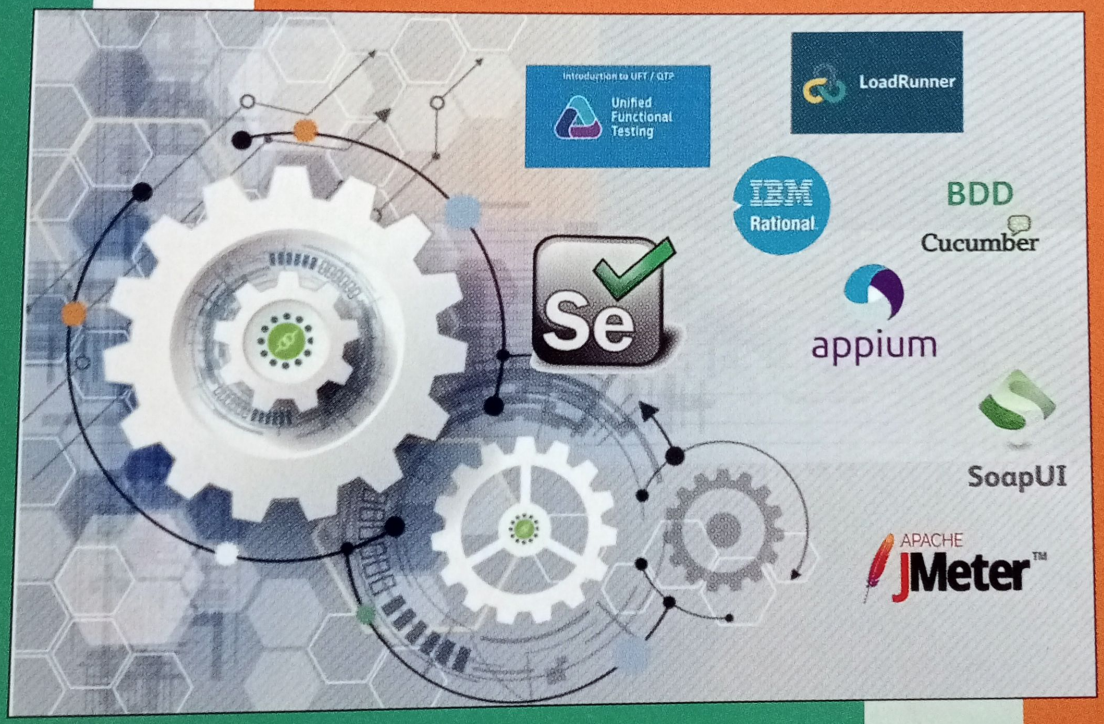
T. Y. B. Sc.
COMPUTER SCIENCE
SEMESTER-VI

NEW SYLLABUS
CBCS PATTERN

SOFTWARE TESTING TOOLS

Ms. SMITA GHORPADE
Ms. KIRTI MORE



NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

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.

