

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

**NEW SYLLABUS
CBCS PATTERN**

SOFTWARE TESTING

**Dr. Ms. MANISHA BHARAMBE
NIKET TAJNE**



SPPU New Syllabus

A Book Of

SOFTWARE TESTING

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

[Course Code CS 362 : Credits – 2]

CBCS Pattern

As Per New Syllabus

Dr. Ms. Manisha Bharambe

M.Sc. (Comp. Sci.), M.Phil. Ph.D. (Comp. Sci.)

Vice Principal, Associate Professor, Department of Computer Science
MES's Abasaheb Garware College
Pune

Niket Tajne

M.C.A.

Assistant Professor, Center for Distance and Online Education
Bharati Vidyapeeth Deemed University
Pune

Price ₹ 190.00



N5947

SOFTWARE TESTING**ISBN 978-93-5451-329-9****First Edition : January 2022****© : Authors**

The text of this publication, or any part thereof, should not be reproduced or transmitted in any form or stored in any computer storage system or device for distribution including photocopy, recording, taping or information retrieval system or reproduced on any disc, tape, perforated media or other information storage device etc., without the written permission of Authors with whom the rights are reserved. Breach of this condition is liable for legal action.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may have crept in. Any mistake, error or discrepancy so noted and shall be brought to our notice shall be taken care of in the next edition. It is notified that neither the publisher nor the authors or seller shall be responsible for any damage or loss of action to any one, of any kind, in any manner, therefrom. The reader must cross check all the facts and contents with original Government notification or publications.

**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**" 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 Software Testing, Software Testing Strategies and Techniques, Levels of Testing, Testing Web Applications and Agile Testing.

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 Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takle, 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 Software Testing **(5 Lectures)**

- Basics of Software Testing – Faults, Errors and Failures
- Testing Objectives
- Principles of Testing
- Testing and Debugging
- Testing Metrics and Measurements
- Verification and Validation
- Testing Life Cycle

2. Software Testing Strategies and Techniques **(10 Lectures)**

- Testability – Characteristics Lead to Testable software
- Test Characteristics
- Test Case Design for Desktop, Mobile, Web Application using Excel
- White Box Testing – Basis Path Testing, Control Structure Testing
- Black Box Testing – Boundary Value Analysis, Equivalence Partitioning
- Differences between BBT and WBT

3. Levels of Testing **(10 Lectures)**

- A Strategic Approach to Software Testing
- Test Strategies for Conventional Software
- Unit Testing
- Integration Testing – Top-Down, Bottom-up Integration
- System Testing – Acceptance, Performance, Regression, Load/Stress Testing, Security Testing, Internationalization Testing
- Alpha, Beta Testing
- Usability and Accessibility Testing
- Configuration, Compatibility Testing

4. Testing Web Applications **(6 Lectures)**

- Dimension of Quality, Error within a WebApp Environment
- Testing Strategy for WebApp
- Test Planning
- The Testing Process – an Overview

5. Agile Testing **(5 Lectures)**

- Agile Testing,
- Difference between Traditional and Agile testing
- Agile Principles and Values
- Agile Testing Quadrants
- Automated Tests



Contents ...

1. Introduction to Software Testing	1.1 – 1.42
2. Software Testing Strategies and Techniques	2.1 – 2.42
3. Levels of Testing	3.1 – 3.46
4. Testing Web Applications	4.1 – 4.26
5. Agile Testing	5.1 – 5.26



Introduction to Software Testing

Objectives...

- To understand Basic Concept of Software Testing
 - To learn Principles and Life Cycle of Testing
 - To understand Testing Metrics and Measurements
 - To study Verification and Validation
-

1.0 INTRODUCTION

- The 21st century is referred to as a knowledge era which is strongly driven by Information Technology (IT).
 - IT brought in revolutionary transformation in the way we collect, assimilate, process and distribute information.
 - Also, specific information is availed to the right users across the world anytime and anywhere in a required form so as to drive every critical transaction or decision.
 - These all could happen because of Information Technology (IT) which is a result of four Cs (Computers, Communication, Connectivity, Content and software).
 - Thus, software is not only a major contributor in IT revolution but also, gifted the world an un-precedent growth.
 - So in this IT world, success of any system or organization depends majorly on software which is part of the system or organization.
 - We need to ensure that software shall work properly. Making software to work properly is not an easy work but is a very challenging and a highly creative work.
 - In this endeavor, we need to address the problem at software development level and bugs level.
 - Software testing plays an important role at both these levels and is a means of ensuring product quality and reduced rework.
 - Also, testing increasing number of organizations and clients understand the value of testing and are now appreciating its contribution.
-

- If software has to work properly, it has to be syntactically and semantically right; meet client and user requirements; fit to the operational environment; and also fulfill requirements like efficiency, reliability, usability, portability and maintainability.
- The purpose of software testing is to analyze a particular item (hardware, software, process) of a software product to determine the differences between its requirements and actual behavior and to evaluate the features and functionality of the given item to ensure it meets the needs of the user.
- In addition, testing evaluates for fulfillment of requirements like efficiency, reliability, usability, portability and maintainability.

What is Testing?

- Testing is the process of **executing software or program or system with the intent of finding/detecting errors.**
- Basically, the purpose of testing is to detect and combat all factors which can lead to software or system failures before delivering the final product to the end users.
- Another purpose of testing is to **assurance of quality of software or system** as per desired outcomes/requirements and preventing of defects and overall software/system **failure.**
- Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.
- In simple words, testing is executing a system in order to identify any errors or missing requirements in contrary to the actual requirements.
- The purpose of testing is to show that a system performs its intended functions correctly.
- Testing is the process of demonstrating that errors (bugs) are not present in the software or system.
- Testing can be defined as, “the process of evaluating a system by manual or automated means to verify that it satisfies specified requirements.”
- The general testing process is the creation of a test strategy (which sometimes includes the creation of test cases), creation of a test plan/design (which usually includes test cases and test procedures) and the execution of tests.
- The purpose of the test process is to provide information to assure the quality of the product, decisions and the processes for a testing assignment.
- Fig. 1.1 shows process of testing.
- Testing is the process consisting of all life cycle activities; static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

- The main intent of software testing is to detect failures of the application so that failures can be discovered and corrected.

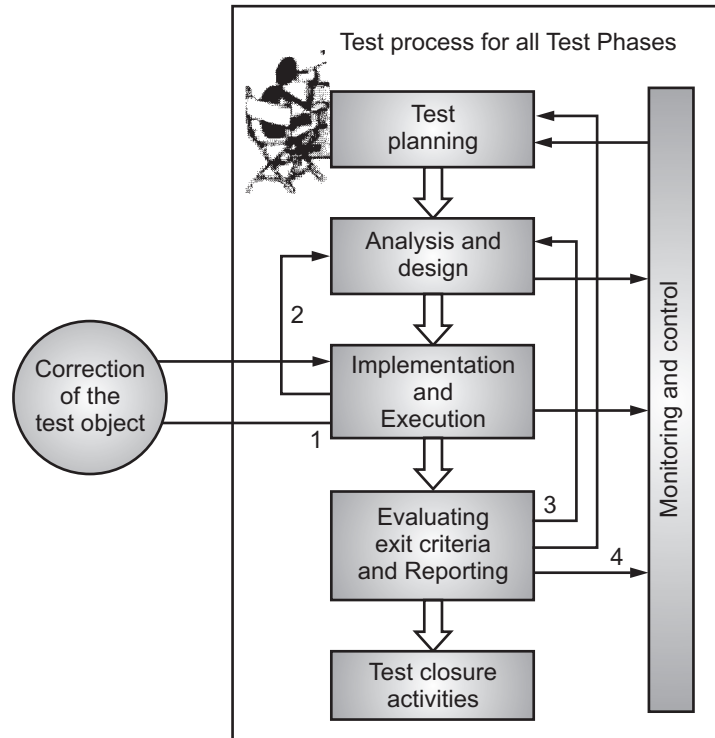


Fig. 1.1: Testing Process

- Software testing is more difficult, complex and time consuming because of the vast array of programming languages, operating systems and hardware platforms that have evolved in the intervening decades.
- Software testing is defined as, “a systematic process of executing a program or system with the intent of finding errors.”

Advantages of Software Testing:

1. Software testing ensures that the software is being developed according to user requirements/expectations.
2. Software testing finds and removes errors, which prevent the software from producing output according to user requirements/expectations.
3. Software testing improves the quality of the software by removing maximum possible errors from it.
4. Software testing removes errors that lead to software failure.
5. Software testing ensures that the software conforms to business as well as user needs.

1.1 BASICS OF SOFTWARE TESTING

- At the highest level, software is a system that operates on a given inputs, processes and transforms these inputs into outputs.
- Testing plays an important role in achieving and assessing the quality of a software product.
- The software product is the complete set of computer programs, procedures and associated documentation and data designated for delivery to a customer (or end user).
- The effectiveness of a computer application in a business environment is determined by how well that application fits in to the environment in which it operates.
- Usually such an environment is composed of Data, People, Structure and Rules and Standards.
- Software testing is a structured process of evaluating measuring and rating the system or its components by manual or automated means against its quality characteristics and sub-characteristics derived from its specified requirements, latent requirements and also standards.
- Thus, software testing is basically a structured process. It involves evaluation of software to know how well it meets specified requirements, standards and fitness to cost and use.
- This makes us to perform software testing efficiently and effectively by binding people, methodology and tools and technology to work with improved capability to produce predictable results.
- Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed/delivered to the end user.
- Software testing is intended to find errors in the software or system. Software testing is an investigation/observation conducted to provide stakeholders with information about the quality of the software product or service under test.
- Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.
- Test techniques include the process of executing a program or application with the intent of detecting failures and verifying that the software product is fit for use.
- Software testing can provide objective, independent information about the quality of software and risk of its failure to users.
- A primary purpose of testing is to detect software failures so that defects may be discovered and corrected.

- Software testing involves the execution of the software or the system to evaluate one or more properties of interest. These properties indicate the extent to which the component or system under test:
 1. Is responds correctly to all kinds of inputs?
 2. Is system sufficiently usable?
 3. Is meets the requirements that guided its design and development?
 4. Is performs its functions within an acceptable time?
 5. Is the system or software achieves the general result its stakeholder's desire.
 6. Is the system or software can be installed and run in its intended environments?
- Types of testing that may be used to test a software includes:
 - Manual testing includes testing a software manually, i.e., without using any automated tool or any script.
 - In manual testing, the tester takes over the role of an end-user and tests the software to **identify any unexpected behavior (or bug/error).**
 - **Automation testing** (or test automation) is a software testing technique in which testing achieved by writing test scripts or using any automation testing tools like LoadRunner, WinRunner, TestComplete, LambdaTest, AvoAssure, Kobiton, Ranorex Studio and so on.
- Some testing roles are given below:
 1. **Test Lead or Test Manager:** The role of Test Lead/Manager is to **effectively and efficiently lead the testing team.** To fulfill this role the Leader must understand the discipline of testing and how to effectively implement a **testing process while fulfilling the traditional leadership roles of a manager.**
 2. **Test Architect:** The role of the Test Architect is to formulate an integrated test architecture that supports the testing process and **create test estimates** and define/build reusable **testing assets for large/complex projects.** To fulfill this role the Test Architect must have a clear understanding of the short-term as well as long-term goals/objectives of the organization, the resources available to the organization and a clear vision on how to most effectively deploy these assets to form integrated test architecture.
 3. **Test Designer or Tester:** The role of the Test Designer/Tester is to **design test cases, execute tests, record test results, document defects and perform test coverage analysis.** To fulfill this role the Test Designer/Tester must be able to apply the most appropriate testing techniques to test the application as efficiently and effectively as possible while meeting the test organizations testing mandate.
 4. **Test Automation Engineer:** The role of the Test Automation Engineer is to create automated **test case scripts** that perform the tests as designed by the Test Designer/Tester. To fulfill this role the Test Automation Engineer must develop and maintain an effective and efficient test automation infrastructure using the tools and techniques available to the testing organization/firm.

