cases for the SMS application.

- Send to the invalid phone number and verify if it shows error or success message.
- Send it to the valid phone number and verify if the message gets sent.
- Send to the valid phone number and verify if the receiver able to open it.
- Send it to the valid phone number and verify if the receiver can read the contents of it.
- Check the Message text editor character limit.
- Check the Message text editor for dictionary support.
- Check the Message editor for the auto-correct option.
- Check the Message editor for the template support option.
- Check the Message editor for the language support.
- Check if the Message editor for the type checker.
- Does the Message text accept emoticons in text format or image format?
- Does the Message app get a notification as sending failed?

- Does the Message app get a notification if the message reaches?
- Does the Message app support multiple languages to type and view on the screen?
- Does the Message app keep the written text as a draft if the network is not available?
- Does the Message app allow sending a text to multiple people?
- Check the method to delete the Message.
- Check the method to forward the SMS.
- Check the method to send SMS to more than one person.
- Check the method to set the message as read.
- Check the method to set the message as unread.
- Check if the app allows deleting multiple messages.
- Check if the app allows the deletion of all messages at the same time.
- Check if its content is allowed to be blank.
- Check if its content is allowed to add more than the character limit in an editor.
- Check if its content is allowed to add exactly like the character limit in an editor.
- Check if its editor opens default with English content.
- Check if its editor opens default with multiple language support.
- Check if its app shows the time of the SMS.
- Check if its app shows the phone number of the sender.
- Check if its app shows the short content of the SMS in the notification.

3.2.1 Writing Test Case

- A test case refers to the sequence of actions required to verify a specific feature or functionality.
- Essentially, the test case details the steps, data, prerequisites, and post-conditions necessary to verify a feature.
- Test case contains input, execution, and expected output. Basically, a test case tells what to do, how to do it, and what results are acceptable.
- Test case writing is an activity which has a great impact on the testing and this makes test cases an important part of the test execution process.

Objective of Writing Test Cases:

1. To validate specific features and functions of the software.
2. To guide testers through their day-to-day hands-on activity.
3. To provide a blueprint for future projects.
4. To help detect usability issues and design gaps early on.

- The steps for writing test cases are given below:

Step 1: **Review the Requirements:** Before writing test cases, the requirements need to be reviewed to ensure that they reflect the requirements' quality factors.

Step 2: **Write a Test Plan:** A software test plan is a document that describes the objectives, scope, approach and focus of a software testing effort. The process of preparing a test plan is a useful way to think through the efforts needed to validate the acceptability of a software product. The completed document will help the whole team understand the "why".

Step 3: **Identify the Test Suite:** After the test plan has been completed and the requirements are "testable," an effective way of transforming the requirements to test cases is to first design the test suites. A test suite, also known as a validation suite, is a collection of test cases that are intended to be used as input to a software program to show that it has some specified set of behaviors. Test suites are used to group similar test cases together. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps and descriptions of the following tests.

Step 4: **Name the Test Cases:** Having an organized system test suite makes it easier to list test cases because the task is broken down into many small, specific subtasks. There may be some list items or grid cells that really should be empty. A name of each test case should be a short phrase describing a general test situation. Use distinct test cases when different steps will be needed to test each situation. One test case can be used when the steps are the same and different input values are needed.

Step 5: Write Test Case Descriptions and Objectives: For each test case, write one or two sentences describing its purpose and objectives. The description should provide enough information so that we could come back to it after several weeks and recall the same ad hoc testing steps that you have in mind now. Later, when we actually write detailed steps in the test case, any team member can carry out the test the same way that we intended. The act of writing the descriptions forces us to think a bit more about each test case. When describing a test case, we may realize that it should actually be split into two test cases, or merged with another test case. Again, make sure to note any requirements problems or questions that you uncover.

Step 6: **Create the Test Cases**: Next step is to write the test case steps and specify test data. This is where the testing techniques can help you define the test data and conditions.

Step 7: **Review the Test Cases:** The main reason of the reviewing test cases is to increase test coverage and therefore software product quality.

3.3 PREPARE TEST REPORT FOR TEST CASES 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 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.
- The test reporting process is a process to collect data, analyze the data, supplement the data with metrics, graphs and charts and other pictorial representations which help the developers and users interpret that data.
- A test report is a document which includes summary of testing objectives, activities results and status.
- A test report is created by testing team which focuses on product quality and decides whether a product, feature or defect resolution is on track for release.
- This document serves as a physical log that records exactly what code was tested, in what system configuration the code was tested on and any bugs that came up during testing.
- It is used to help product manager, analysts, testing team and developers.
- Test Report is a document which contains a summary of all test activities and final test results of a testing project.
- Based on the test report, stakeholders can evaluate the quality of the tested product and make a decision on the software release.

Test Report Template is given below:

Module	Scenarios	Sub Levels	Complexity	Tester	Execution Date	Status	Defect_Id	Severity

Example: Check Login Page.

Module	Scenarios	Sub Levels	Complexity	Tester	Execution date	Status	Defect_Id	Severity
Admin	Login Page	Login	Medium	Tester A	30-5-2015	Pass		1-High
Admin	Product management	1. Add Product. 2. Delete Product. 3. Verify list of Products.	Medium	Tester B	10-6-2015	Pass	1011: Can't View Product List	
Admin	Customer management	1. Create Profile. 2. Update Profile.	Medium	Tester A	20-6-2015	Fail		2-Medium

- Testing requires constant communication between the test team and other teams like the development team. Test reporting is a means of achieving this communication.
- There are two types of reports or communication that are required i.e., test incident reports and test summary reports (also called test completion reports).

3.3.1 Test Incident Report

- An incident report provides a formal mechanism for recording software incidents, defects, enhancements and their status.
- It is a communication that happens through the testing cycle, when defects are encountered.
- A test incident report is nothing but an entry made in the defect repository. A test incident report observes the defect has a unique ID and this ID is used to identify the incident.
- The high impact test incidence (defects) is highlighted in the test summary report.
- Fig. 3.3 shows the IEEE template for a Test Incident Report.

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

Contents

1. Incident Summary Report Identifier
2. Incident Summary
3. Incident Description
   - 3.1 Inputs
   - 3.2 Expected Results
   - 3.3 Actual Results
   - 3.4 Anomalies
   - 3.5 Date and Time
   - 3.6 Procedure Step
   - 3.7 Environment
   - 3.8 Attempts to Repeat
   - 3.9 Testers
   - 3.10 Observers
4. Impact
5. Investigation
6. Metrics
7. Disposition

Fig. 3.3: IEEE Template for a Test Incident Report

- The **Incident Summary Report Identifier** uses the organization's incident tracking numbering scheme to identify this incident and its corresponding report.
- The **Incident Summary** is the information that relates the incident back to the procedure or test case that discovered it.
- The **Author** of the incident report should include enough information so that the readers of the report will be able to understand and replicate the incident.
- Sometimes, the test case reference alone will be sufficient, but in other instances, information about the setup, environment, and other variables is useful.
- Following table describes the subsections that appear under Incident Description:

Inputs	Describes the inputs actually used (e.g., files, keystrokes, etc.).
Expected Results	Comes from the test case that was running when the incident was discovered.
Actual Results	Actual results are recorded here.

Contd...

Anomalies	How the actual results differ from the expected results.
Date and Time	The date and time of the occurrence of the incident.
Procedure Step	The step in which the incident occurred.
Environment	The environment that was used (e.g., system test environment or acceptance test environment etc.).
Attempts to Repeat	How many attempts were made to repeat the test?
Testers	Who ran the test?
Observers	Who else has knowledge of the situation?

- The **Impact** of the incident report form refers to the potential impact on the user, so the users or their representative should ultimately decide the impact of the incident. The impact will also be one of the prime determinants in the prioritization of bug fixes, although the resources required to fix each bug will also have an effect on the prioritization.
- The **Investigation** of the incident report explains who found the incident and who the key players are in its resolution. Some people also collect some metrics here on the estimated amount of time required to isolate the bug.
- The **Metrics** of the incident report can be used to record any number of different metrics on the type, location and cause of the incidents.
- The **Disposition (Status)** shows the current status of all incidents with log of incidents. In a good defect tracking system, there should be the capability to maintain a log or audit trail of the incident as it goes through the analysis, debugging, correction, re-testing and implementation process.

3.3.2 Test Summary Report

- Test summary report summarizes the completion of the respective test activities and deliverables.
- Test summary report also supporting documentation that provides documented evidence that the tests are executed and each of the test cases are evaluated.
- At the completion of a test cycle, a test summary report is produce or developed. The purpose of the test summary report is to summarize the results of the testing activities and to provide an evaluation based on these results.
- The summary report provides advice on the release readiness of the product and should document any known anomalies or shortcomings in the software product.
- Test summary report allows the test manager to summarize the testing and to identify limitations of the software and the failure likelihood. There should be a test summary report that corresponds to every test plan.

- The test summary report shown in Fig. 3.4.

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

- Various parameters in test summary report template are explained below:
  1. **Test Summary Report Identifier:** It is a unique number that identifies the report and is used to place the test summary report under configuration management.
  2. **Summary:** It summarizes what testing activities took place, including the versions/releases of the software, the environment and so forth. It also supply references to the test plan, test-design specifications, test procedures and test cases.
  3. **Variances:** It describes any variances between the testing that was planned and the testing that really occurred.
  4. **Comprehensive Assessment:** We should evaluate the comprehensiveness of the testing process against the criteria specified in the test plan. These criteria are based upon the inventory, requirements, design, code coverage or some combination thereof.
  5. **Summary of Results:** It summarizes the results of testing here. Identify all resolved incidents and summarize their resolution. It identify all unresolved incidents and also will be contain metrics about defects and their
  6. **Evaluation:** It provides an overall evaluation of each test item, including its limitations. This evaluation should be based upon the test results and the item pass/fail criteria.

7. **Recommendations:** It is test manager's job to make recommendations based on what they discover during the testing.
8. **Summary of Activities:** They summarize the major testing activities and events. Summarize resource consumption data; for example, total staffing level, total machine time, and elapsed time used for each of the major testing activities. Summary of activities is important to the test manager, because the data recorded here is part of the information required for estimating future testing efforts.
9. **Approvals:** They specify the names and titles of all persons who must approve this report. Provide space for the signatures and dates. Ideally, we would like the approvers of this report to be the same people who approved the corresponding test plan, since the test summary report summarizes all of the activities outlined in the plan. By signing this document, the approvers are certifying that they concur with the results as stated in the report, and that the report, as written, represents a consensus of all of the approvers.