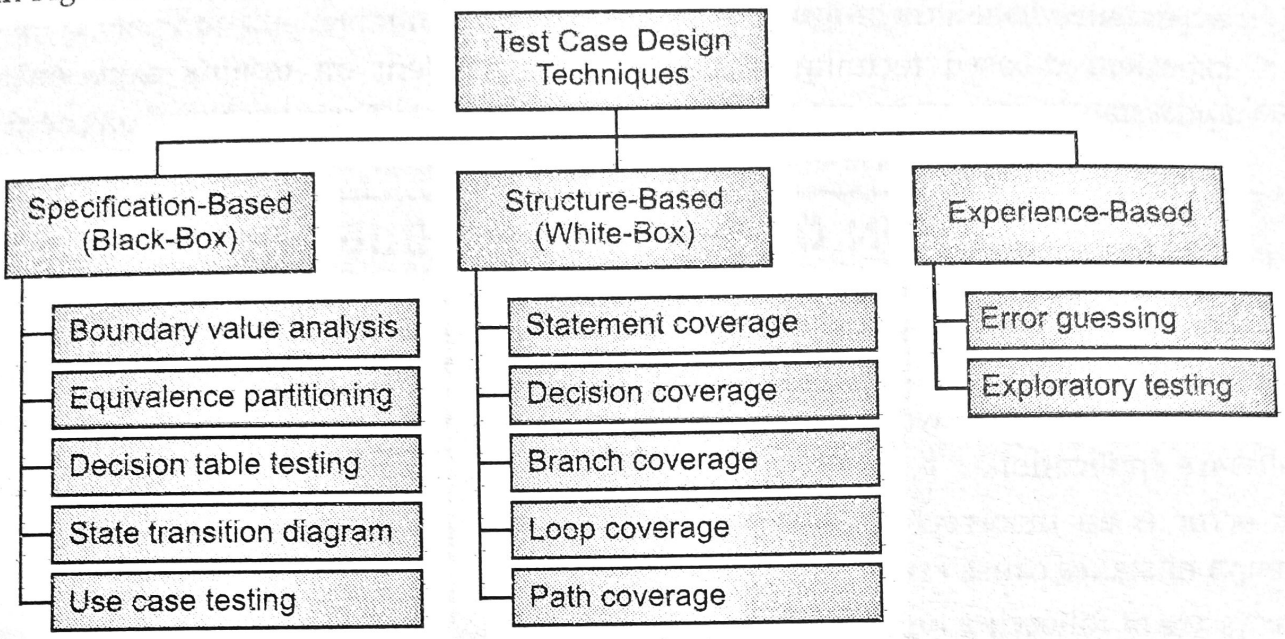


Fig. 1.1: Techniques for Test Case Design

- 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.

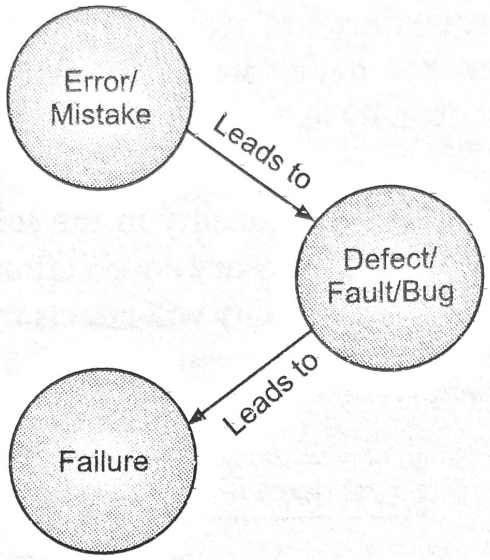


Fig. 1.2: Relation between Error, Bug and Failure

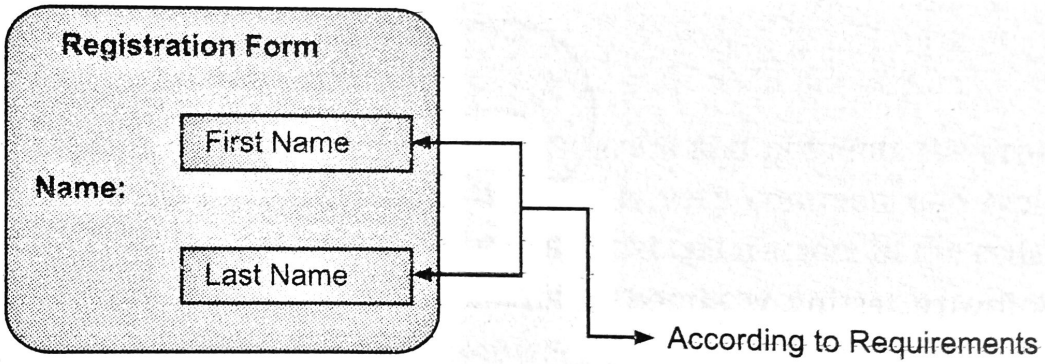
- Software testing is aimed at identifying any bugs, errors, faults or failures (if any) present in the software.
- Bug is defined as, a logical mistake which is caused by a software developer while writing the software code.
- Error is defined as, the measure of deviation of the output given by the 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