5. **Test Methodologist or Methodology Specialist:** The role of the Test Methodologist/Methodology Specialist is to provide the test organization/firm with resources on testing methodologies. To fulfill this role the Test Methodologist/Methodology Specialist works with Quality Assurance (QA) to facilitate continuous quality improvement within the testing methodology and the testing organization/firm as a whole. To this end the Test Methodologist/Methodology Specialist evaluates the test strategy, provides testing frameworks and templates, and ensures effective implementation of the appropriate testing techniques.
6. **Software Tester:** Software tester is a expert who conducts prescribed tests on software programs and applications prior to their implementation to ensure quality, design integrity and proper functionality. The tester's job is to find out defects so that they will be eventually fixed by developers before the software product goes to the user/customer. A software tester must have complete knowledge about testing as a discipline. The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

1.1.1 Goals of Software Testing

- The main goal of software testing is to find errors or bugs as early as possible and to fix bugs and to make sure that the software is bug free.
- The goals of software testing may be classified in to three major categories as shown in Fig. 1.2.

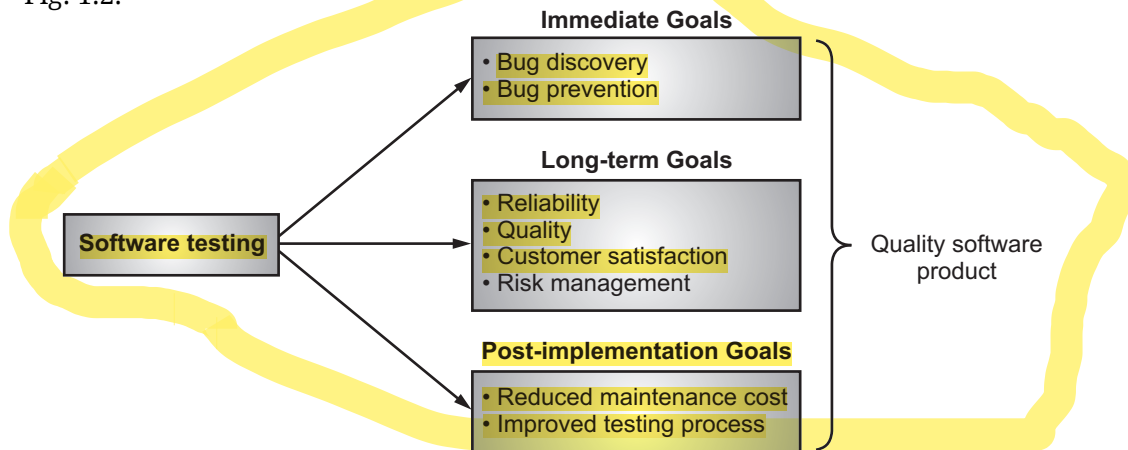


Fig. 1.2: Goals of Software Testing

- Fig. 1.2 shows following goals of software testing:
 1. **Immediate Goals:** These goals are also called as short term goals. These testing goals are the immediate result after testing. These goals contain:
 - (i) **Bug Discovery:** Is the immediate goal of software testing to find errors at any stage of software development. Numbers of the bugs are discovered in early stage of testing.
 - (ii) **Bug Prevention:** Is the consequent action of bug discovery.

2. **Long Term Goals:** These testing goals affect the software product quality in the long term. These include:
 - (i) **Quality:** These goals enhance quality of the software product.
 - (ii) **Customer Satisfaction:** This goal verifies the customers satisfaction for developed software product.
 - (iii) **Reliability:** It is a matter of confidence that the software will not fail. In short reliability means to gain the confidence of the customers by providing them a quality product.
 - (iv) **Risk Management:** Risk must be done to reduce the failure of product and to manage risk in different situations.
3. **Post Implemented Goals:** These goals are important after the software product released. Some of them are listed below:
 - (i) **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify.
 - (ii) **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals.

1.1.2 Model for Software Testing

- The Fig. 1.3 shows model for software testing.

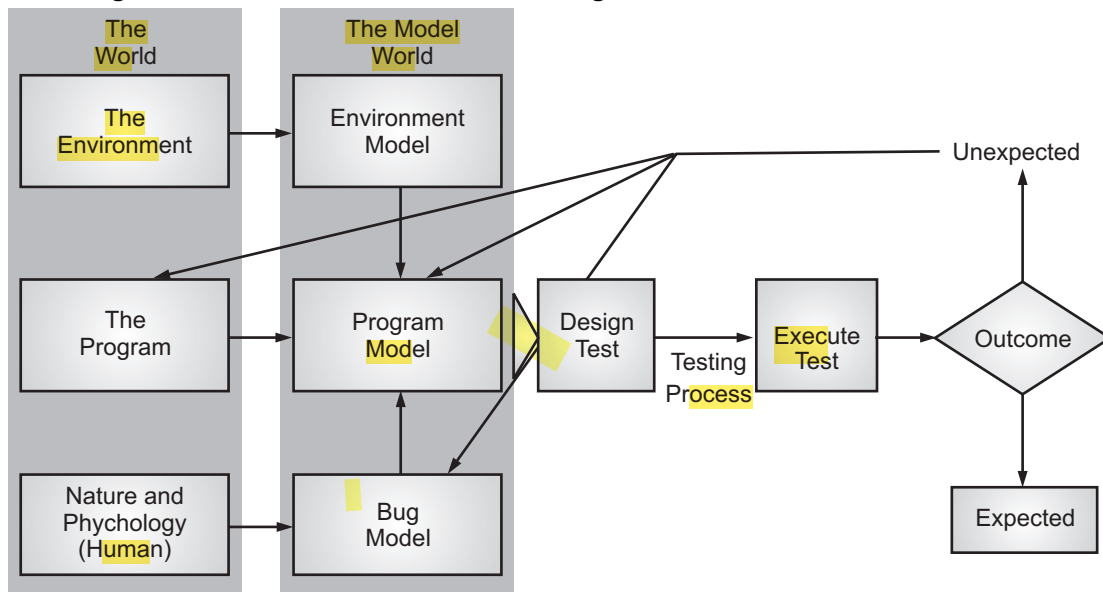


Fig. 1.3: Software Testing Model

- The software is basically a part of a system/environment for which it is being developed or produced. Systems consist of hardware and software to make the software product run.

- Program is a combination of code and data. A program's environment is the hardware and software required to make it run.
- If the program is built using a structured programming concepts with correctness, functionality, modularity and description then it is easy to test. If the code is highly complex then it is difficult to test.
- If we build the model of the program simple with appropriate level of description then testing becomes easy.
- Errors made by human beings in software development produce bugs in software work products like design or code or system. If a code or system with bug is executed, the system may experience a failure.
- Entire bug model is built around nature and psychology of programmers and testers who use bad beliefs or bad assumptions while developing the software.
- Number of the bugs appears because of these bad beliefs or bad assumptions of programmers and testers.
- Any unexpected result reported during testing may force programmers and testers to change their bad beliefs and thereby change the bug model.
- Tests are formal procedures in which inputs must be prepared, outcomes predicted, tests documented, commands executed and results observed; all these steps are subject to error.
- There is nothing magical about testing and test design that immunizes testers against bugs. An unexpected test result is as often cause by a test bug as it is by a rest bug.
- Bugs can creep into the documentation, the inputs and the commands and becloud our observation of results. An unexpected test result, therefore, way lead us to revise the tests.
 - **Unit/Component Testing:** A Unit is the smallest testable piece of software that can be compiled, assembled, linked, loaded etc. A unit is usually the work of one programmer and consists of several hundred or fewer lines of code. Unit testing is the testing we do to show that the unit does not satisfy its functional specification or that its implementation structure does not match the intended design structure.
 - **Integration Testing:** Integration is the process by which components are aggregated to create larger components. Integration testing is testing done to show that even though the components were individually satisfactory (after passing component testing), checks the combination of components are incorrect or inconsistent.
 - **System Testing:** A system is a big component. System testing is aimed at revealing bugs that cannot be attributed to components. It includes testing for performance, security, accountability, configuration sensitivity, startup and recovery.

1.1.3 Faults, Errors and Failures

- Errors are a part of our daily life. Humans make errors in their thoughts, in their actions and in the products that might result from their actions.
- Errors occur almost everywhere. To determine whether there are any errors in our thought, actions and the products generated we resort to the process of testing.
- The primary goal of testing is to determine if the thoughts, actions, and products are as desired i.e., they conform to the requirements.
- Testing of thoughts is usually designed to determine if a concept or method has been understood satisfactorily.
- Fig. 1.4 shows one class of meanings for the terms error, fault and failure.
- A programmer writes a program. An error occurs in the process of writing a program. A fault is the manifestation of one or more errors.
- A failure occurs when a faulty piece of program code is executed leading to an incorrect state that propagates to the program's output. The computer programmer might misinterpret the requirements and consequently write incorrect program code.
- Upon execution, the program might display behavior that does not match with the expected behavior/requirement implying thereby that a failure has occurred.
- A fault in the program code is also commonly referred to as a bug or a defect. The terms error and bug are by far the most common ways of referring to something "wrong" in the program code that might lead to a failure.
- Errors, faults and failures present in the software or system. Bug is defined as, "a logical mistake which is caused by a software developer while writing the software code".
- Error is defined as, "the difference/variance between the outputs produced by the software and the output desired by the user (expected output)".
- Fault is defined as, "the condition that leads to malfunctioning of the software". Malfunctioning of software is caused due to several reasons, such as change in the design, architecture or software program code.
- Defect that causes error in operation or negative impact is called as failure. Failure is defined as, "the state in which software is unable to perform a function according to user requirements/expectations".
- Bugs, errors, faults and failures prevent software from performing efficiently and hence, cause the software to produce unexpected results/outputs.