Case Study for verify the functionality of Flipkart Login Page.

Test Plan:

Parameter	Description
Test Plan Identifier	TP_1001 Version 001
Introduction	**Objectives:** Check the functionality of Flipkart login page. **Scope:** • Verify the functionality of Flipkart login page. • Prepare Test Cases. • Prepare Test Steps. • Prepare Test data. • Define preconditions and post conditions. • Obtain Status. **Resources required:** • **Human Resources:** Test Manager, Test Designer, System Tester, DB Manager, Designer, Implementers. • **System Resources:** Client test PC, Web server and application server interface, Database server interface. **Test Scenario:** • Minimum and Maximum lengths should be set for all the text boxes.

	• Password should be displayed in masked format rather than showing actual text format. • Login credentials in UPPER case should not be treated as invalid. • Validation message should be shown when special. characters are entered in the username field, or when invalid username and/or password is entered or the fields are left blank. • Reset button should clear data from all the text boxes in the form. • Login credentials, especially password, should be stored in database in encrypted format.
Test Items	Module Verify Login Page.
Featured to be tested	• Verify user is able to login with valid username and valid password. • Verify if a user can't login with a valid username and invalid password. • Verify if a user can't login with a invalid username and valid password. • Verify if a user can't login with an invalid username and invalid password.
Featured not to be tested	"About us" on website.
Approach	Input Test data with various test cases.
Item Pass/Fail	• For valid username and valid password Status will be Pass • For Remaining three negative conditions Status will be Fail
Suspension Criteria	If 50% of test cases fails then suspended the testing process.
Test Deliverables	Includes: • Test Plan, Test Suite, Test cases, Test data, Test Results, Defect Report, Test Evaluation Report.
Testing Tasks	Describes dependencies between tasks and resources needed.
Environmental Needs	Ensures test environment and assets are managed and maintained.

Contd...

Responsibilities	**Test Manager:** Overall management and technical directions. **Test Designer:** Identifies, prioritizes and implements test cases. **System Tester:** Executes the test. **Database Manager:** Ensures test data environment and management. **Designer:** Identifies and defines test classes and test packages. **Implementer:** Creates test classes and test packages.
Staffing Needs	Additional Human Resources required.
Schedule	**Test Planning:** Effort (1 day), **Start Date:** 12 April, **End Date:** 13 April. **Test Design:** Effort (5 hrs), **Start Date:** 13 April, **End Date:** 13 April. **Test Development:** Effort (1 day), **Start Date:** 14 April, **End Date:** 15 April. **Test Execution:** Effort (8 hrs.), **Start Date:** 16 April, **End Date:** 16 April. **Test Evaluation:** Effort (2 hrs.), **Start Date:** 17 April, **End Date:** 17 April.
Risks	Assessment of risks.
Approvals	Respective Authorities names.

Test Cases and Test Report:

Project Name	Flipcart
Module name	Login
Created by	Rajvardhini
Creation Date	MM-DD-YY
Reviewd by	Test Lead/Peer
Reviewed Date	MM-DD-YY

Test Scenrio id	Test Scenario Description	Test case id	Test case Description	Test Steps	Preconditions	Test Data	Postconditions	Expected Result	Actual Result	Status
TS_001	Verify the functionality of Flipcart login page	TC_FC_Login_001	Enter valid username and valid password	1. Enter valid username 2. Enter valid password 3. Click on Login button	valid URL Test data	username : Flipcart@gmail.com password : p@sswoRD	User should able to see the home page	Successful login	Home Page Display	Pass
TS_002	Verify the functionality of Flipcart login page	TC_FC_Login_002	Enter valid username and invalid password	1. Enter valid username 2. Enter invalid password 3. Click on Login button	valid URL Test data	username : Flipcart@gmail.com password : xxxxxxx	Invalid Username or Password	A popup message box to show an error "Invalid user	A popup message box to show an error	Fail
TS_003	Verify the functionality of Flipcart login page	TC_FC_Login_003	Enter invalid username and valid password	1. Enter invalid username 2. Enter valid password 3. Click on Login button	valid URL Test data	username : xxxxxx@gmail.com password : p@sswoRD	Invalid Username or Password	A popup message box to show an error "Invalid user name or invalid password	A popup message box to show an error "Invalid user name or invalid password	
TS_004	Verify the functionality of Flipcart login page	TC_FC_Login_004	Enter invalid username and invalid password	1. Enter invalid username 2. Enter invalid password 3. Click on Login button	valid URL Test data	username : xxxxxxx@gmail.com password : xxxxxxx	Invalid Username or Password	A popup message box to show an error "Invalid user name or invalid password	A popup message box to show an error "Invalid user name or invalid password	

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which of the following is not part of the Test document??
   (a) Test Case (b) Test Strategy
   (c) Requirements Traceability Matrix (RTM)
   (d) Project Initiation Note (PIN)
2. Which following term is used to define testing?
   (a) Evaluating deliverable to find errors (b) Finding broken code
   (c) A stage of all projects (d) None of the mentioned
3. Which of the following document detailing the objectives, resources, and processes for a specific test for a software or hardware product?
   (a) Test Technique (b) Test Strategy
   (c) Test Case (d) Test Plan
4. Which of the following report outlines the summation of software testing activities and final testing results?
   (a) Test Incident (b) Test Summary
   (c) Test Planning (d) None of the mentioned
5. Which of the below is not a part of the Test Plan?
   (a) Schedule (b) Risk
   (c) Incident reports (d) Entry and Exit criteria
6. Which test document is used to define the Exit Criteria of testing?
   (a) Defect Report (b) Test Summary Report
   (c) Test Case (d) Test Plan
7. A set of inputs, execution preconditions and expected outcomes is known as a,
   (a) Test Plan (b) Test Case
   (c) Test Document (d) Test Suite
8. Which is a translator (translates source code to object/target code)?
   (a) Requirement ID (b) Test case ID
   (c) Bug ID (d) None of the mentioned
9. Which is a document detailing the objectives, resources, and processes for a specific test for a software or hardware product?
   (a) Test Technique (b) Test Strategy
   (c) Test Case (d) Test Plan
10. Which is a document generated after the culmination of software testing process, wherein the various incidents and defects are reported by the testing team and to take important steps to resolve these issues?

(a) Test Incident Report (b) Test Summary Report
(c) Test Strategy Report (d) Test Planning Report

11. A test report is an organized summary of,
    (a) testing objectives (b) testing activities
    (c) testing results (d) All of the mentioned
12. What is the main task of test planning?
    (a) Measuring and analyzing results (b) Evaluating exit criteria and reporting
    (c) Determining the test approach (d) Preparing the test specification
13. What is a document containing an organized list of test cases for different test scenarios that check whether or not the software has the intended functionality?
    (a) Test Case Template (b) Test Summary Template
    (c) Test Strategy Template (d) Test Plan Template
14. Which following is a set of guiding principles that determines the test design & regulates how the software testing process will be done.?
    (a) Test Technique (b) Test Strategy
    (c) Test Case (d) Test Plan

Answers

1. (d)	2. (a)	3. (d)	4. (b)	5. (c)	6. (d)	7. (b)	8. (c)	9. (d)	10. (a)
11. (d)	12. (c)	13. (a)	14. (b)						

Q. II Fill in the Blanks:

1. _____ is a detailed document that outlines the test strategy, objectives, test schedule, required resources, test estimation and test deliverables.
2. Test Plan ensures about the _____ requirements that guided its design and development.
3. _____defines the features, functional or non-functional requirements of the software that will be tested.
4. _____ is a set of actions executed to verify a particular feature or functionality of the software application.
5. _____ is a document which includes summary of testing objectives, activities results and status.
6. The test plan is a blueprint for conducting software testing _____.
7. Test _____ report is a formal document that summarizes the results of all testing efforts for a particular testing cycle of a software.
8. Defect _____ report is a description of an incident (variation or deviation observed in system behavior from what is expected) observed during testing.
9. The purpose of test _____ to show whether software works correctly as per requirements.

10. The objective of the Test _____ is to provide a systematic approach to the software testing process in order to ensure the quality, traceability, reliability and better planning.
11. Each test case should be represented by a unique _____.
12. The test case _____ are a set of conditions made according to the requirements.

Answers

1. Test Plan	2. requirements	3. Scope	4. Test case
5. Test report	6. operations	7. summary	8. incident
9. planning	10. Strategy	11. ID	12. templates

Q. III State True or False:

1. Testing process describes Evaluating deliverable to find errors.
2. Customer gets a 100% bug-free product after testing.
3. A test report is a document which includes summary of testing objectives, activities results and status.
4. Test planning describes the scope, approach, resources and schedule of intended testing activities.
5. Test suites are used to group similar test cases together.
6. A test plan is a detailed document is considered to be a pre-planned blueprint of everything that is required to make the testing process complete.
7. Test criteria denote to standards or rules governing all activities in a testing the software product.
8. The test incident report serves as a blueprint to conduct software testing activities as a defined process.
9. Test planning describes the test strategy, objectives, schedule, estimation, deliverables and resources required to perform testing for a software product.
10. Testing strategy describes overall approach to testing.
11. After the starting of a testing, a test summary report is produce.
12. Test cases contain a sequence of steps to validate whether a software application is free of bugs and working as required by the end user.
13. A Test Scenario is any functionality that can be tested, also called as Test Condition.
14. A test report is any description of the status of a software test product.

Answers

1. (T)	2. (F)	3. (T)	4. (T)	5. (T)	6. (T)	7. (T)	8. (F)	9. (T)	10. (T)
11. (F)	12. (T)	13. (T)	14. (T)						

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. What is test plan?
2. What is meant by exit criteria in test plan?
3. What is test case?
4. List the general components of test plan.
5. What is test scenario?
6. What is test report?
7. What is a suspension criterion?
8. What is test strategy?
9. Define test summary report.
10. What is test incident report?
11. Define test planning.
12. List test deliverable produced.

(B) Long Answer Questions:

1. What are components of test plan? Explain in detail.
2. What is test case? Explain general test case template briefly.
3. Design test cases for an application “login page activity”.
4. Explain test steps with example.
5. What is test scenario? What is the difference between test cases and test scenario?
6. Explain parameters of test report.
7. How to prepare test plan?
8. How to write test cases for given application.
9. Design a test report for Web application.
10. Prepare test plan to check login credentials.
11. Explain test strategy in detail.
12. With the help of diagram describe test summary report.
13. Explain test incident report in detail.
14. How to prepare test report for test cases executed? Describe in detail.