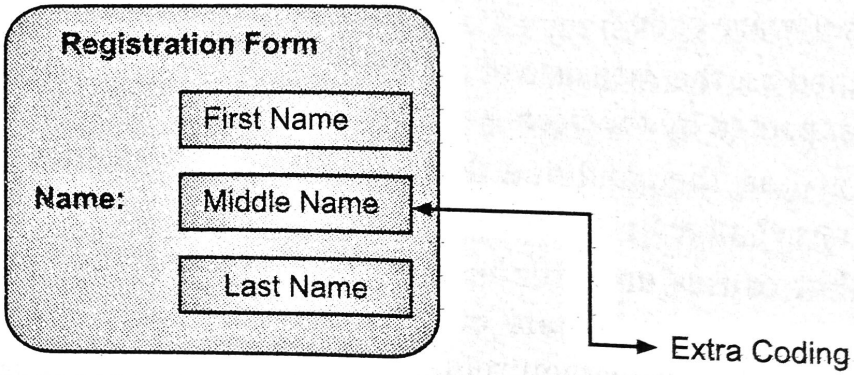
to the "draft" page instead of "inbox", this is happening because of the wrong coding which is done by the developer, hence it is a bug.

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.



(a)



(b)

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.

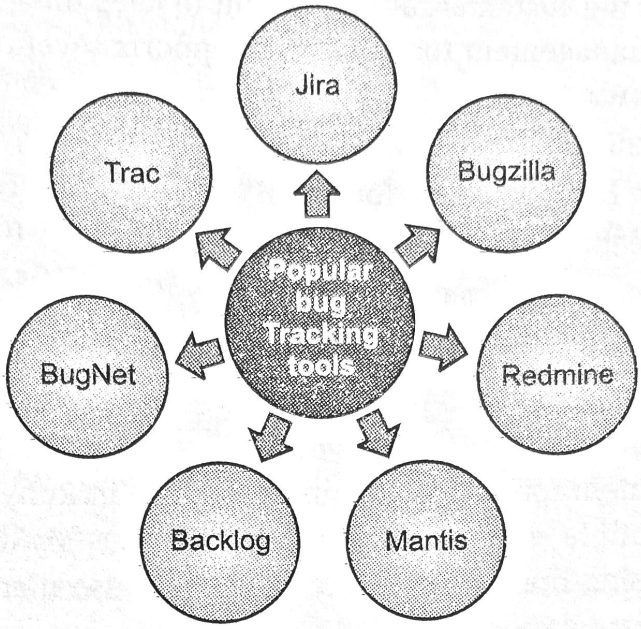


Fig. 1.4: Bug Tracking Tools

- 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.

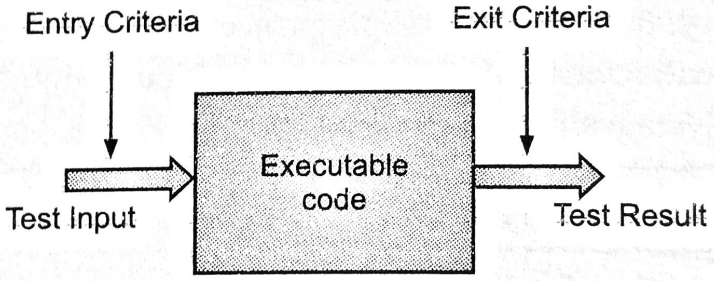


Fig. 1.5: Entry Criteria and Exit Criteria in a Test Case

- 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.

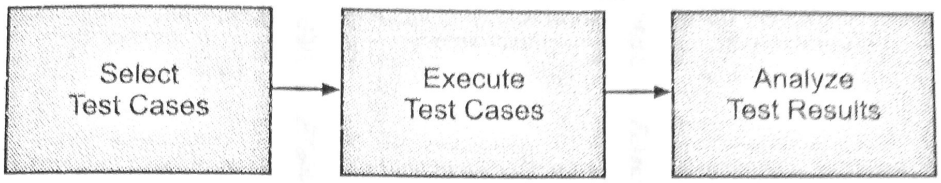


Fig. 1.6: Testing Process using Test Cases

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.

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.