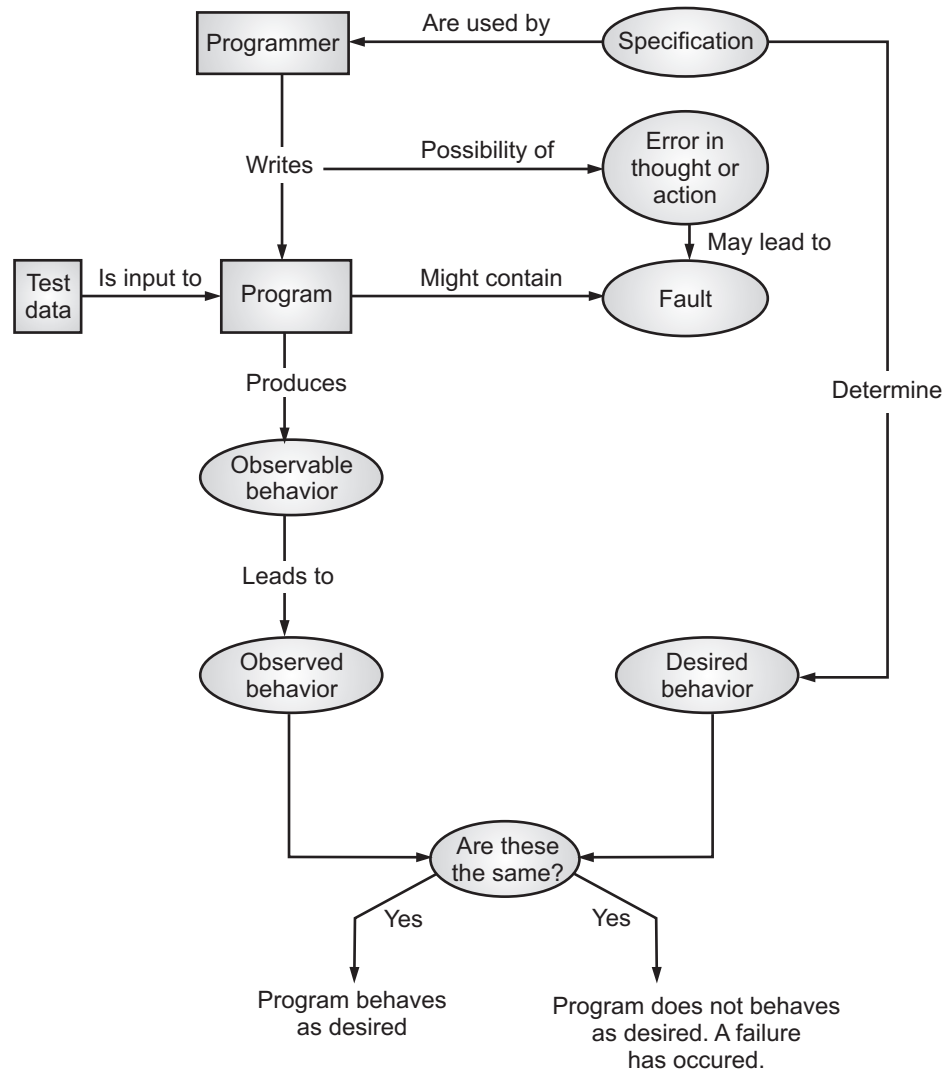


Fig. 1.4: Errors, Faults and Failures in the Process of Testing

- In short,
 - Defect or bug leads to deviation from the expected results Bug leads to faults. Faults can be viewed as incorrect results/steps occurred during the system/program execution.
 - Faults further lead to failures. Failures are the lack of ability of a system to deliver its required function or results within specified performance requirements.

1.1.3.1 Faults

- Fault is a condition that causes the software or system to fail in performing its required function. A fault is the result of an error.

- A fault is an incorrect **step or process in a computer program** which causes the program to perform in an unintended or unanticipated manner.
- A fault (or a defect) is a **missing or incorrect statement in a program resulting from an error** is a fault. So, a fault is the representation of an error.
- A computer programmer makes an error (mistake), which results in a **fault** (defect, bug) in the software source code. A single fault in software or system may result in a wide range of failure symptoms.
- Faults are introduced when the program code is being developed by computer software programmers/developers.
- The computer software programmers/developers may introduce the faults during **original design** or when they are **adding new features, making design changes** or repairing faults that have been identified.
- A fault is the basic reason for software malfunction. Fault can be defined as, "the condition that leads to malfunctioning of the change in the design architecture or software code".
- A **fault is also commonly called a bug**. A bug is a fault in a program which causes the program to perform **in** an unintended behavior.
- **Bug can be defined as, "a logical mistake, which is caused by a software developer/programmer while writing the software code"**.
- 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.
- In simple words, a defect is an error in coding or logic that causes a program to fail or to produce incorrect/unexpected results.

Examples of Faults:

1. **Program does not work.**
2. Program **delivers incorrect results.**
3. **Programs are not written as per the requirements.**
4. **System does not function as per the requirements.**
5. **The wrong data is picked up for the execution.**
6. System response is poor in the case of large volume of data or number of users accessing the system.

1.1.3.2 Errors

- Error is a discrepancy between actual value of the output given by the software and the specified correct value of the output for that given input.
- The error refers to the **difference between the actual output of software and the correct output.**
- We can define an error as, **"the measure of deviation of the outputs given by the software from the outputs expected by the user"**.

- The mistake made by programmer is known as an 'Error'. Mistakes are defined as, errors of thoughts in which a person's cognitive activities lead to actions or decisions that are contrary to what was intended.
- Error also refers to human actions those results in software containing a defect or fault.

Examples of Errors:

1. Syntax errors in the code.
 2. Improper logic.
 3. Improper understanding of requirements
 4. Incorrect system designs (even though requirements have understood!).
 5. Wrong assessment of system or data load.
- Errors can be classified into following two categories:
 1. **Syntax Error:** A syntax error is a program statement that violates one or more rules of the language in which it is written.
 2. **Logic Error:** A logic error deals with incorrect data fields, out of range terms and invalid combinations.

1.1.3.3 Failures

- Failure is the inability of a system or software to perform required function according to its specification.
- In other words, a failure is the state or condition of not meeting a desirable or intended objective/goal.
- A failure occurs when a fault executes. The manifested inability of a system or software to perform a required/expected function within specified specification is known as a failure.
- A failure in a system or a software is evidenced by incorrect output, abnormal termination or unmet time and space constraints.
- Failure is defined as, "that stage of software or system under which it is unable to perform functions according to the user requirements".
- A software failure occurs when the behavior of software is different from the required behavior. A failure is produced only when there is a fault in the system.

Examples of Failures:

1. Incorrect programming/design specifications.
2. Software installation fails.
3. Software generates misleading results, thus fails to get accepted.
4. Customer refuses to accept the software and hence it fails to deliver.

Differences between Bug, Fault and Failure:

Sr. No.	Bugs	Faults	Failures
1.	All sorts of syntax errors in programming code.	Non-working of the programs.	Failure during software installation.
2.	Logic misappropriates.	Delivering not accurate.	Generates misleading results and overall failure in the software operation.
3.	Requirements improperly understood.	Programs are not as per the client requirements.	End user refuses to take delivery of the software and hence entire software failure occurs.
4.	System design incorrect.	System does not perform its functions as per the requirement.	End user implementation failure.
5.	Incorrect documentation.	Operating instructions for the software are not mentioned properly in the manual.	User not able to operate the systems correctly.
6.	Not proper data provided as input data.	Picked up wrong data for execution.	Results are incorrect due to software coding bugs.
7.	Poor system security.	System vulnerable to threats from outside world.	System not fully secured and customers not rely on.

1.2 TESTING OBJECTIVES

- Software testing not only ensures that the product functions properly under all conditions but also ensures that the product does not function properly under specific conditions.
- Software testing is the process of exercising software to verify that it satisfies requirements and to detect errors.
- The **objectives of testing** are listed below:
 1. **Finding Error:** Testing is a process of finding an error in a program or a system. A successful test is one that uncovers an undiscovered error.
 2. **Creating Good Test Cases:** A good test case is one that has a high probability of finding undiscovered error/bugs.
 3. **Quality Improvement:** Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort.

4. **Satisfying Customer Requirements:** A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications.
5. **Reliability and Quality:** The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect – it can only show that software defects are present.

1.3 PRINCIPLES OF TESTING

- There are certain principles that are followed during software testing. These principles act as a standard to test software and make testing more effective and efficient.
- The commonly used software testing principles are explained below:
 1. **Principle #1: Testing shows Presence of Defects:** Testing can show that defects are present in the system or software, but cannot provide that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software or system but, even if no defects are found, it is not a proof of correctness.
 2. **Principle #2: Exhaustive Testing is Impossible:** Testing everything such as all combinations of inputs and pre-conditions is not feasible except for trivial cases. Instead of exhaustive testing, we use risks and priorities to focus testing efforts.
 3. **Principle #3: Early Testing:** Testing activities should start as early as possible in the software or system development life cycle and should be focused on defined objectives or goals.
 4. **Principle #4: Defect Clustering:** A small number of modules contain most of the defects discovered during pre-release testing or show the most operational failures.
 5. **Principle #5: Pesticide Paradox:** If the same tests are repeated again and again, eventually the same set of test cases will no longer find any new bugs. To overcome this 'pesticide paradox', the test cases need to be regularly reviewed and revised and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.
 6. **Principle #6: Testing is Context Dependent:** Testing is done differently in different contexts in different environments. For example, safety-critical software is tested differently from an e-commerce Web site.
 7. **Principle #7: Absence-of-Errors Fallacy:** Finding (or detecting) and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.
 8. **Principle #8: Tests should be as per Client Requirements:** In order to solve any sorts of defects/errors in software, proper tests cases should be developed and

- hardcore testing should be performed manually or either with automation testing tools for making the software usable and bug free to meet the client's requirements.
9. **Principle #9: Define the Expected Output:** When programs are executed during testing, they may or may not produce the expected or required outputs due to different types of errors present in the software or system. To avoid this, it is necessary to define the expected output or result before software testing begins. Without knowledge of the expected/required results, testers may fail to detect an erroneous output.
 10. **Principle #10: Inspect Output of each Test Completely:** Software testing should be performed once the software is complete in order to check its performance and functionality. Also, testing should be performed to find the errors that occur in various phases of software development.
 11. **Principle #11: Pareto Principle to Software Testing:** Pareto principle, states that 80% of the errors are uncovered during testing procedures and 20% is likely to be traced. The problem is to isolate all sorts of errors with thorough testing of source code to make the overall source code error free.
 12. **Principle #12: Testing should Begin “in small” and Progress toward Testing “in large”:** The smallest programming units (or modules) should be tested first and then expanded to other parts of the system.

1.4 TESTING AND DEBUGGING

- Testing and Debugging are the most important steps during the development and after the development of any software or software application developed in any computer programming language.
- Testing involves identifying bug/error/defect/failure in the software without correcting it while Debugging involves identifying, isolating, and fixing the problems/bugs/errors.
- Testing is the process of detecting an error. It is a proactive process where we intend to find defects in the system or software.
- Debugging is the process of identifying the cause of the error once the testing/running of the software detects an error/defect.
- Software testing is a process of identifying defects in the software product and is performed to validate the behavior of the software or the application compared to requirements.
- Debugging means identifying, locating and correcting the bugs usually by running the program. The bugs in debugging are usually logical errors.
- Software testing can be defined as "a set of activities conducted with the intent of finding errors in software" or "a process of verifying that a set of programs collectively functions correctly".

- Debugging is defined as, “the process of identifying, locating and correcting the errors/bugs usually by running the program”.
- Debugging is a technique to find out an error(s) and remove them in a program or set of programs; otherwise, they will lead to failure of these programs.