CHAPTER 4

Defect Report

Objectives...

- To understand Concept of Defect
- To learn Defect Life Cycle
- To Write Defect Report

4.0 INTRODUCTION

- A software defect arises when the expected result don't match with the actual results. It can also be error, flaw, failure, or fault in a computer program.
- The variation in the expected and actual results is known as defects. A defect is an error or a bug in the application.
- A defect is a variance from specification/expectation.
- Defect can be defined as "an inconsistency in the behavior of the software." **OR** "any significant, unplanned event that occurs during testing that requires subsequent investigation and/or correction. Defects are raised when expected and actual test results differ".
- When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect.
- In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.
- A software bug arises when the expected result don't match with the actual results. It can also be error, flaw, failure, or fault in a computer program.
- Root causes of defects are:
  1. Requirements are not defined clearly by customer.
  2. Designs are wrong and not implemented properly.
  3. People are not trained for work in collecting requirements.
  4. Processes used for product development, testing are not capable. They are not able to deliver right software product.

4.1 DEFECT LIFE CYCLE

- A defect is an error or a bug in the application. While designing and building the software programmer can make mistakes or error. These mistakes are flaws in the software. These are called defects.
- When expected results are different from the actual results in testing a software application or product then it results into a defect.
- While testing of an application to find as many defects as possible, it is the responsibility of a tester.
- It ensures a quality of product which will reach the customer's requirements. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.
- Defect life cycle also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime.
- It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.
- It is a cycle of defects from which it goes through covering the different states in its entire life.

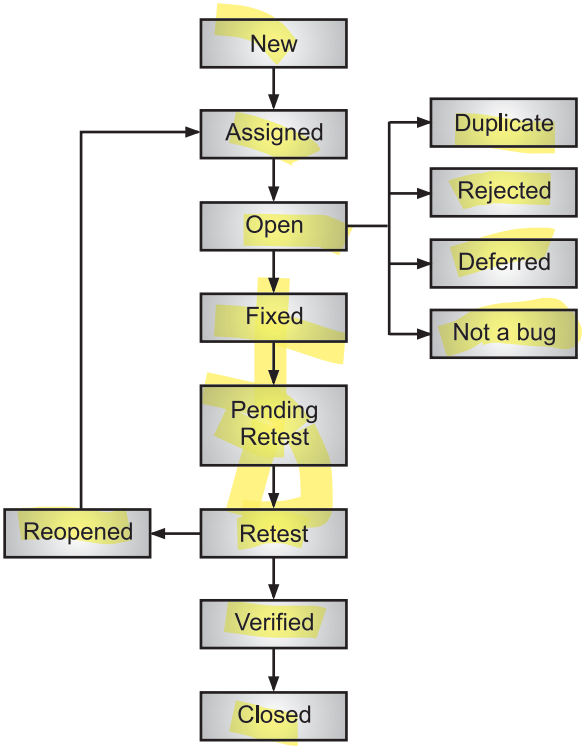


Fig. 4.1: Defect/Bug Life Cycle

- This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect assuring that it won't get reproduced again.
- The actual workflow of a Defect Life Cycle with the help of a simple diagram as shown in Fig. 4.1.
- Defect life cycle is a cycle in which a defect goes through during its lifetime. The defect life cycle starts when defect is found and ends when a defect is closed, after ensuring it's not reproduced.
- Defect life cycle is the representation of the different states of a defect which it attains at different levels during its lifetime.
- The defect/bug has following states in its life cycle:
  1. **New:** Whenever any new defect is found in application, it is defined as a 'New' state, and validations and testing are performed on this defect in the later stages of the Defect Life Cycle.
  2. **Assigned:** Newly created defect is assigned to the development team to work on the defect. This is assigned by the project lead or the manager of the testing team to a developer. It's state is now defined as "assign"
  3. **Open:** Developer analyzes the defect and works on it and fix it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the below four states namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
  4. **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then it is marked as "Fixed".
  5. **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
  6. **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
  7. **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
  8. **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
  9. **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".
- A few more state includes:
  1. **Rejected:** If the defect is not considered a genuine defect by the developer then it is marked as "Rejected" by the developer.

2. **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect then the status of the defect is changed to 'Duplicate' by the developer.
3. **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, he can change the status of the defect as 'Deferred'.
4. **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

4.2 DEFECT CLASSIFICATION

- Defects may be a result of wrong implementation of requirements, something missing that is required by the customer or putting something more (extra) than required.
- Defects may be classified in different ways under different schemes. Fig. 4.2 shows classification of defect.

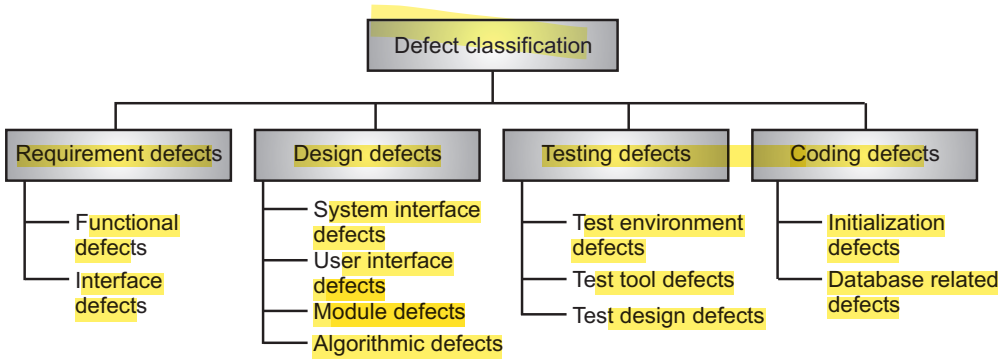


Fig. 4.2: Classification of Defects

- Let us see above defects in detail:

1. Requirement Defects:

- These defects arise in a product when one fails to understand what is required by the customer.
- Requirement defects may be due to customer gap, where the customer is unable to define his requirements or producer gap, where developing team is not able to make a product as per requirements.
- Requirement-related defects may be further classified as given below:

(i) Functional defects are mainly about the functionalities present/absent in the applications which are expected/not expected by the customer.

(ii) Interface defects talk about various interfaces essential as per the requirement statement. Generally, these defects may be due to user interface problems, problems with connectivity to other systems including hardware, etc.

2. Design Defects:

- These defects generally refer to the way of design creation or its usage while creating a software product.
- The customer may or may not be in a position to understand these defects, if structures are not correct.
- They may be due to problems with design creation and implementation during SDLC (Software Development Life Cycle).
- Design related defects may be classified as follows:
  - **(i) Algorithmic defects** may be introduced in a product if designs of various decisions are not handled correctly.
  - **(ii) Module interface defects** are about communication problem between various modules. If one module gives some parameters which are not recognized by another, it creates module-interface defects.
  - **(iii) User interface defects** may be a part of system - interface defects where the other system working with the application is a human being. User-interface defects may be a part of navigation, look and feel type of defects which affect usability of an application.
  - **(iv) System interface defects** may be generated when application communication with environmental factors is hampered. System may not be able to recognize inputs coming from the environment or may not be able to give outputs which can be used by the environment.

3. Coding Defects:

- These defects may arise when designs are implemented wrongly. If there is absence of development/coding standards or if they are wrong, it may lead to coding defects.
- Some coding defect are given below:
  - **(i) Variable declaration/initialization** defects arise when variables are not initialized properly, or variables are not declared correctly. These types of defects refer to wrong coding practices which may arise due to wrong standards of development.
  - **(ii) Commenting/Documentation defects**, coding also needs adequate commenting to make it readable and maintainable in future.
  - **(iii) Database related defects** may occur when a database is not created appropriately or it is not optimized. It may be a part of design defects if database design is wrong or it may be a coding defect if database is not implemented correctly as per design.

4. Testing Defects:

- These defects are introduced in an application due to wrong testing or defects in the test artifacts leading to wrong testing.

(i) **Test design defects** refer to defects in test artifacts. These can be defects in test plans, test scenarios, test cases and test data definition which can lead to defect introduction in software if it is not handled correctly.

(ii) **Test tool defects** generally, assumed that there are no defects in a test tool. Any defect introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual tests as against automated tools.

(iii) **Test environment defects** may arise when test environment is not set properly. Test environment may be comprised of hardware, software, simulators and people doing testing. If test environment definition and reality are mismatched, it may lead to defects. Test environment may include operator training also.

4.3 WRITE DEFECT REPORT

- Reporting defects is an important step for an organization in software development life cycle to establish capability of processes and for initiating corrective and preventive actions along with corrections, if problems are found.
- 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.
- A defect report describes defects in software and documentation. Defect reports have a priority rating that indicates the impact the problem has on the customer.
- A defect report documents an anomaly discovered during testing. Defect report also known as Bug Report, is a document that identifies and describes a defect detected by a tester.
- 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.
- A bug report in Software testing is a detailed document about bugs found in the software application.
- Defect/bug report contains each detail about bugs like description, date when bug was found, name of tester who found it, name of developer who fixed it, etc. Bug report helps to identify similar bugs in future so it can be avoided.
- A good defect report should:
  1. Give sufficient and high-quality information to reproduce and fix the defect.
  2. Be well written and simple to understand.
  3. Enable stakeholders to make wise decisions about which defects to fix.
- A defect report or defect template lists an anomaly discovered during testing. Each and every organization must have a defect template which captures a defect data.
- Different software management tools offers defect templates, sometimes test case template is extended as defect template.

- A defect report includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc.
- The general sample defect template is shown in Fig. 4.3.

DEFECT REPORT

Defect ID : _____ **Defect Name :** _____

Software/Project Name : _____ **Module Name :** _____ **Version:** _____

Tester : _____ **Date :** _____ **Assigned To:** _____

Severity : S1 S2 S3 S4 **Priority: 1 2 3 4** **Reproducible : Y/N**

Title: _____

Description: _____

_____

_____

_____

_____

_____

Resolution : Fixed / No-Reproducible Can't Fix / Deferred Won't Fix (Minor)

Date Resolved : _____ **Resolved By :** _____ **Version :** _____

Resolution Comment: _____

_____

_____

Retested By : _____ **Version Tested :** _____ **Dated Tested :** _____

Retest Comment : _____

_____

_____

Signatures :

Originator: _____ **Tested:** _____

Programmer: _____ **Project Manager:** _____

Marketing: _____ **Product Support:** _____

Fig. 4.3: Sample Defect Template

- Every defect has distinctive attributes in comparison to a test case. It defines the defect in detail and also helps in identification/fixing and categorization of defect.
- Following are some of the attributes of a defect.
  1. **Defect ID** is the unique identification number for the defect.
  2. **Defect Name** is the name of the defect must explain the defect in brief, its nature and type.
  3. **Project Name** indicates the project for which the defect is found.
  4. **Module/Sub-module Name** for which the defect is found may be mentioned to create a reference. It may not be required, if defect ID already contains this information.
  5. **Phase Introduced** gives information about when the defect has been added in the application being developed.
  6. **Phase Found** is the phase of the project when the defect is found is added here. This is used to find defect leakage or stage contamination or stage containment.
  7. **Defect Type** is the definition of defect type may be declared in test plan.
  8. **Severity** is the definition of severity of the defect is declared in test plan. Severity describes the impact of the defect on the application. Severities may be critical, high, medium or low depending on the impact of a defect. Severity can be defined as, "the degree of impact that a defect has on the development or operation of a component or system."

     Classification of Defect Severity:

     (i) Critical: The defect affects critical functionality or critical data. It does not have a workaround. Example: Unsuccessful installation, complete failure of a feature.

     (ii) Major: The defect affects major functionality or major data. It has a workaround but is not obvious and is difficult. Example: A feature is not functional from one module but the task is doable if 10 complicated indirect steps are followed in another module/s.

     (iii) Minor: The defect affects minor functionality or non-critical data. It has an easy workaround. Example: A minor feature that is not functional in one module but the same task is easily doable from another module.

     (iv) Trivial: The defect does not affect functionality or data. It does not even need a workaround. It does not impact productivity or efficiency. It is merely an inconvenience. Example: Petty layout discrepancies, spelling/grammatical errors.

Severity is also denoted as:

- S1 = Critical
- S2 = Major
- S3 = Minor
- S4 = Trivial

9. **Priority** is defined on the basis of how the project decides a schedule to take the defects for fixing. Defect or Bug Priority indicates the importance or urgency of fixing a defect. Though priority may be initially set by the Software Tester, it is usually finalized by the Project/Product Manager.

   Priority can be categorized into the following levels:

   (i) High (P_1): Must be fixed in any of the upcoming builds but should be included in the release.

   (ii) Medium (P_2): May be fixed after the release / in the next release.

   (iii) Low (P_3): May or may not be fixed at all.

   Priority is also denoted as P_1 for high, P_2 for Medium and P_3 for low.

10. **Summary** of the defect shall tell in short about the defect, how many times it has been reproduced and which browsers, operating system and environmental facts have been used for reproducing it.
11. **Defect Description** is the detailed description of the defect including information about the module in which Defect was found.
12. **Status of a Defect** is a dynamic field and it indicates the present status of the defect. Typically, the status of the defect is 'open', 'resolved', 'closed', 'deferred' or 'reopened'.
13. **Reported By/Reported On** is information about the testers who found the effect and date of finding defects are report in the defect report.
14. **Version** is the version of the application in which defect was found.
15. **Date Raised** is the date when the defect is raised.
16. **Detected By** is the name/ID of the tester who raised the defect.
17. **Fixed By** is the name/ID of the developer who fixed the defect.
18. **Date Closed** is the date when the defect is closed.

- Defect report varies from organization to organization. Here, we take another defect report template.
- To prepare a concise and clear report of the bugs occurred during testing/debugging to convey the performance report of an application, it is very necessary to present the details in a manner that is easily comprehensible.
- Defect template is a very important part of the defect life cycle. Defect template is a systematic document to arrange defect in easy to find manner.

DEFECT TEMPLATE			
Category :		Last Changed :	
Title :			
Status :		Browser :	
Severity :		Found in Version :	
Priority :		Found in Build :	
Module :		Fixed in Version :	
Originator :		Fixed in Build :	
Assigned to :		Duplicate ID :	
Steps To Reproduce :			
Actual Result :			
Expected Result :			
Comment :			

Example: Defect/Bug report for Facebook Login Page.

DEFECT TEMPLATE			
Category :	Defect	Last Changed :	
Title :	Facebook_LoginPage_Cant Login with Valid Credentials		
Status :	New	Browser :	Chrome
Severity :	1-Critical	Found in Version :	1.0v
Priority :	P-1	Found in Build :	B01
Module :	LoginPage	Fixed in Version :	
Originator :	Tester_1	Fixed in Build :	
Assigned to :	Developer_1	Duplicate ID :	
Steps To Reproduce :			
1) Launch the browser and Enter valid url www.facebook.com/login/ 2) Enter Email Address and Password. 3) Click on [Log In] Button.			
Actual Result :			
We cannot enter in Homepage after enter valid credential, instead of message popup "wrong email address".			
Expected Result :			
User should enters Homepage.			
Comment :			

- With the help of defect attributes, defects easily customized and tracked. Here, are some major attribute used in defect reports:

Category :	Defect/Project Bug
	Enhancement
	Query
	Others
Status :	New
	Open
	Fixed
	Fix Verified
	Close
	Reopen
	Rejected
	Duplicate
	Can't Reproduce
	Need more Info
	Deffered
Severity :	1-Critical
	2-Major
	3-Moderate
	4-Minimal
Priority :	P-1
	P-2
	P-3
	P-4
Module :	Login Page
	Home Page
	Settings
	Account
Browser :	Internet Explorer
	Firefox
	Chrome
	All
Found in Version / Fixed in Version :	1.0v
	2.0v
	3.0v
	4.0v
Found in Build / Fixed in Build :	B01
	B02
	B03
	B04
Originator :	Test Engineer
	Test Lead
	Test Manager
	Developer
	Developer Lead
	BA / SA

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which of the following is a variance between expected results and actual results of execution of test case on the software or system?
   (a) defect (b) error
   (c) fault (d) failure
2. Causes of defects include,
   (a) incomplete requirements and missing specifications
   (b) Error in coding
   (c) Incorrect design (d) All of the mentioned
3. Which is a cycle of defects from which it goes through covering the different states in its entire life (start to finish/completion)?
   (a) Defect Life Cycle (b) Bug Life Cycle
   (c) Both (a) and (b) (d) None of the mentioned
4. Which of the following is not a state of a Bug in Bug Life Cycle?
   (a) New (b) open
   (c) deferred (d) critical
5. How severe the bug is affecting the application is called as?
   (a) Severity (b) Priority
   (c) Traceability (d) Fixability
6. Deferred status in bug life cycle means,
   (a) Developer feels that the bug is not genuine.
   (b) Bug is repeated twice or the two bugs mention the same concept of the bug.
   (c) The bug is expected to be fixed in next releases.
   (d) None of the mentioned
7. If the defect is not considered a genuine defect by the developer then it is marked as,
   (a) New (b) Open
   (c) Closed (d) Rejected
8. If the defect has been fixed accurately then the status of the defect gets assigned to,
   (a) Retest (b) Verified
   (c) Reopen (d) Closed
9. Which of the following defects may arise when designs are implemented wrongly?
   (a) Design (b) Testing
   (c) Coding (d) Requirement
10. Which of the following defects bound to software's speed, stability, response time, and resource consumption?
    (a) Performance defects (b) Functional defects
    (c) Usability defects (d) None of the mentioned

11. Which is a document that has concise details about what defects are identified in testing, what action steps make against the defects and what are the expected results?
    (a) Error report (b) Defect report
    (c) Test case report (d) Test strategy report
12. The good qualities of the defect report include,
    (a) simple to understand (b) to be reproducible
    (c) having a standard format (d) All of the mentioned

Answers

1. (a)	2. (d)	3. (c)	4. (d)	5. (a)	6. (c)	7. (d)	8. (b)	9. (c)	10. (a)
11. (b)	12. (d)								

Q. II Fill in the Blanks:

1. When actual result deviates from the expected result while testing a software application or product then it results into a _____.
2. High _____ defects are the errors that must be fixed in an upcoming release.
3. _____ can be decided based on how bad/crucial is the defect for the system.
4. If the tester does not find any issue in the defect, its state is_____.
5. A software _____ arises when the expected result don't match with the actual results.
6. The number of defects (all major ones and minor ones) found in the application is directly proportional to the _____ of the software application.
7. If the defect is not considered a genuine defect by the developer then its state is _____.
8. While designing and building the software programmer can make _____ or error is known as defects.
9. A defect's status is set to _____ if it has no impact on the application's operation.
10. _____ defects arise in a product when one fails to understand what is required by the customer/end user.
11. The goal of the defect _____ cycle is to make the defect repair process more systematic and efficient.
12. Defects _____ provide information about defects in the testing process.

Answers

1. defect	2. priority	3. Severity	4. Verified
5. bug/defect	6. quality	7. Rejected	8. mistakes
9. Not a bug	10. Requirements	11. life	12. reports

Q. III State True or False:

1. A defect is a mistake in an software application that prevents the program's regular flow by mismatching the intended behavior with the actual behavior.
2. A defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

3. When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a defect report.
4. A defect can also be error, flaw, failure, or fault in the software.
5. The life cycle refers to the phases/steps that a defect passes through throughout the course of its existence.
6. When a developer makes a mistake during the design or development of an application, and this problem is discovered by a tester, it is referred to as a defect.
7. Whenever, any new defect is found in application, it is defined as a 'Start' state.
8. Defect template is a systematic document to arrange defect in easy to find manner.
9. Priority in defect is defined on the basis of how the project decides a schedule to take the defects for fixing.

Answers

1. (T)	2. (T)	3. (F)	4. (T)	5. (T)	6. (T)	7. (F)	8. (T)	9. (T)	

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. What is defect?
2. When to define defect in 'New' state.
3. List the types of defect.
4. When to define defect in 'Deferred' state.
5. What is defect report?
6. List out types of the software defects by nature.
7. List out types of the software defects by priority.
8. List out types of the software defects by severity.
9. What are critical defects?
10. What is mean by high priority defect?
11. Define defect report.