Process of Debugging:

- Debugging is the process of analyzing causes behind the bugs and removing them. Debugging starts with a list of reported bugs with unknown initial conditions.
- In debugging it is not possible to plan and estimate schedule and effort for debugging. Debugging is a problem solving involving deduction detailed design knowledge is of great help in good debugging.
- Debugging is done by the development team and hence is an insider's work. Also, automation of debugging is not in place.
- The goal of debugging is to find the cause of the software error and correct it. Fig. 1.5 shows the process of debugging.
- Debugging attempts to match symptom with cause, thereby, leading to error correction. Debugging will always have following one of two outcomes:
 1. The cause will be found and corrected and removed or
 2. The cause will not be found.

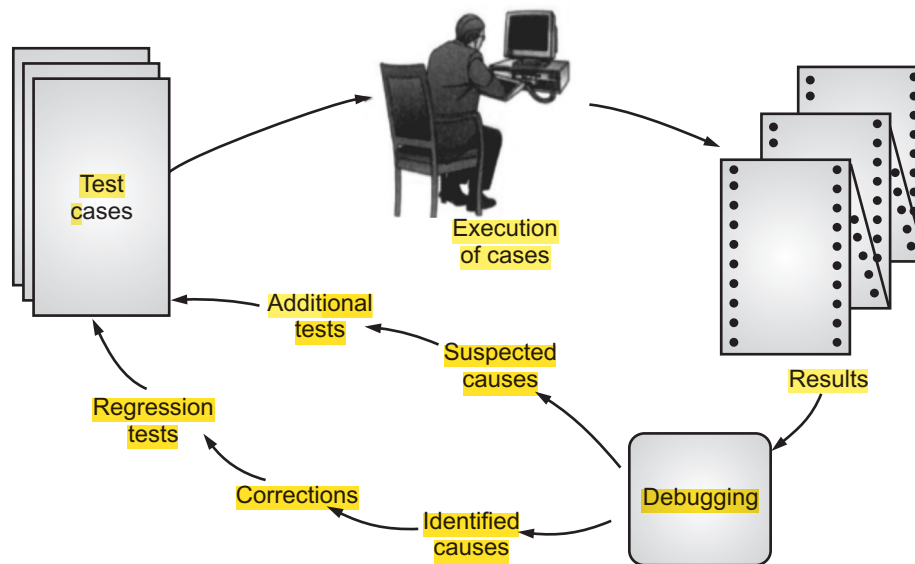


Fig. 1.5: Debugging Process

Stages involved in Debugging:

- Number of peoples thinks that program testing and debugging are the same thing. Though closely related, they are two distinct processes.
- Testing establishes the presence of errors in the program while Debugging is the locating of those errors and correcting them.

- Fig. 1.6 shows various stages in debugging. Debugging depends on the output of testing which tells the software developer about the presence or absence of errors.
- In debugging, the incorrect parts of the program code are located and the program is modified to meet its requirements.
- After repairing, the program is tested again to ensure that the errors have been corrected or removed. Debugging can be viewed as a problem-solving process.

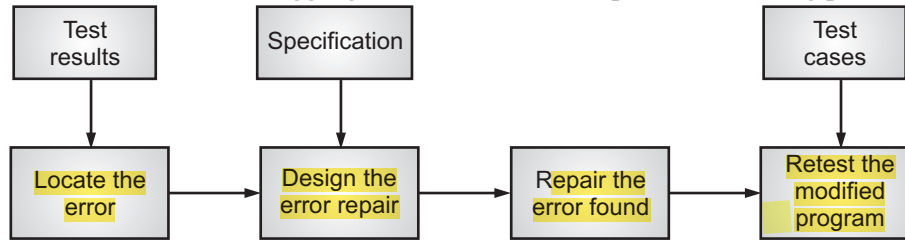


Fig. 1.6: Stages in Debugging

- There is no standard method to teach how to **debug a program**. The debugger must be a skilled person who can easily understand the errors by viewing the output.
- The debugger must have knowledge of common errors, which occur very often in a program.
- After errors have been discovered, then correct the error. If the error is a coding error, then that error can be corrected easily.
- But, if the error is some design mistake, then it may require effort and time. Program listings and the hard copy of the output can be an aid in debugging.

Debugging Strategies:

- Debugging is a systematic process of fixing the number of bugs or defects, in a piece of software so that the software is behaving as expected.
- Various debugging strategies are explained below:

1. Backtracking:

- Backtracking is an effective debugging technique commonly used in small programs.
- In Backtracking debugging technique, the programmer backtracks from the statement which gives the error symptoms for the first time and from this place, all the statements are checked for possible cause of errors.
- Unfortunately, as the number of program source code lines increases, the number of potential backward paths may become unmanageably large.
- Backtracking is done by analyzing the appearance of the error, determining the location of the symptoms and manually following the program's control flow' back to tracking the source program code until the cause of the error is found.

2. Cause Elimination:

- This debugging technique is manifested by induction or deduction and introduces the concept of binary partitioning. In Cause Elimination debugging strategy, the data related to the error occurrence are organized to isolate potential causes.
- In Cause Elimination debugging strategy, a list of all possible causes is developed and tests are conducted to eliminate each.
- If initial tests indicate that a particular cause hypothesis shows promise, the data are refined in an attempt to isolate the bug or error.

3. Program Slicing:

- This debugging strategy is similar to backtracking. However, the search space is reduced by defining slices.
- A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.

4. Brute-force Debugging:

- In this debugging strategy, the programmer appends the print or write statement which, when executed, displays the value of a variable.
- The computer software developer may trace the value printed and locate the statement containing the error. Earlier when the item for execution was quite high, the software developer had to use the core dumps (referred to as the static image of the memory and this may be scanned to identify the bug).
- For debugging we can apply wide variety of debugging tools. Debugging tool is a computer program that is used to test and debug other programs.
- Examples of automated debugging tools include Radare2, WinDbg, Valgrind and so on.

Difference between Testing and Debugging:

Sr. No.	Key Terms	Testing	Debugging
1.	Definition	Testing is a process which intent is identifying the defects.	Debugging is the activity performed by developers to fix the bug found in the system.
2.	Purpose	The purpose of testing is to show that the program or set of programs has a bug or an error. The correction phase is left to the set of developers or designers.	The purpose of debugging is to find out an error or a bug that occurs during the execution of the program and then to correct the bug.

contd. ...

3.	Objective	Main objective of testing is to find bugs and errors in an application which get missed during the unit testing by the developer.	The main objective of debugging is to find the exact root cause at code level to fix the errors and bugs found during the testing.
4.	Perform	As testing is mainly to find out the errors and bugs is mainly performed by the testers. Also if testing is at developer end known as unit testing then it is performed by the developer.	Debugging is to find the missing or de-faulty code in an application hence major performed by the developers only.
5.	Knowledge Required	As testing covers the functional and behavioral flow of an application so only functional knowledge is required for the tester to perform the testing.	Debugging is to find the error at code level so technical and code level knowledge is required for the developer to perform debugging.
6.	Automation	Testing can be manual or made automated with the help of different tools.	Debugging can't be get automated it is always be the manual.
7.	Level	Testing on basis of level of performing is at different level i.e., unit testing, integration testing, system testing, etc.	No such level of debugging.
8.	Programmer's View	Testing proves a programmer's failure.	Debugging is the programmer's vindication (justification).
9.	Process	After getting the errors, testing delivers the errors to the group of programmers to resolve them.	Debugging resolves all the errors which are provided by testing users.

1.5 TESTING METRICS AND MEASUREMENTS

- Metrics and measurement are necessary aspects of managing a software development project.
- For effective monitoring, the management needs to get information about the project such as how far it has progressed, how much development has taken place, how far behind schedule it is, what is the quality of the development? Based on this information, decisions can be made about the project.

- Without proper metrics to quantify the required information, subjective opinion would have to be used, but this is unreliable and goes against the fundamental goals of engineering.
- Software metrics are quantifiable measures that could be measure different characteristics of software systems (and/or the software development process).
- Metric is extremely important to measure the quality, cost and effectiveness of the project and the processes. Without measuring these, a project cannot head towards successful completion.
- Software metrics are a measure of some property of a piece of software systems or its specifications.
- Software testing metric is defined as, “a quantitative measure that helps to estimate the progress and quality of a software testing process”.
- A software testing metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute.
- Software testing metrics can be classified into following categories:
 1. **Product Metrics:** A product metric is a metric used to measure the characteristics of the product such as size, complexity, design features, performance, and quality level. Product metrics in software are used to quantify characteristics of the product being developed, i.e. the software. Product metrics can be further classified as:
 - (i) **Project Metrics:** A set of metrics that indicates how the project is planned and executed. Project Metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.
 - (ii) **Progress Metrics:** A set of metrics that tracks how the different activities of the project are progressing. The activities include both development activities and testing activities. Progress metrics is monitored during testing phase. Progress metrics, is classified into test defect metrics and development defect metrics.
 - (iii) **Productivity Metrics:** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities. These metrics help in planning and estimating of testing activities.
 2. **Process Metrics:** A process metric is a metric used to measure characteristics of the methods, techniques and tools employed in developing, implementing and maintaining the software product or system. A process metric can be used to improve the development and maintenance activities of the software. Process metrics in software are used to quantify characteristics of the environment or the

- process being employed to develop the software. Process metrics aim to measure such considerations as productivity, cost and resource requirements, and the effect of development techniques and tools.
- Measurement plays a significant role in software testing. To assess the quality of the engineered software product or system and to better understand the models that are created, some measures are used.
 - These measures are collected throughout the software development life cycle with an intention to improve the software process on a continuous basis.
 - Measurement helps in estimation, quality control, productivity assessment and project control throughout a software project.
 - **Measurement is the action of measuring something.** Measurement is the assignment of a number to a characteristic of an object or event, which can be compared with other objects or events.
 - Actual measurement must be performed on a given software system in order to use metrics for quantifying characteristics of the given software.
 - **The LoC (Lines of Code) is the most widely used metrics for measurement of size of a program.**
 - Measure can be defined as, “quantitative indication of amount, dimension, capacity, or size of product and process attributes”. Measurement can be defined as, “the process of determining the measure”.
 - **Metrics can be defined as, “quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project and product”.**
 - A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement.
 - **A metric is a measurement of the degree that any attribute belongs to a system, product or process.** For example the number of errors per person hours would be a metric.

1.5.1 Metric

- The term “metric” refers to a standard of measurement. A **metric is a quantitative measure of that system or software.** Software metrics are used to measure the quality of software.
- To achieve an accurate schedule and cost estimate, better quality products, and higher productivity, an effective software management is required, which in turn can be attained through the use of software metrics.
- The **IEEE defines metric** as, “a quantitative measure of the degree to which a system, component or process possesses a given attribute”.

- The goal of software metrics is to identify and control essential parameters that affect software development.
- The **objectives of using software metrics** are given below:
 1. Measuring the size of the software quantitatively.
 2. Assessing the level of complexity involved.
 3. Estimating cost of resources and project schedule.
 4. Assessing the strength of the module by measuring coupling.
 5. Assessing the testing techniques.
 6. Specifying when to stop testing.
 7. Determining the date of release of the software.
- In testing, metric is a unit used for describing or measuring an attribute. Test metrics are the means by which the software product quality can be measured.
- Test metrics provides the visibility into the readiness of the software product and gives clear measurement of the quality and completeness of the software product.
- Metric is a standard unit of measurement that quantifies results and used for evaluating the software processes, products and services is termed as software metrics.
- Fig. 1.7 (a) shows metrics life cycle. The steps/stages in life cycle metrics of metrics are given below:

1. **Analysis:** In this stage, developers identify the required metrics and define them.