(B) Long Answer Questions:

1. What is defect? List it's causes in a software.
2. Explain defect life cycle with help of detailed diagram.
3. Explain different states of defect in defect workflow.
4. Explain the attributes of defect report.
5. Explain functional defect in detail.
6. Explain classification of defects.
7. Explain components of Defect Report.
8. Write short note on coding defect.
9. What are software defects by severity
10. Write short note on testing defect.
11. Explain defect report template with example.

CHAPTER 5

Testing Tools

Objectives...

- To understand Concept of Testing Tools
- To learn different Types of Automation Tools
- To implement Automation Testing using Selenium Tool

5.0 INTRODUCTION

- Testing is an integral part of any successful software project. The type of testing (manual or automated) depends on various factors, including project requirements, budget, timeline, expertise, and suitability.
- In manual testing (as the name suggests), test cases are executed manually (by a human, that is) without any support from tools or scripts. But with automated testing, test cases are executed with the assistance of automation testing tools, scripts, and software.
- Manual testing is the oldest and most rigorous type of software testing. Manual testing is a method used by software developers to run tests manually.
- Manual testing is performed by execution of test cases, compares actual test results and expected results.
- As source code changes, each time manual tests are repeated and prone to errors. It is also difficult to execute manual testing on multiple platforms.
- 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.
- Manual software testing is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations.

Limitations of Manual Testing:

1. **Manual Testing is Slow and Costly:** Because it is very labour-intensive, it takes a long time to complete tests. Increasing headcount increases cost.

2. **Manual Tests do not Scale Well:** As the complexity of the software increases, the complexity of the testing problem grows exponentially. This leads to an increase in the total time devoted to testing as well as the total cost of testing.
3. **Manual Testing is Not Consistent or Repeatable:** Variations in how the tests are performed are inevitable, for various reasons. One tester may approach and perform a certain test differently from another, resulting in different results on the same test, because the tests are not being performed identically.
4. **Lack of Training is a Common Problem:** Although not unique to manual software testing.
5. **Testing is difficult to Manage:** There are more unknowns and greater uncertainty in testing than in code development so, it will be 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.
- Automation testing is the application of tools and technology to test software with the goal of reducing testing efforts, delivering capability faster and more affordably. It helps in building better quality software with less effort.
- We can define test automation as, the automation of test-related activities with little or no human interaction. We could define automation as the technique of performing tasks without human intervention.
- Automation testing is also known as Test Automation. In automation testing the tester writes scripts and uses other software to test the software product.

Benefits of Automated Testing:

1. **Faster Feedback Cycle:** Test automation helps to reduce the feedback cycle and bring faster validation for phases in the development of software product.
2. **Increased Efficiency:** Test automation useful because to detect problems or bugs early on during the development phase, which increases the team's efficiency.
3. **Reliable:** Tests in automation perform precisely the same operations each time they are run, thereby eliminating human error.
4. **Repeatable:** We can test how the software reacts under repeated execution of the same operations.
5. **Programmable:** We can program sophisticated tests that bring out hidden information from the application.
6. **Comprehensive:** We can build a suite of tests that covers every feature in the application.
7. **Reusable:** We can reuse tests on different versions of an application, even if the User Interface (UI) changes.
8. **Better Quality Software:** Because we can run more tests in less time with fewer resources.

9. **Fast:** Automated tools run tests significantly faster than human users.
10. **Cost Reduction:** An organization will save money as fewer resources are spent on testing software product using an automated test environment.
11. **Improved Accuracy:** Automated tests can execute tests with 100% accuracy as they produce the same result every time you run them.
12. **Higher Test Coverage:** Automation allows spending time writing new tests and adding them to automated test suite. This increases the test coverage for a product, so more features are properly tested resulting in a higher quality application.

5.1 HOW TO MAKE USE OF AUTOMATION TOOLS?

- Automation testing is a software testing technique that performs using special automated testing software tools to execute a test case.
- Developing or producing software to test the software is called test automation/ automated testing. Automated testing is automating the manual testing process. It is used to replace or supplement manual testing with a suite of testing tools.
- An automation testing tool is software that lets us to define software testing tasks. Test automation is the process of testing various parts of new software with little to no human involvement.
- Automated testing tools assist software testers to evaluate the quality of the software by automating the mechanical aspects of the software testing task. Automated testing tools vary in their underlying approach, quality, and ease of use.
- Software testing tools are frequently used to ensure consistency, thoroughness, and efficiency in testing software products and to fulfill the requirements of planned testing activities.
- These tools may facilitate unit (module) testing and subsequent integration testing (e.g., drivers and stubs) as well as commercial software testing tools.
- There are two types of tasting tools i.e., static testing tools and dynamic testing tools. Static testing tools seek to support the static testing process whereas dynamic testing tools support the dynamic testing process.
    1. **Static Test Tools** do not involve actual input and output. Rather, they take a symbolic approach to testing, i.e., they do not test the actual execution of the software. These tools include Coverage analyzers, Flow analyzers, Flow tests and so on.
    2. **Dynamic Test Tools** test the software system with “live” data. Dynamic test tools include Mutation analyzers, Emulators, Test driver, Test beds and so on.
- For usages 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:

- The testing tools available today have some following salient features:
  1. Testing tools should be easy to use.
  2. These tools should provide complete code coverage and create test documentation in various formats like .doc, .html, .rtf, etc.
  3. Testing tools should use one or more testing strategy for performing testing on host and target platforms.
  4. These tools should provide a clear and correct report on test case, test case status (PASS/FAIL), etc.
  5. Testing tools should support GUI-based test preparation.
  6. These tools should be able to adopt the underlying hardware.

Test Automation Frameworks:

- A test framework is a set of guidelines which can be followed to create test cases and related processes.
- These guidelines can be about coding practices, storage and retrieval of test data and test results, interaction with external resources and many other things.
- A test framework usually contains internal libraries and reusable code modules which provide a foundation for test automation and can be leveraged to build test automation systems for different types of applications.
- Typically, there are four test automation frameworks that are adopted while automating the applications.
  1. **Data Driven Automation Framework:** In data-driven testing frameworks, 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 and stored in an external resource such as text file, excel spreadsheet, CSV file or database table. These frameworks provide the flexibility of executing the same test script with multiple data sets.
  2. **Keyword Driven Automation Framework:** In keyword driven frameworks, 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 behavior.

     For example, HP QTP and Selenium are widely used for keyword-driven testing.
  3. **Modular Automation Framework:** In modular frameworks, the application can be divided into different modules which can be tested independently. Test scripts can be reused to an extent, thus reducing the test script development time.

Programming knowledge is required to work with these frameworks. Using object-oriented concepts, an abstraction layer can be developed which can help in easier maintenance of test scripts.

4. **Hybrid Automation Framework:** In hybrid frameworks, more than one framework is used to achieve the desired objectives. The combination should be designed in such a way that weaknesses of one framework are compensated with the strengths of other frameworks.

- A test automation framework offers following several benefits:
  1. **Flexibility to Run Selective Test Cases:** A test automation framework generally provides the flexibility to execute a single test case, a suite of test cases, or all test cases.
  2. **Reduced Manual Interactions:** The main purpose of automating any process is to reduce manual labor and, implementing a test automation framework helps reduce manual labor that was involved in the execution of test cases.
  3. **Efficiency:** A test automation framework provides a structure for the test automation tasks like test creation, test execution, organization of tests, test data creation or reporting.
  4. **Easy Maintenance:** As all test cases are created following the same structure, debugging and isolation of bugs becomes easier.
- To performing testing with automated testing tools, the following points should be noted:
  1. Select a tool that allows the implementation of automated testing in a way that conforms to the specified long-term testing strategy
  2. Clear and reasonable expectations should be established in order to know what can and what cannot be accomplished with automated testing in the organization.
  3. There should be a clear understanding of the requirements that should be met in order to achieve successful automated testing. This requires that the technical personnel should use the tools effectively.
  4. The organization should have detailed, reusable test cases which contain exact expected results and a standalone test environment with a restorable database.
  5. The testing tool should be cost-effective. The tool must ensure that test cases developed for manual testing are also useful for automated testing.

5.2 TYPES OF TESTING TOOLS

- A test automation tool is a piece of software that enables people to define software testing tasks that are afterwards run with as little human interaction as possible.
- Tools from a software testing context is a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.

- In today's era most of the enterprises and businesses demand quality software and faster releases to get quicker Return on Investment (RoI).
- The demand for delivering quality software at high speed is the maxim today that essentially requires organizations to adopt Agile and DevOps Continuous Integration (CI), Continuous Development (CD) methodologies to achieve faster releases and quality software.
- Testing tools can be classified based on following several parameters:
  1. The purpose of the tool
  2. The Activities that are supported within the tool
  3. The Type/level of testing it supports
  4. The Kind of licensing (open source, freeware, commercial)
  5. The technology used
- Following table gives types of testing tools:

Sr. No.	Tool Type	Used for	Used by
1.	Test Management Tool	Test Managing, scheduling, defect logging, tracking and analysis.	Testers.
2.	Configuration management tool	For Implementation, execution, tracking changes.	All Team members.
3.	Static Analysis Tools	Static Testing.	Developers.
4.	Test data Preparation Tools	Analysis and Design, Test data generation.	Testers.
5.	Test Execution Tools	Implementation, Execution.	Testers.
6.	Test Comparators	Comparing expected and actual results.	All Team members.
7.	Coverage measurement tools	Provides structural coverage.	Developers.
8.	Performance Testing tools	Monitoring the performance, response time.	Testers.
9.	Project planning and Tracking Tools	For Planning.	Project Managers.
10.	Incident Management Tools	For managing the tests.	Testers.

Popular Open Source Automation Testing Tools:

1. **Katalon Studio:**

- Katalon Studio is a test automation tool that enables us to test Web, apps, Mobile apps and Desktop APIs.
- It is Powerful in enabling cross-functional operations for product development teams at scale.
- Katalon Studio is easy to use, robust to expand, yet contains the necessary components for advanced needs with built-in keywords and project templates.

2. **QA Wolf:**

- It is an open-source end-to-end automated testing tool and one of the fastest ways to create QA tests.
- It is fully hosted, so no downloads or installation needed.
- It is automatic code generation and low learning curve enable your entire team to get involved in test creation from the non-technical members to the senior developers.

3. **Selenium:**

- Selenium is one of the most commonly used open-source test automation tool that is used to automate browsers.
- The tool supports multiple programming languages such as Java, C#, Python, etc. to create selenium test scripts.
- Helps to create very effective test scripts for regression testing, exploratory testing, and quick bug reproduction.