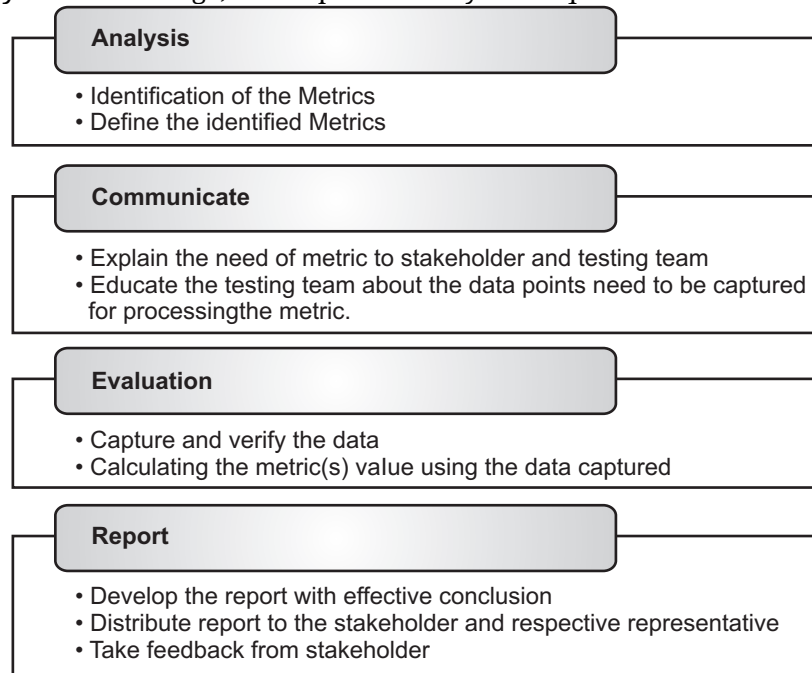


Fig. 1.7 (a)

2. **Communicate:** Once metrics are identified, developers have to explain their importance to stakeholders and the testing team.
 3. **Evaluation:** This stage includes quantifying and verifying the data. Then testers have to use the data to calculate the value of the metric.
 4. **Report:** Once the evaluation process is finished, the development team needs to create a report including a detailed summary of the conclusion. Then the report is distributed among stakeholders and relevant representatives. The stakeholders then give their feedback after reading the information carefully.
- A software metrics is a measurement based technique which is applied to processes, products and services to supply engineering and management information.
 - A software metrics working on the information supplied to improve processes, products and services, if required.
 - A test metric measures some aspect of the test process. Test metrics could be at various levels such as at the level of an organization, a project, a process or a product.

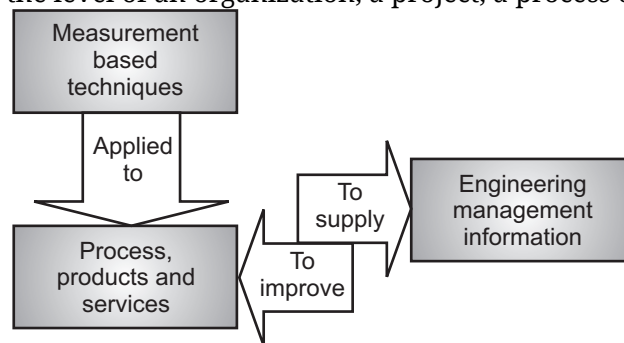


Fig. 1.7 (b): Test Metric Process

- Various characteristics of **software Metrics**:
 1. **Economical:** Computation of metrics should be economical.
 2. **Quantitative:** Metrics must possess quantitative nature. It means metrics can be expressed in values.
 3. **Language Independent:** Metrics should not depend on any programming language.
 4. **Understandable:** Metric computation should be easily understood ,the method of computing metric should be clearly defined.
 5. **Applicability:** Metrics should be applicable in the initial phases of development of the software.
 6. **Repeatable:** The metric values should be same when measured repeatedly and consistent in nature.

Example of Test Metrics:

1. How many defects are existed within the module?
2. How many test cases are executed per person?
3. What is the Test coverage %?

1.5.2 Measurement

- Measurement is the quantitative indication of extent, amount, dimension, capacity or size of some attribute of a software product or software process.
- The **software measurement helps** in estimation, quality control, productivity assessment and project control throughout a software project.
- In software measurement, software is measured to find its efficiency and accuracy. The functionality of software product is totally dependent on the measurement process.
- If the software gives correct output or result in the software measurement process, then it can be sure that software will give meaningful and valuable information within a defined time.
- Software measurements are of two categories namely, direct measures and indirect measures.
 1. **Direct measures** include software processes such as cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported.
 2. **Indirect measures** include products such as functionality, quality, complexity, reliability, maintainability, and many more.

Need of Software Measurement:

- Measurements are the key element for controlling software project processes. Measurements improve the processes by modifying the activities based on different measures.
- Software measurement is needed for the following activities as shown in Fig. 1.8.
 1. **Understanding:** Metrics help in making the aspects of a software process more visible, thereby giving a better understanding of the relationships among the activities and entities they affect.
 2. **Control:** Using baselines, goals/objectives and an understanding of the relationships, we can predict what is likely to happen and correspondingly, make appropriate changes in the process to help meet the goals.
 3. **Improvement:** By taking corrective actions and making appropriate changes, we can improve a software product. Similarly, based on the analysis of a project, a process can also be improved.

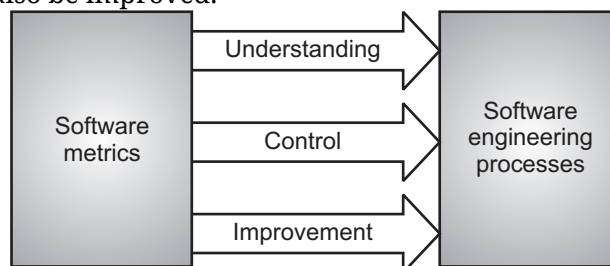


Fig. 1.8

- Metrics are derived from measurements using appropriate formulae or calculations.

1.6 VERIFICATION AND VALIDATION

- The idea of catching defects within each phase, without letting them reach the testing phase, leads to define Verification and Validation.
- **Verification and Validation (V&V) activities are two branches of software testing.** They are complementary to each other and not substitutes of each other. Verification and validation are completely dependent on each other.
- According to Stauffer and Fuji (1986), software V&V is a systems engineering process employing a rigorous methodology for evaluating the correctness and quality of software product through the software life cycle."
 - **Verification refers to the process of ensuring that the software is being developed according to its specifications.** For verification, techniques such as a reviews, analyses, inspections and walkthroughs are commonly used.
 - **Validation refers to the process of checking that the developed software meets the requirements specified by the user.**
- Software verification and validation activities check the software against its specification.
- During requirement gathering phase requirements are collected. The SRS (Software Requirements Specification) document is product of requirement phase. It is verified from the customer to check that proper requirements are captured.
- The design phase takes SRS and implements the design which is useful during coding. The requirement team checks whether the SRS is mapped to design phase properly.
- **Verification means “Are we building the product right?”** and Validation means “Are we building the right product?”
- The V&V is the generic name given to checking processes which ensure that the software conforms to its specification and meets the needs of the customer.
 - **Verification:** It involves checking that the program conforms to its specification.
 - **Validation:** It involves checking that the program as implemented meets the expectations of the customer.
- The system should be verified and validated at each stage of the software development process using documents produced in earlier stages.
- **Verification and validation thus starts with requirements reviews and continues through design and code reviews to product testing.**
- Fig. 1.9 shows software verification and validation.

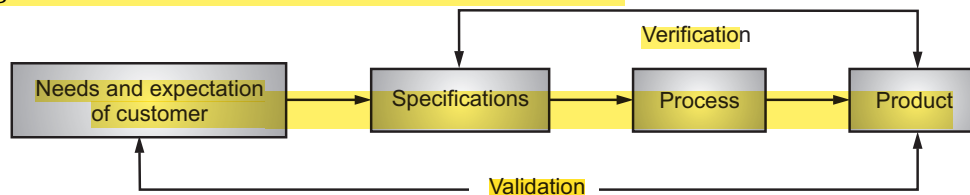


Fig. 1.9: Process of Verification and Validation (V&V)

- Requirements review, design review, code review etc. are examples of verification. Unit testing, Integration testing, system testing etc. are examples of validation.

1.6.1 Verification

- Verification is a disciplined approach to evaluate whether a software product fulfills the requirements or conditions imposed on them by the standards or processes.
- Verification is done to ensure that the processes and procedures defined by the customer and/or organization for development and testing are followed correctly.
- Verification is also called 'static technique' as it does not involve execution of any code, program or work product.
- Verification refers to, "confirmation by examination and provisions of objective evidence that specified requirements/expectations have been fulfilled".
- A test bench or testing workbench is a virtual environment used to verify the correctness of software product.
- A test bench has following components:
 1. **Input:** The entrance criteria or deliverables needed to perform work.
 2. **Procedures to Do:** The task or processes that will transform the input into the output.
 3. **Procedures to Check:** The processes that determine that the output meets the standards.
 4. **Output:** The exit criteria or deliverables produced from the workbench.

Workbench of Verification:

- Verification is the process of checking or verifying the credentials, data or information to confirm their credibility and accuracy.
- A verification workbench is where verification activities are conducted and may be a physical or virtual entity.
- A workbench is a way of documenting how a specific activity has to be performed. Fig. 1.10 shows verification workbench.

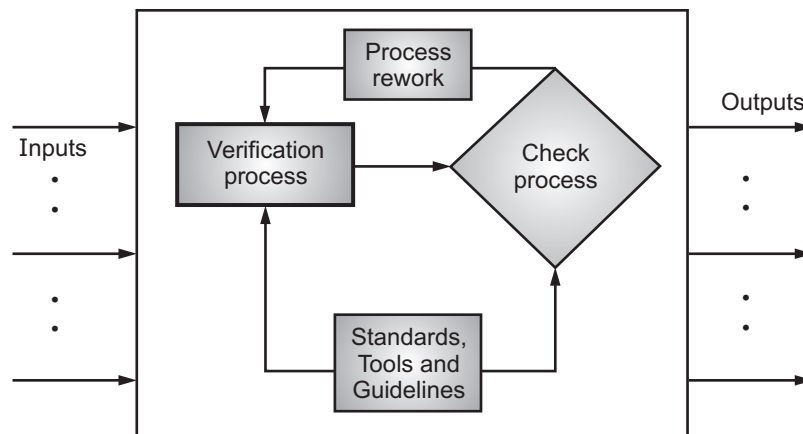


Fig. 1.10: Workbench of Verification

- For every workbench the following basic things are required:
 1. **Inputs:** Must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria for the earlier work bench. For example, if we are verifying a code file, then we must have a written and compatible code as an input to the work bench along with verification plan, verification artifacts, etc.
 2. **Output:** Must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include review comments and the work product with defects if any.
 3. **Verification Process:** Must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while verifying the work product under review.
 4. **Check Process:** Must describe how the verification process has been checked. It is not a verification of the work product but a verification of the process used for verification. Quality plan must define the objectives/goals to be achieved and check processes must verify that the objectives have been really achieved.
 5. **Standards, Tools and Guidelines:** May be termed 'the tools' available for conducting verification activities. There may be coding guidelines or testing standards available for verifications. Sometimes, checklists are used for doing verification.