4. **Appium:**

- Appium is an open source test automation framework for mobile apps.
- It is built on client/server architecture.

- Appium automates the applications that are created for iOS and Android.
- It is a well-liked mobile automation testing tool attributable to its easy installation and usage.

5. Robotium:

- Robotium 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.

6. Cucumber:

- Cucumber is an open-source tool based upon the concept of Behavioral Driven Development, allows to do automated acceptance testing by executing examples that optimally describe the behavior of the application.
- It has cross-platform OS support and compatibility with programming languages like Ruby, Java and .NET.

7. WATiR:

- Web Application Testing in Rubyis (WATiR) an extremely lightweight, technology independent open source testing tool for web automation testing.
- It allows us to write simple, adaptable readable and maintainable automated tests.

8. Sikuli:

- Sikuli is an open source testing tool built upon the concept of image recognition and possesses the ability to automate anything that is seen on the screen.
- It is useful to automate non-web-based desktop applications.
- It is also known for its quick bug reproduction.

9. Apache JMeter:

- Apache JMeter is an open source Java desktop app intended mainly for web applications' load testing.
- Apache JMeter is also supports unit testing and limited functional testing.
- It has features like dynamic reporting, portability, powerful Test IDE etc.
- It supports different type of applications, protocols, shell scripts, Java objects, and databases.

10. WatiN:

- Web Application Testing in.NET (WatiN) an open source test automation framework that aids in UI and functional web app testing.
- Mainly intended for Internet Explorer and Firefox browsers.

11. SoapUI:

- SoapUI is a popular open source API Test Automation Framework for SOAP and REST.

- SoapUI supports functional testing, performance testing, data-driven testing and test reporting as well.

12. Capybara:

- Capybara is an open source acceptance test framework, helpful in testing web applications.
- Capybara simulates the behavior of a real user that interacts with the application.
- It is used in conjunction with other testing tools like Cucumber, RSpec, Minitest, etc.

13. Testia Tarantula:

- Testia Tarantula is an open source tool is created by software company "Prove Expertise" in Finland.
- Modern web tool for software test management mainly intended for agile projects.
- Test executions can be quickly planned by using its tagging features and easy drag and drop interface.
- It has smart tags for fix verification and dashboard for managers.

14. Testlink:

- Testlink is an open source web-based test management tool primarily featured for test plans, test cases, user roles, test projects and test specifications.
- It offers cross-platform OS support.
- Well integrated with other bug tracking systems like JIRA, Bugzilla, Redmine.

15. Windmill:

- Windmill is an open source web testing tool created for automating and debugging the web applications.
- It offers cross browser and cross platform support for web app testing.

16. TestNG:

- TestNG is an open source testing framework, supports almost all kinds of testing like unit testing, functional testing, integration testing, data-driven testing, end-to-end testing.

17. Marathon:

- Marathon 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.

18. Httest:

- Httest is used to implement all types of Http-based tests.
- It offers a range of Http based functionalities.

- Httest allows testing of complex scenarios very effectively.

19. Xmind:

- Xmind is an open source and free mind mapping software useful for regression testing.
- Xmind is built on java platform and has cross-OS support
- It is a light-weight app, provides good encapsulation and also produces an artifact that tells about the total time spent on testing.

20. Wiremock:

- Wiremock is an open source testing tool for Http based application programming interfaces
- It acts as a service virtualization tool that mocks the API for providing quick and powerful end to end testing

21. k6:

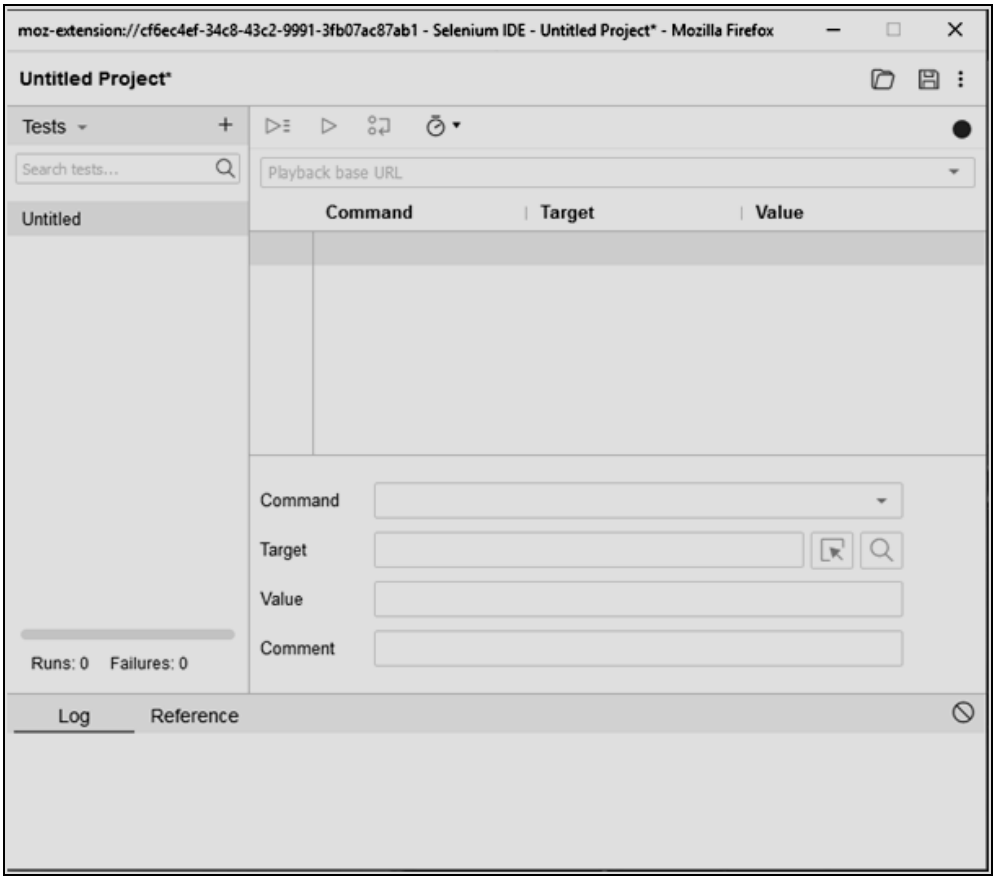
- k6 is an open source load and performance testing tool for testing cloud-native applications, APIs and micro services.
- Modern developer-centric CLI tool with test cases written in ES6 JavaScript and with builtin support for HTTP/1.1, HTTP/2 and WebSocket protocols
- k6 built for automation, and can easily be introduced into automation pipelines in Jenkins, GitLab, Azure DevOps Pipelines, CircleCI and other CI/CD tools for performance regression testing.

5.3 IMPLEMENTATION OF AUTOMATION TESTING USING SELENIUM TOOL

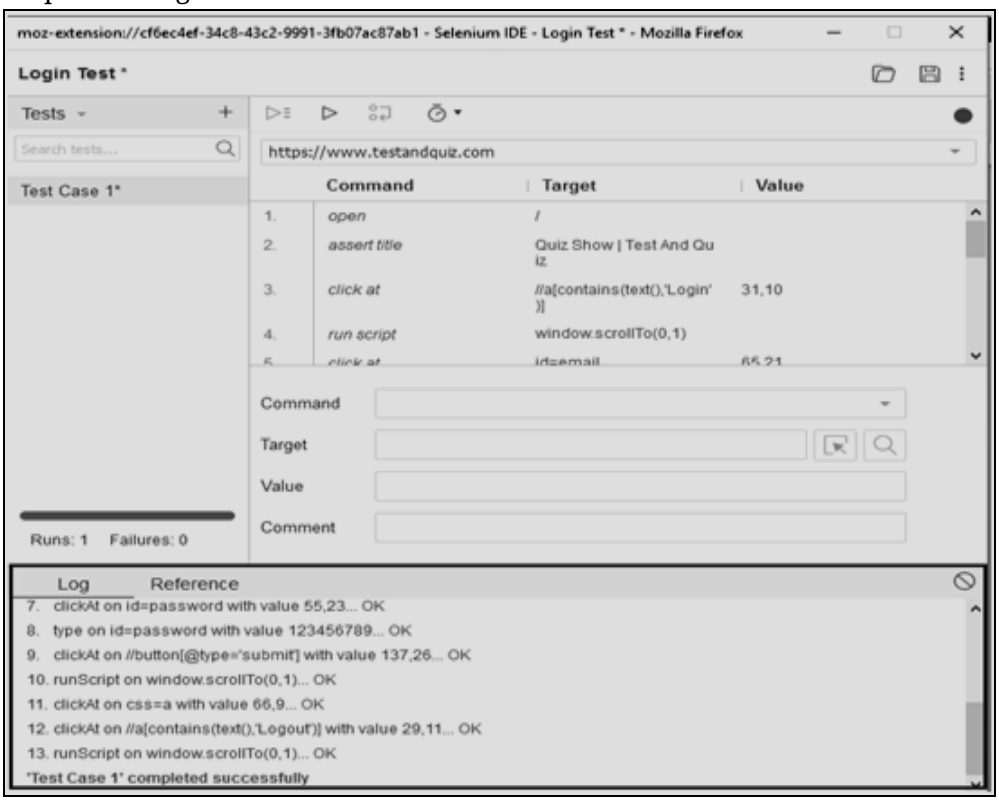
- Selenium is a popular open-source software testing tool. Selenium was created by Jason Huggins in 2004.
- Selenium is a very well-known tool when it comes to testing automation that allows its users to write scripts in a lot of different languages, including Java, C#, Python, Perl, and Ruby.
- Selenium automation testing tool also runs in several operating systems such as Windows, Linux, Solaris, Macintosh and also supports iOS, Windows mobile and Android and browsers like Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Safari and so on.
- Selenium is not just a single tool but a suite of software, each with a different approach to support automation testing.
- Selenium comprises of following components which include Selenium Integrated Development Environment (IDE), WebDriver and Selenium Grid.