Methods of Verification:

- There are many methods available of software work products. Some of them are given below:
 1. **Walkthrough:** It is a semi-formal type of a review as it involves larger teams along with the author reviewing a work product. It may involve a project team or part of a project team doing a review jointly as the case may be.
 2. **Self Review:** They may not be referred as an official way of review in most of the software verification descriptions, as it is assumed that everybody does a self check before giving work product for further verification. One must capture the self review records and defects found in self review to improve the process. It is basically a self-learning and retrospection process.
 3. **Peer Review:** It is the most informal type of review where an author and a peer are involved. It is a review done by a peer and review records are maintained. A peer may be a fellow tester as the case may be. There is also a possibility of superior review where peer is a supervisor with better knowledge and experience.
 4. **Audits:** It is a formal review based on samples. They are conducted by auditors who may or may not be experts in the given work product.

5. **Inspection (Formal Review):** Where people external to the team may be involved as inspectors. They are 'subject matter experts' who review the work product. It is also termed 'Fagan's inspection'.

Advantages of Verification:

1. Verification reduces the chances of failures in the software application or product.
2. Verification helps in lowering down the count of the defect in the later stages of software development.
3. Verifying or checking the software product at the starting phase of the development will help in understanding the software product in a better way.
4. It helps in building the software product as per the customer specifications, requirements and needs.
5. Verification can confirm that the work product has followed the processes correctly as defined by an organization or customer.
6. Verification can reduce the cost of finding and fixing defects as each work product is reviewed and corrected faster. Sometimes, defects are fixed as and when they are found.
7. Verification can locate the defect easily as the work product under review is yet to be integrated with other work products. Cost of fixing defect is less as there is no/minimum impact on other parts of software.

Disadvantages of Verification:

1. Actual working software may not be accessed by verification as it does not cover any kind of execution of a work product. 'Fit for use' cannot be assessed in verification.
2. Verification cannot show whether the developed software is correct or not. Rather, it shows whether the processes have been followed or not. If processes are not capable, then the outcome may not be good.

1.6.2 Validation

- Validation is a disciplined approach to evaluate whether the final built software product fulfills its specific intended use.
- It is meant to validate the requirements as defined in requirement specification, ensure that the application as developed matches with the requirements defined and is fit for the intended or future use.
- Validation is also called 'dynamic testing' as the application is executed during validation, with the intention to find defects.
- Software Validation is a process of evaluating software product, so as to ensure that the software meets the pre-defined and specified business requirements as well as the end users/customers' demands and expectations.

- Validation can be defined as, "the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements".

Validation Workbench:

- A validation work bench is a place where validation activities are conducted on the work products and may be a physical or virtual entity.
- Fig. 1.11 shows a validation workbench.

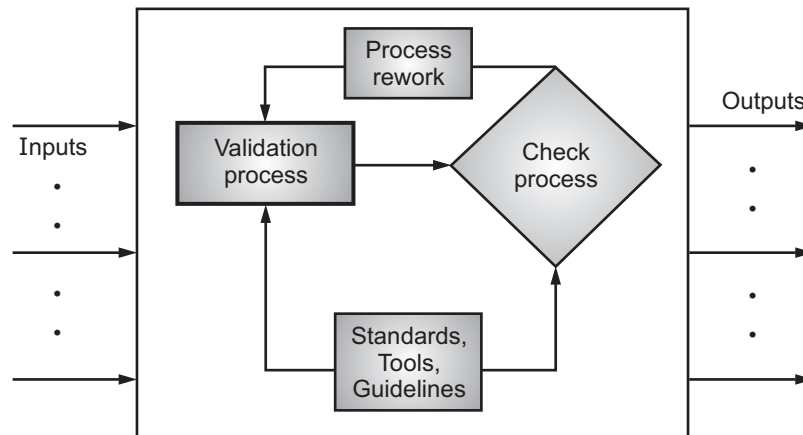


Fig. 1.11: Workbench of Validation

- Fig. 1.11 shows following basic things in workbench in validation:
 - Inputs:** It must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench. For example, if we are validating an application or part of it, then we must have a written and compliable code as an input to the work bench along with test plan and test cases/test data as per definition of input.
 - Outputs:** They are exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include validated work products, defects and test logs.
 - Validation Process:** It must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while validating the work product under testing.
 - Check Process:** It must describe how the validation process has been checked. It is not a validation of the work product but a process. Test plan must define the objectives/goals to be achieved during validation and check processes must verify that the objectives have been really achieved.
 - Standards, Tools and Guidelines:** May be termed "the tools" available for validation. There may be testing guidelines or testing standards available. Sometimes, checklists are used for doing validation.

Levels of Validation:

1. **Unit Testing:** In this testing the smallest piece of software that can be tested in isolation. It is procedure used to validate that individual unit of source code is working properly.
2. **Integration Testing:** It starts at module level when various modules are integrated with each other to form a sub-system or system. Focuses on design and construction of the software architecture.
3. **System Testing:** It is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. The system testing validates that the system meets its functional and non-functional requirements. The system testing is also intended to test up to and beyond the bounds defined in the software/hardware requirements specifications.
4. **Acceptance Testing:** Acceptance testing is the final stage of testing before the system is accepted for operational use. Acceptance testing validates User needs (Functional) and System Performance (Non-Functional).

Advantages of Validation:

1. Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.
2. During verification if some defects cannot be find or missed then during validation process it can be caught as failures.
3. If during verification some specification of the software product is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood.
4. Validation is done during testing such as feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.
5. Validation is generally independent of the platform of development, database or any technical aspect related to software development.

Disadvantages of Validation:

1. Sometimes, it may result into redundant testing as the tester is not aware of internal structure and same part of the code gets executed again and again.
2. No amount of testing can prove that software does not have defects. If defects are not found by conducting the defined test cases, we can only conclude that there is no defect in the set of transactions actually executed. There can be many more defects not captured by these test cases.

Difference between Verification and Validation:

Sr. No.	Verification	Validation
1.	Verification means “are we building the product right?”	Validation means “are we building the right product?”
2.	Verification is a static practice of verifying documents, design, code and program.	Validation is a dynamic mechanism of validating and testing the actual product.
3.	It does not involve executing the code.	It always involves executing the code.
4.	It is human based checking of documents and files.	It is computer based execution of program.
5.	Verification uses methods like inspections, reviews, walkthroughs, and desk-checking etc.	Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
6.	Verification is to check whether the software conforms to specifications.	Validation is to check whether software meets the customer expectations and requirements.
7.	It can catch errors that validation cannot catch. It is low level exercise.	It can catch errors that verification cannot catch. It is high level exercise.
8.	Verification is done by testing team to ensure that the software is as per the specifications in the SRS document.	Validation is carried out with the involvement of testing team.
9.	It generally comes first-done before validation.	It generally follows after verification.

1.7 SOFTWARE TESTING LIFE CYCLE

- Software testing is regarded as careful investigation done by software testing professionals on the source code/every module of the source code to detect errors and to enhance the quality of the software product.
- Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the software product.
- STLC is a group of circularly arranged testing activities, in a specific sequence to understand and test the software in a structured way.
- The fundamental model called the ‘Waterfall model’ is for STLC is discussed in this section. The waterfall model is strictly sequential means it is a series of phases with each phase having a distinct beginning and ending.

- Each phase in Waterfall model enhances development, resulting in production of artifacts and at the end of the life cycle in the desired product/project.
- Fig. 1.12 shows Waterfall testing cycle. The waterfall model is a linear and sequential model defined for software engineering life cycle.

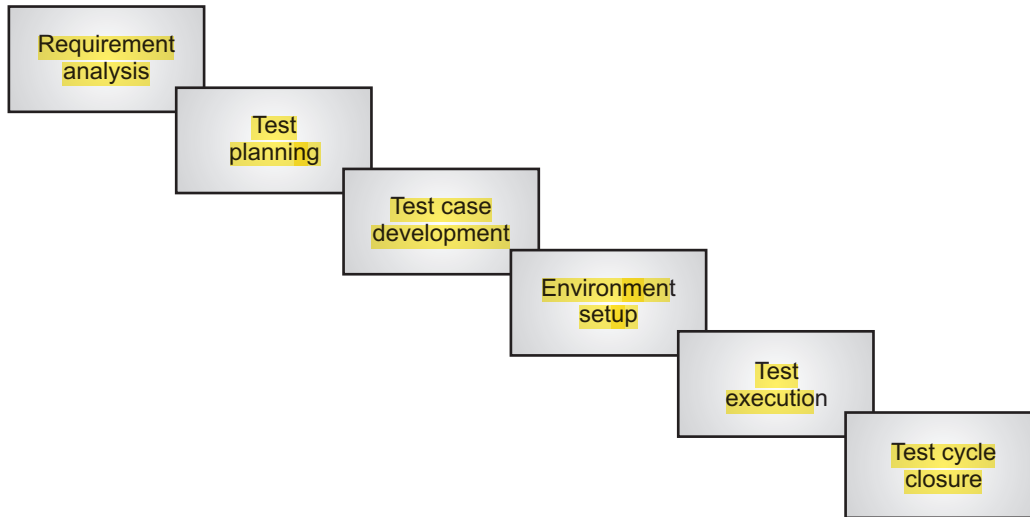


Fig. 1.12: Waterfall Model for Software Testing Life Cycle (STLC)

- Fig. 1.12 shows the phases involved in the STLC. Each of these phases is linked to a specific entry and exit criterion as well as to activities and services.
- The entry criteria contain the required elements that must be executed before starting the test process. The exit criteria define the items that must be completed before the test can be completed.
- Various phases in the Fig. 1.12 are as explained below:

1. Requirements Analysis: Testing should begin in the requirements phase to understand requirements in detail. The requirements can be both functional (what the software program should do) and non-functional (which defines the overall performance, security, availability, etc. of the system). At this stage, the feasibility study is used. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.

Activities in Requirement Phase of STLC:

- Identify test environment details where testing is supposed to be carried out.
- Identify types of tests to be performed.
- Gather details about testing priorities.
- Prepare Requirement Traceability Matrix (RTM). The RTM is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document,

delivered at the conclusion of the software development life cycle. The main purpose of RTM is to validate that all requirements are checked via test cases such that no functionality is unchecked during software testing.

2. Test Planning: In this phase the test strategy or the test plan is defined.

Activities in Test Planning Phase of STLC:

- Preparation of test plan/strategy document for various types of testing.
- Test tool selection.
- Test effort estimation.
- Resource planning and determining roles and responsibilities.
- Training requirement.

3. Test Case Development: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software. The test case development phase involves the creation, verification and rework of test cases and test scripts after the test plan is ready. Initially, the test data is identified then created and reviewed and then reworked based on the preconditions.

Activities in Test Case Development Phase of STLC:

- Create test cases, automation scripts (if applicable).
- Review and baseline test cases and scripts.
- Create test data (if Test Environment is available).

4. Test Environment Setup: This STLC phase decides the software and hardware conditions under which a work product is tested.

Activities in Test Environment Setup Phase of STLC:

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the test environment.
- Setup test environment and test data.
- Perform smoke test on the build.

5. Test Execution: Test execution phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed. Testers execute the software based on the plans and test documents then report any errors found to the development team. The test cases are run according to the given steps and expected results and every test case is marked as pass or fail and test results are created.

Activities in Test Execution Phase in STLC:

- Execute tests as per plan.
- Document test results, and log defects for failed cases.
- Map defects to test cases in RTM.
- Retest the Defect fixes.
- Track the defects to closure.

6. **Test Closure Activity:** Once, the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects. Test cycle closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

Activities in Test Cycle Closure Phase in STLC:

- Evaluate cycle completion criteria based on time, test coverage, cost, software, critical business objectives, and quality.
- Prepare test metrics based on the above parameters.
- Document the learning out of the project.
- Prepare Test closure report.
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Concept of V Model:

- The first variant of the waterfall model is the 'V model' of development (so called because it is in the shape of the alphabet V). The V-model is also known as Verification and Validation (V&V) model.
- The V model (though essentially a waterfall with sequential phases) is different from it, in that testing forms an important part of the model to arrive at a better quality product/project.
- The V-model is SDLC model where execution of processes happens in a sequential manner in V shape. It is also known as Verification and Validation model.
- The V model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.
- The V model is a highly disciplined model and next phase starts only after completion of the previous phase.

- The **V** model consists of a number of phases. The Verification Phases are on the left hand side of the V, the Coding Phase is at the bottom of the V and the Validation Phases are on the right hand side of the V, (See Fig. 1.13).
- Fig. 1.13 shows the different phases in V model of SDLC.

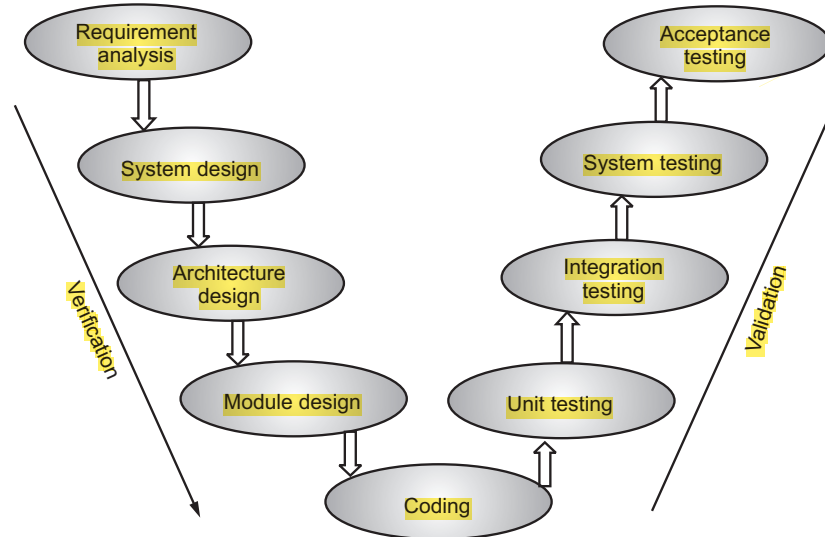


Fig. 1.13: V Model

- Fig. 1.13 shows following phases of V model.
- 1. Verification Phases:** Following are the Verification phases in V-Model:
 - (i) Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement.
 - (ii) System Design:** Once, **you have** the clear and detailed product require-ments, it is time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development.
 - (iii) Architectural Design:** Architectural specifications are understood and designed in this phase. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).
 - (iv) Module Design:** In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design
 - (v) Coding Phase:** The actual coding of the system modules designed in the design phase is taken up in the Coding phase.

2. Validation Phases: Following are the Validation phases in V Model:

- (i) Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase.
- (ii) Integration Testing:** Integration tests are performed to test the coexistence and communication of the internal modules within the system.
- (iii) System Testing:** System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.
- (iv) Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment.

Advantages of V Model:

- 1. It is very easy to understand and apply.
- 2. It works well for smaller projects where requirements are very well understood.
- 3. The simplicity of this model also makes it easier to manage.
- 4. This is a highly disciplined model and Phases are completed one at a time.
- 5. This model is time saving and quick.
- 6. This model avoids the downward flow of the defects.
- 7. In this model testing activities like planning, test designing happens well before coding.

Disadvantages of V Model:

- 1. The model is not flexible to changes.
- 2. This model has high risk and uncertainty.
- 3. This model is poor for long and ongoing projects.
- 4. In this model once, an application is in the testing stage, it is difficult to go back and change functionality.
- 5. V-model is very rigid.
- 6. This model is not a good model for complex and object-oriented projects.
- 7. Software is developed during the implementation phase, so no early prototypes of the software are produced.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which is a process to check whether the software the software meets the user requirements or not?
(a) Testing (b) Strategy
(c) Agile (d) None of the mentioned
2. The objective of testing is,
(a) to debugging bugs (b) to find/detect errors
(c) Both (a) and (b) (d) None of the mentioned
3. Which are independent procedures that are used together for checking that a product, service or system meets requirements and specifications and that it fulfills its intended purpose?
(a) Verification and Testing (b) Testing and Strategy
(c) Verification and Validation (V&V) (d) None of the mentioned
4. Which is a systematic and well defined process of detecting/finding and correcting errors in the software?
(a) Software strategy (b) Software Testing
(c) Both (a) and (b) (d) None of the mentioned
5. Which is the inability of a system or component to perform required function according to its specification?
(a) Fault (b) Error
(c) Bug (d) Failure
6. Which is the step-by-step procedure of fixing the number of errors of developed software?
(a) Testing (b) Strategy
(c) Debugging (d) None of the mentioned
7. Which is a quantitative measure of the software or system.?
(a) metric (b) measurement
(c) Both (a) and (b) (d) None of the mentioned
8. Which metrics describes the project characteristic and execution process such as Number of software developer, cost and schedule etc.?
(a) quality (b) product
(c) process (d) project

9. Test Metrics are used to,
 - (a) Take the decision for the next phase of activities such as, estimate the cost and schedule of future projects.
 - (b) Understand the kind of improvement required to success the project.
 - (c) Used to improve the quality of products and services thus helps in achieving customer satisfaction.
 - (d) All of the mentioned
10. Which is the process to ensure that the software work products meet their specifications?
 - (a) Verification
 - (b) Validation
 - (c) Both (a) and (b)
 - (d) None of the mentioned
11. Software is measured to:
 - (a) Create the quality of the current product or process.
 - (b) Enhance the quality of a product or process.
 - (c) Regulate the state of the project in relation to budget and schedule.
 - (d) All of the mentioned
12. In which model where execution of processes happens in a sequential manner in a V-shape and also known as Verification and Validation (V&V) model.
 - (a) Waterfall model
 - (b) Prototyping
 - (c) V-model
 - (d) None of the mentioned
13. Characteristics of software testing include,
 - (a) Testing must optimize test time and effort to develop an time to execute means testing process means have excellent documentation for the future decisions.
 - (b) Testing results must be accountable with the suitable proofs.
 - (c) Testing must have real time operational scenario.
 - (d) All of the mentioned
14. Validation means,
 - (a) Are we building the product right?
 - (b) Are we building the right product?
 - (c) Both (a) and (b)
 - (d) None of the mentioned
15. Principle of testing include,
 - (a) testing should be as per client requirements
 - (b) pre-plan before testing
 - (c) testing should begin 'in the small' and progress towards 'in the large'
 - (d) None of the mentioned

Answers

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

Q. II Fill in the Blanks:

1. Software testing is the process of executing a software or system with the intent of _____ errors.
2. A mistake in coding is called _____. Errors emerge from the source code itself, caused by inconsistencies or outright fallacies in the internal code structure.
3. _____ is regarded as process of finding and resolving defects/errors within the program that prevents correct operation of the software.
4. _____ is used to improve quality of software according to ser requirements.
5. _____ refers to any shortcoming in a software system that causes it to behave in unexpected and undesirable ways.
6. Software _____ is a process of Verification and Validation (V&V) against the requirements and specifications.
7. The _____ metrics are crucial to measuring the quality and performance of the software.
8. verification means are we building the right product, and Validation means are we building the product right.
9. product metrics define the size, design, performance, quality, and complexity of a product.
10. validation is a process in which the requirements of the customer are actually met by the software functionality.
11. A failure is the inability of a software system or component to perform its required functions within specified performance requirements.
12. The v model is an extension of the waterfall model which is a highly-disciplined model and the next phase starts only after completion of the previous phase.
13. A fault is introduced into the software as the result of an error.
14. Verification and Validation (V&V) is the process of investigating that a software system satisfies spacefication and standards and it fulfills the required purpose.
15. A software testing metrics helps the team to keep a _____ on the software quality at every stage in the software development cycle and also provides information to control and reduce the number of errors.
16. A flaw in a software product that causes the software to fail is called defect/ bug

Answers

1. finding	2. error	3. Debugging	4. Metrics
5. Bug	6. testing	7. test	8. <u>Verification</u>
9. Product	10. Validation	11. failure	12. V-model
13. fault	14. specifications	15. track	16. defect/bug

Q. III State True or False:

1. Software testing is a systematic process which is highly planned with proper test cases and strategy and results are evaluated as per prescribed standards.
2. The term metric refers to a standard of measurement.
3. Validation testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product. **F**
4. A measurement is a manifestation of the size, quantity, amount or dimension of a particular attribute of a product or process.
5. Verification in Software Testing is a process of checking documents, design, code, and program in order to check if the software has been built according to the requirements or not.
6. The V&V is the process of checking that a software system meets specifications and requirements so that it fulfills its intended purpose and may also be referred to as software quality control.
7. Fault is a condition that causes the software to fail to perform its required function.
8. The term error refers to coding or programming mistake that usually shows up due to incorrect syntax or faulty loops.
9. Software measurements have been used in making quantitative/qualitative decisions as well as in risk assessment and reduction in software projects. **F**
10. The V-model (or V&V) model is a type of SDLC model where process executes in a sequential manner in V-shape.
11. Software testing metrics provide quantitative approach to measure the quality and effectiveness of the software development and testing process.
12. Verification is intended to check that a product, service, or system meets a set of design specifications.
13. The goal of software testing metrics is to improve the efficiency and effectiveness in the software testing process and to help make better decisions for further testing process by providing reliable data about the testing process.
14. Software testing metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process.
15. In V&V, the Verification is used to check whether the software conforms to specifications while the Validation is to check whether the software needs the customer expectations.

16. Software testing determines the correctness, completeness and quality of software being developed.

Answers

1. (T)	2. (T)	3. (F)	4. (T)	5. (T)	6. (T)	7. (T)	8. (T)	9. (F)	10. (T)
11. (T)	12. (T)	13. (T)	14. (T)	15. (T)	16. (T)				

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define testing.
2. What is fault?
3. Define error.
4. Define software metric.
5. What is measurement?
6. Give the purpose of software testing.
7. Define verification and validation.
8. Define debugging.
9. Define STLC.
10. Give the purpose of V&V.
11. List any two objectives of software testing.
12. Compare testing and debugging. Any two points.
13. "Debugging starts only after successful conduct of testing". State true or false. Justify.
14. List types of errors.