Selenium Integrated Development Environment (IDE):

- Selenium IDE is a complete integrated development environment (IDE) framework for Selenium tests. It allows for recording, editing and debugging of tests.
- Selenium IDE is a Chrome and Firefox plugin that can log 'natural' interactions in the browser and generate its code in programming languages like C#, Java, Python, and Ruby, as well as Selenese (Selenium's own scripting language).
- Selenium IDE (Integrated Development Environment) is primarily a record/run tool that a test case developer uses to develop Selenium Test cases.
- Selenium IDE is an easy-to-use tool from the Selenium Test Suite and can even be used by someone new to developing automated test cases for their web applications.
- Selenium IDE allows a user or a test case developer to create the test cases and test suites and edit it later as per their requirements.
- Selenium IDE is a 'record and playback tool' that can help us to create and edit test cases and test suites. Selenium IDE is used to create and execute test cases. It lets us to create test cases by recording our interactions with the browser.



- Example of a Login Test case in Selenium IDE:



1. Selenium WebDriver:

- Selenium WebDriver is a programming interface that can be used to create and execute test cases.
- WebDriver allows us to test across all the major programming languages, browsers and operating systems.
- The test cases are created using elements locators. We locate your elements with one of the eight Selenium element locator techniques and the WebDriver methods will then let us to perform actions on those elements.
- The script we create interacts directly with the browser (which is the reason it's much faster than depreciated Selenium RC).
- There are different drivers for different browsers, which 'interpret' the script we write for it.

2. Selenium Grid:

- Selenium Grid is also an important component of Selenium Suite which allows us to run our tests on different machines against different browsers in parallel.
- Selenium Grid allows us to run multiple tests at the same time on multiple machines also known as parallel testing.

Selenium installation Steps:

1. Download and Install latest version of Java JDK
   - https://www.oracle.com/java/technologies/downloads/
2. Download and Install latest version of Eclipse
   - While installing Eclipse select "Eclipse IDE for Java Developers" option at the beginning and click on install
   - https://www.eclipse.org/downloads/
3. Download Zip of Selenium and Extract it in one folder(.jar files)
   - https://www.selenium.dev/downloads/

Detail steps for using selenium to test the application:

1. Create a new project through File > New > Java Project. Name the project as "newproject".
2. A new pop-up window will open enter details as follow:
   - Project Name.
   - Location to save project.
   - Select an execution JRE.
   - Select layout project option.
   - Click on Finish button.
3. In this step,
   - Right-click on the newly created project and
   - Select New > Package, and name that package as "newpackage".

   A pop-up window will open to name the package,
   - Enter the name of the package.
   - Click on Finish button.
4. Create a new Java class under newpackage by right-clicking on it and then selecting - New > Class, and then name it as "MyClass".

   When we click on Class, a pop-up window will open, enter details as,
   - Name of the class.
   - Click on Finish button.
5. Now set selenium .jar files into Java Build Path. In this step,
   - Right-click on "newproject" and select Properties.
   - On the Properties dialog, click on "Java Build Path".
   - Click on the Libraries tab, and then
   - Click on "Add External JARs."
     - When we click on "Add External JARs" It will open a pop-up window.
     - Select the JAR files you want to add.
     - Select all files inside the lib folder.
     - Select files outside lib folder.
     - Once done, click "Apply and Close" button.
     - After selecting jar files, click on OK button.

6. Finally, click OK and we are done importing Selenium libraries into our project.
   - There are different types of browser drivers according to types of browser like, Firefox, Internet Explorer and Chrome.
   - These driver is needed while testing the Web application. So download the driver for browser of our choice.
   - In example given above we have used geckodriver for FireFox browser. While writing code there is a method as given below:

```
System.setProperty("webdriver.gecko.driver",
                   "C:\\geckodriver-v0.30.0-win64\\geckodriver.exe");
```

   - In this method we have mention the path of downloaded driver for browser as a second argument.
   - Write the test script code and execute the application and observe the output.

CASE STUDIES

Case Study 1: Write a test script to open the Firefox Browser and open "https://www.google.com/ " website and close it using selenium.

Before performing automation testing for the login functionality, there are a couple of basic steps that need to be followed for the test case to be written:

- Create a Selenium WebDriver instance.
- Configure browser if required.
- Navigate to the required web page.
- Locate the relevant web element.
- Perform action on the web element.
- Verify and validate the action.

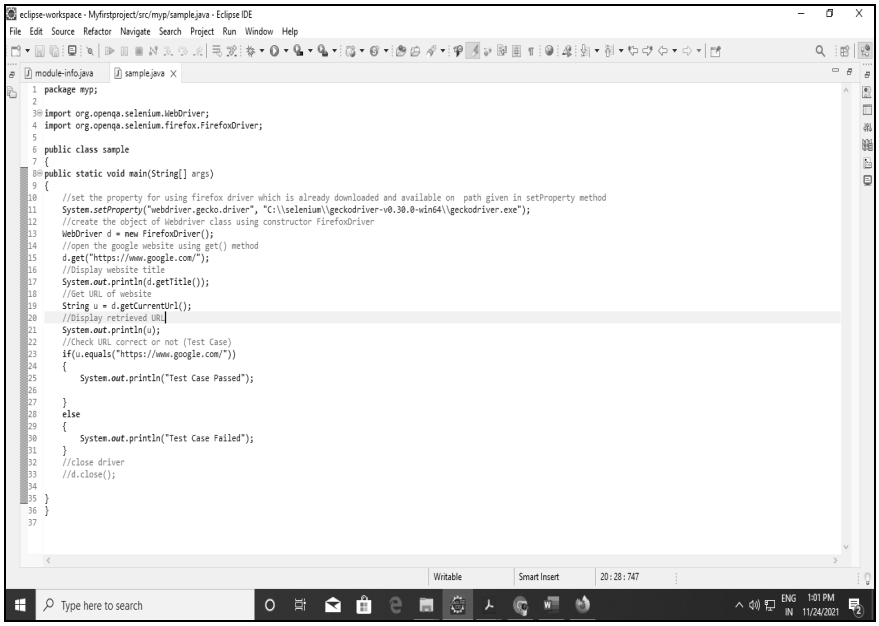
```
Package myp;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class sample
{
   public static void main(String[] args)
   {
      // set the property for using firefox driver which is already
         downloaded and available on  path given in setProperty method
      System.setProperty("webdriver.gecko.driver",
          "C:\\selenium\\geckodriver-v0.30.0-win64\\geckodriver.exe");
      // create the object of Webdriver class using
                                          constructor FirefoxDriver
      WebDriver d = new FirefoxDriver();
      // open the google website using get() method
      d.get("https://www.google.com/");
```

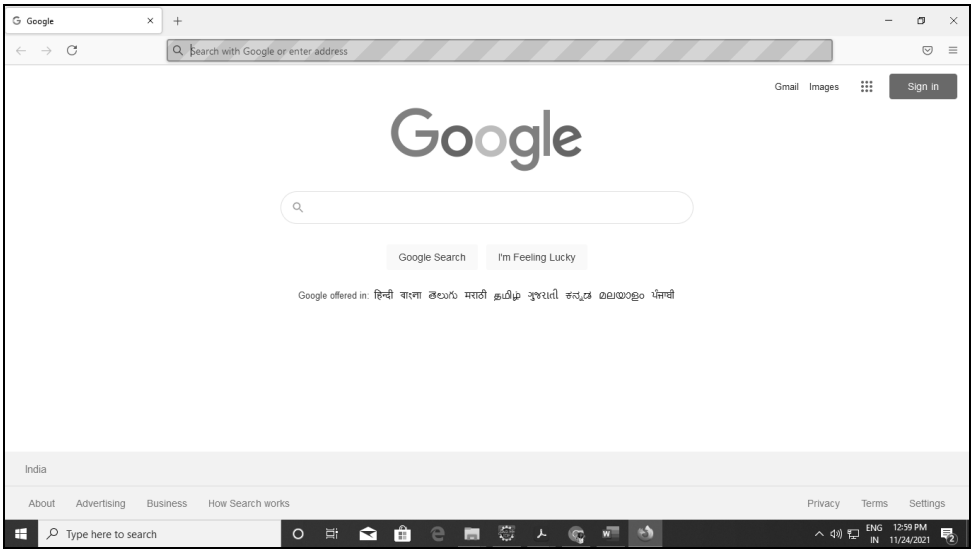
```
            // Display website title
            System.out.println(d.getTitle());
            // Get URL of website
            String u = d.getCurrentUrl();
            // Display retrieved URL
            System.out.println(u);
            // Check URL correct or not (Test Case)
            if(u.equals("https://www.google.com/"))
            {
                  System.out.println("Test Case Passed");
               }
               else
               {
                  System.out.println("Test Case Failed");
               }
               // close driver
               d.close();
         }
      }
```

Output:

```
Google
https://www.google.com/
Test Case Passed
```

Given Test Script Implemented in Selenium:



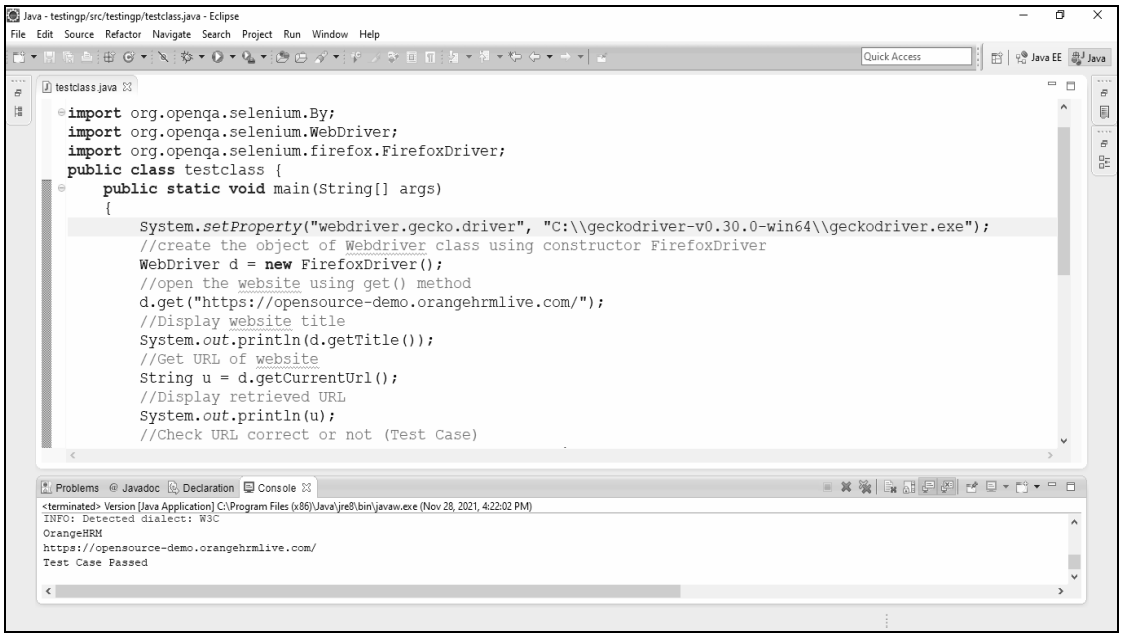


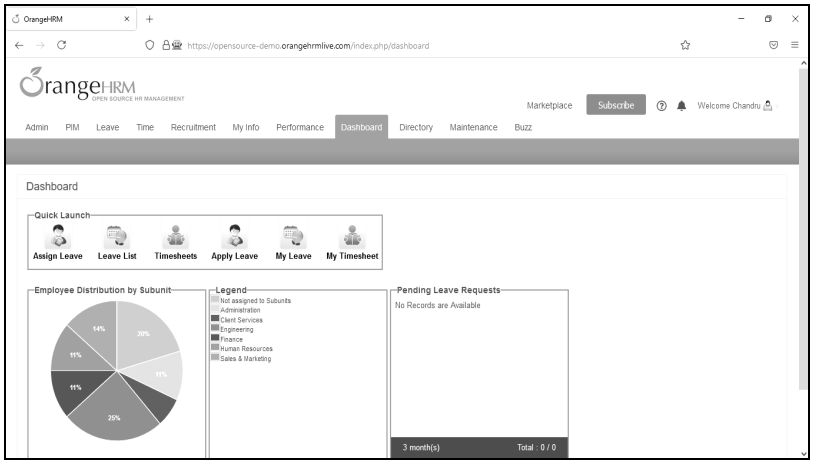
Case Study 2: Write a test script to open the Firefox Browser and open "https://opensource-demo.orangehrmlive.com/" website and enter login credential and test the success or failure of login using selenium.

```
Package testingp;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class testclass
{
   public static void main(String[] args)
   {
      System.setProperty("webdriver.gecko.driver",
             "C:\\geckodriver-v0.30.0-win64\\geckodriver.exe");
      // create the object of Webdriver class using
                                                constructor FirefoxDriver
      WebDriver d = new FirefoxDriver();
      // open the website using get() method
      d.get("https://opensource-demo.orangehrmlive.com/");
      // Display website title
      System.out.println(d.getTitle());
      // Get URL of website
      String u = d.getCurrentUrl();
```

```
        // Display retrieved URL
        System.out.println(u);
        // Check URL correct or not (Test Case)
        if(u.equals("https://opensource-demo.orangehrmlive.com/"))
        {
            System.out.println("Test Case Passed");
        }
        else
        {
            System.out.println("Test Case Failed");
        }
            d.findElement(By.id("txtUsername")).sendKeys("Admin");
            d.findElement(By.id("txtPassword")).sendKeys("admin123");
            d.findElement(By.id("btnLogin")).click();
        // close driver
        // d.close();
    }
}
```

Given Test Script Implemented in Selenium:





PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which testing includes testing a software manually, i.e., without using any automated tool or any script?
   (a) Manual (b) Automation
   (c) Both (a) and (b) (d) None of the mentioned
2. Which testing makes use of specialized tools to control the execution of tests and compares the actual results against the expected result?
   (a) Manual (b) Automation
   (c) Both (a) and (b) (d) None of the mentioned
3. Program testing and fault detection can be aided significantly by testing,
   (a) strategy (b) plan
   (c) tools (d) report

4. Which of the following tool is not an open-source tool?
   (a) Cucumber (b) Selenium
   (c) Bugzilla (d) BugHost
5. Which of the following is/are the uses of software testing tools?
   (i) Test tools are used in reconnaissance.
   (ii) Test tools help in managing the testing process.
   (a) only (i) (b) only (ii)
   (c) Both (i) and (ii) (d) None of the mentioned
6. Which of the following is/are the purposes of using software testing tools?
   (i) To improve the efficiency of test activities by automating repetitive tasks.
   (ii) To automate the activities that requires significant resources when done manually.
   (iii) To automate the activities that cannot be executed manually.
   (a) only (i) and (ii) (b) only (ii) and (iii)
   (c) only (i) and (iii) (d) All (i), (ii) and (iii)
7. Which provide interfaces for executing tests, tracking defects, and managing requirements along with support for quantitative analysis and reporting of the test objects?
   (a) Requirements Management Tools (b) Test Management Tools
   (c) Incident Management Tools (d) Configuration Management Tools
8. Following which testing tools helps with identifying inconsistent or missing requirements.
   (a) Requirements Management Tools (b) Test Management Tools
   (c) Incident Management Tools (d) Configuration Management Tools
9. Which testing tools are used to execute test objects using the automated test scripts?
   (a) Test Data Preparation Tools (b) Monitoring Tools
   (c) Dynamic Analysis Tools (d) Test Execution Tools
10. Which are testing tools necessary for storage and version management of test-ware and related software?
    (a) Requirements Management Tools (b) Test Management Tools
    (c) Incident Management Tools (d) Configuration Management Tools
11. Following which testing tools are helps in planning or risk analysis by providing metrics for the code?
    (a) Review Tools (b) Static Analysis Tools
    (c) Modeling Tools (d) Test Design Tools

12. Which testing tools are used to validate software models by enumerating inconsistencies and finding defects?
    (a) Review Tools (b) Static Analysis Tools
    (c) Modeling Tools (d) Test Design Tools
13. Which testing tools are used to generate test inputs or executable tests?
    (a) Review Tools (b) Static Analysis Tools
    (c) Modeling Tools (d) Test Design Tools
14. Which testing tools manipulate databases, files or data transmissions to set up test data to be used during the execution of tests?
    (a) Test Data Preparation Tools (b) Static Analysis Tools
    (c) Modeling Tools (d) Test Design Tools
15. Which testing tools are used to record tests and usually support scripting languages or GUI-based configuration for parameterization of data and other customization in the tests?
    (a) Test Data Preparation Tools (b) Test Execution Tools
    (c) Dynamic Analysis Tools (d) Test Design Tools
16. Which of the following are the success factors for the deployment of the tool within an organization?
    (i) Assessing whether the benefits will be achieved at a reasonable cost.
    (ii) Adapting and improving processes to fit with the use of the tool.
    (iii) Defining the usage guidelines.
    (a) only (i) and (ii) (b) only (ii) and (iii)
    (c) only (i) and (iii) (d) All (i), (ii) and (iii)
17. Which of the following is/are the main objectives of introducing the selected tool into an organization with a pilot project?
    (i) To learn more detail about the tool.
    (ii) To evaluate how the tool fits with the existing process.
    (iii) To decide the standard ways of using, managing, sorting and maintaining the tool.
    (a) only (i) and (ii) (b) only (ii) and (iii)
    (c) only (i) and (iii) (d) All (i), (ii) and (iii)
18. What criteria should consider while selecting a tool for an organization?
    (i) Evaluating the training needs by considering the current test team's test automation skills.
    (ii) Estimating the cost-benefit ratio based on a concrete business case.
    (iii) Providing training for new users.
    (a) only (i) and (ii) (b) only (ii) and (iii)
    (c) only (i) and (iii) (d) All (i), (ii) and (iii)

Answers

1. (a)	2. (b)	3. (c)	4. (d)	5. (c)	6. (d)	7. (b)	8. (a)	9. (d)	10. (d)
11. (b)	12. (c)	13. (d)	14. (a)	15. (b)	16. (b)	17. (d)	18. (a)		

Q. II Fill in the Blanks:

1. _____ testing is performed by carefully executing predefined test cases, comparing the results to the expected behavior and recording the results.
2. _____ testing is the application of tools and technology to testing software with the goal of reducing testing efforts, delivering capability faster and more affordably.
3. _____ testing tools monitors the performance and response time of system.
4. Test data _____ testing tool does analysis and design, test data generation.
5. _____ management testing tool is responsible for implementation, execution, tracking changes.
6. _____ test tools, test the software system with "live" data.
7. Selenium Grid allows you to run _____ tests at the same time on multiple machines.
8. _____ automation helps to reduce the feedback cycle and bring faster validation.
9. _____ testing tools does coverage analyzers, Flow analyzers, Flow tests and so on.
10. Selenium _____ is a programming interface that can be used to create and execute test cases.
11. Software testing _____ enable to design, develop, maintain, manage, execute and analyze automated tests for applications running on different platforms such as desktop, Web, mobile and so on.

Answers

1. Manual	2. Automation	3. Performance	4. Preparation
5. Configuration	6. Dynamic	7. multiple	8. Test
9. Static	10. WebDriver	11. tools	

Q. III State True or False:

1. Manual testing is a software testing process in which test cases are executed manually without using any automated tool.
2. Selenium is not automation testing tool for Web apps.
3. Automation testing is a type of testing in which we take the help of tools (automation) to perform the testing.
4. JMeter is not an open source Java desktop app which is intended mainly for web applications' load testing
5. It is possible to automate everything in the software.

6. Testing tools in software testing are the software products that support various test activities starting from planning, requirement gathering, build creation, test execution, defect logging and test analysis.
7. Automation testing should be performed before starting the manual testing.
8. There is a risk of suspension of the open-source or free tools project.
9. k6 is an open source load and performance testing tool for testing cloud-native applications, APIs and micro-services.

Answers

1. (T)	2. (F)	3. (T)	4. (F)	5. (F)	6. (T)	7. (F)	8. (T)	9. (F)	

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define manual testing?
2. What is an automation testing tool?
3. List the different test automation frameworks.
4. List popular open source automation software testing tools.
5. List 'Web-based' open source automation software testing tools.
6. List out Selenium installation steps in short.
7. List limitations of manual testing. (Any two).
8. What are the characteristics of testing tools?
9. What is the difference between manual testing and automation testing?
10. List open source automation software testing tools for 'Mobile application'.
11. Define test automation.

(B) Long Answer Questions:

1. What is manual testing? What are limitations of manual testing?
2. Explain different types of testing tools. Explain four of them in short.
3. What are the major criteria for tool selection?
4. Define Automation testing tool. How to make use of Automation tools?
5. Describe features of Selenium testing tool.
6. What are the benefits of Automation testing?
7. Explain key features of any three popular open source automation software testing tools.
8. Write the detail steps for using selenium to test the application.
9. Write note on Selenium tool.
10. Write test script to check login credentials.

❖❖❖

Notes

Notes