(B) Long Answer Questions:

1. What is testing? What is its purpose? List steps for testing.
2. What is debugging? Explain its process diagrammatically.
3. What is test metric and measurement? How they can use in software testing?
4. What is Verification and Validation (V&V)? Describe in detail.
5. Differentiate between faults, errors and failures
6. Explain life cycle of software testing with diagram.
7. Differentiate between testing and debugging.
8. With the help of diagram explain V-model? How it can be used in testing? Also states its advantages and disadvantages.

9. Compare verification and validation. Any four points.
10. With the help of diagram describe test metric life cycle.
11. What is debugging strategy? What are the debugging strategies used in testing?
Explain two of them in detail.
12. With the help of diagram describe relation between faults, errors and failures.

■■■

Software Testing Strategies and Techniques

Objectives...

- To understand Software Testing Strategies and Techniques
 - To learn Concept of Testability
 - To study different Black-box Testing and White-box Testing
-

2.0 INTRODUCTION

- Test strategy is an integral part of software development. A test strategy is an outline that describes the testing approach of the software development cycle.
 - A test strategy is a **clear blueprint of the testing approach to be followed in a software development cycle.**
 - A testing strategy for software testing integrates software test case design techniques into a well-planned series of steps that result in the successful construction of the software.
 - A testing strategy defines a template for software testing – a set of steps into which we can place specific test case design techniques and testing methods.
 - Test strategies describe how the product risks of the stakeholders are mitigated at the test-level, which types of testing are to be performed and which entry and exit criteria apply.
 - The strategy provides a road map that describes the steps to be conducted as part of testing, when these steps are planned and then undertake, and how much effort, time, and resources will be required.
 - Therefore, any testing strategy must incorporate test planning, test case design, test execution, and resultant data collection and evaluation.
 - A testing strategy is used to identify the levels of testing which are to be applied along with the methods, techniques, and tools to be used during testing.
-

- Testing strategy also decides test cases, test specifications, test case decisions, and puts them together for execution.
- The purpose of the strategy is to clarify the major tasks and approach toward the test project. It should be very clear, unambiguous, and specific to the objective of the project.
- A number of software testing strategies have been proposed in the literature and all these strategies provides software developer with a template for testing.
- All software testing strategies have the following characteristics:
 1. Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
 2. To perform effective testing, a software development team should conduct effective formal technical reviews. By doing this, number of errors will be eliminated before testing commences.
 3. Testing is conducted by the developer of the software and (for large projects) an independent test group.
 4. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
 5. Different testing techniques are appropriate at different points in time.
- A testing strategy should be developed with the intent to provide the most effective and efficient way of testing the software.
- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented, intermediate level tests designed to uncover errors in the interfaces between modules and high level tests that validate major systems functions against customer requirements.
- A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible.
- A test strategy aims to inform the team leaders and their group of professionals, about the primary testing objectives, the methodologies to be pursued, potential hurdles and the ways to overcome them and finally a definite timeline with the schedules within which the testing process should be finished.
- There are many different types of software test techniques. In this section, we will discuss about the techniques used for testing. Technique refers to the way or method of executing an activity.
- The most common testing techniques are white-box testing and black-box testing.

- Once the software is developed, it should be tested in a proper manner before the system is delivered to the end user. For this, two techniques that provide systematic guidance for designing tests are used.
- These two techniques are explained below:
 1. Once the internal working of the software is known, tests are performed to ensure that all internal operations of the software are performed according to specifications. This is referred to as white-box testing.
 2. Once the specified function of which the software has been designed is known, tests are performed to ensure that each function is working properly. This is referred to as black-box testing.

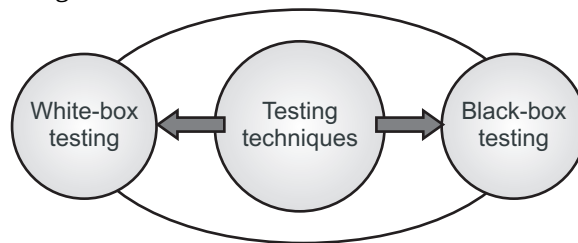


Fig. 2.1: Techniques for Testing

2.1 TESTABILITY

- Testability is the degree to which a program facilitates the establishment of test criteria and execution of tests to determine whether the criteria have been met or not.
- The testability of a program is the degree to which a program facilitates the establishment and measurement of some performance criteria.
- According to IEEE, testability is defined as, “the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.” **OR**
- James Bach defines testability as, “how easily (a computer program) can be tested.”
- There are several characteristics of testability, which are listed below:
 1. **Operability (the better it works, the more efficiently it can be tested):** If a system is designed and implemented with quality in mind, relatively few bugs will block the execution of tests, allowing testing to progress without fits and starts.
 2. **Controllability (the better we can control the software, the more the testing can be automated and optimized):** Software and hardware states and variables can be controlled directly by the test engineer. Tests can be conveniently specified, automated and reproduced.
 3. **Observability (what you see is what you test):** Inputs provided as part of testing produce distinct outputs. System states and variables are visible or queryable during execution. Incorrect output is easily identified. Internal errors are

- automatically detected and reported. Source code is accessible. Testers can easily identify whether the output generated for certain input is accurate or not simply by observing it.
4. **Simplicity (the less there is to test, the more quickly we can test it):** The program should exhibit functional simplicity (e.g. the feature set is the minimum necessary to meet requirements), structural simplicity (e.g. architecture is modularized to limit the propagation of faults) and code simplicity (e.g. a coding standard is adopted for ease of inspection and maintenance).
 5. **Understandability (the more information we have, the smarter we will test):** The architectural design and the dependencies between internal, external and shared components are well understood. Technical documentation is instantly accessible, well organized, specific and detailed and accurate. Changes to the design are communicated to testers. Software that is easy to understand can be tested in an efficient manner.
 6. **Stability (the fewer the changes, the fewer the disruptions to testing):** Changes to the software are infrequent, controlled when they do occur and do not invalidate existing tests. The software recovers well from failures. Software becomes stable when changes made to the software are controlled and when the existing tests can still be performed.
 7. **Decomposability (by controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting):** The software system is built from independent modules that can be tested independently. By breaking software into independent modules, problems can be easily isolated and the modules can be easily tested.

2.2 TEST CHARACTERISTICS

- Various test characteristics are explained below:
 1. **A good test should be neither too simple nor too complex:** Although it is sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors. In general, each and every test should be executed separately.
 2. **A good test has a high probability of finding an error:** The tester must understand the software and attempt to develop a mental picture of how the software might fail. Ideally, the classes of failure are probed. For example, one class of potential failure in a GUI (Graphical User Interface) is a failure to recognize proper mouse position. A set of tests would be designed to exercise the mouse in an attempt to demonstrate an error in mouse position recognition.

3. **A good test should be "best of breed":** In a group of tests have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests. In such cases, the test that has the highest likelihood of uncovering a whole class of errors should be used.
4. **A good test is not redundant:** Testing time and resources are limited. There is no point in conducting a test that has the same purpose as another test. Each and every test should have a different purpose (even if it is subtly different).
- Various characteristics for test system are follows:
 1. A test system must be efficient. Test execution should be quick. This implies again the need for test case order independence, as well as low coupling between tests.
 2. A test system must be portable (at least, if the system under test is or ever will be). It should allow the testers to run tests on all supported platforms.
 3. A test system must be maintainable. It should grow with the system under test, including being able to support new platforms and new features.
 4. A test system must be functional. It should cover the critical quality risks. It should accurately model field usage and failures so your results are meaningful and understandable in context.
 5. The test system should be robust. Minor bugs and small changes in the test environment should not cause major failures and blockages in the tests.
 6. A test system must be reliable. We should be able to repeat your tests. When we run the exact same test against the exact same system under test, we should get the same results.
 7. A test system must be usable by all those who will use it. This means that, in the hands of the testers and other users, the test system feels natural and powerful, with a clear purpose and operation.
 8. The learning curve for the test system should be short, given properly qualified testing team.

2.3 TEST CASE DESIGN

- A test case is a set of conditions under which a tester will determine whether a functionality of software is working correctly or not.
- Test case is a well documented procedure designed to test the functionality of a future in the system. The primary purpose designing a test case is to find errors in the software or system.
- A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective to validate a specific functionality in the software application or system under test.
- IEEE defines test case as, "a set of input values, execution preconditions and expected outcomes developed for a particular objectives".

OR

- A test case is defined as, "test condition such as to exercise a particular program path or to verify compliance with a specific requirement".
- A test case consist of set of input values, execution pre-condition, excepted results and executed post-condition, developed to cover certain test condition.
- Test cases are derived (or written) from test scenario (basically a documentation of a use case).
- A test case is also considered as test suit (a collection of number of test cases to validate the system which is being developed against its original user 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.
- The sample test case template is shown in Fig. 2.2.

Test Case ID:			Test Designed By: <Name>		
Test Priority (Low/Medium/High):			Test Designed Date: <Date>		
Module Name:			Test Executed By: <Name>		
Test Title:			Test Execution Date: <Date>		
Description:					
Pre-conditions:					
Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)

Fig. 2.2

Where,

1. **Test Case ID:** Unique ID for each test case.
2. **Test Priority (Low/Medium/High):** This is useful while test execution. Test priority for business rules and functional test cases can be medium or higher whereas minor user interface cases can be low priority. Test priority should be set by reviewer.
3. **Module Name:** Mention name of main module or sub module.
4. **Test Designed By:** Name of tester.
5. **Test Designed Date:** Date when wrote.
6. **Test Executed By:** Name of tester who executed this test. To be filled after test execution.
7. **Test Execution Date:** Date when test executed.
8. **Test Title/Name:** Test case title. For example, verify login page with valid username and password.
9. **Test Summary/Description:** Describe test objective in brief.
10. **Pre-condition:** Any prerequisite that must be fulfilled before execution of this test case. List all pre-conditions in order to successfully execute this test case.

- 11. Test Steps:** List all test execution steps in detail. Write test steps in the order in which these should be executed. Make sure to provide as much details as you can. Tip – to efficiently manage test case with lesser number of fields use this field to describe test conditions, test data and user roles for running test.
- 12. Test Data:** Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.
- 13. Expected Result:** What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on screen.
- 14. Actual result:** Actual test result should be filled after test execution.
- 15. Status (Pass/Fail):** If actual result is not as per the expected result mark this test as failed. Otherwise update as passed.

- **For Example:** Consider the test case for username and password field of login form:

Test Case ID: 10	Test Designed By: <Name>
Test Priority (Low/Medium/High): Medium	Test Designed Date: <Date>
Module Name: login screen	Test Executed By: <Name>
Test Title: Verify login with valid username and password	Test Execution Date: <Date>
Description: Test the login page	
Pre-conditions: User has valid username and password.	

- Fig. 2.3 shows a test case for above form.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status Pass/Fail)
1.	Navigate to login page	–	–	–	–
2.	Provide valid username	Username=example@gmail.com	Shall accept the username	Accepts the username	Pass
3.	Provide valid password	Password=1234	Shall accept the password	Accepts the password	Pass
4.	Click on Login button	Press Submit Button	User should be able to login	User success full-logged in	Pass

Fig. 2.3

- There are two fundamental approaches to testing software i.e., test-to-pass and test-to-fail.
 1. When we test-to-pass, we really assure only that the software minimally works. We do not push its capabilities. We don't see what we can do to break it. We treat it as the most straightforward test cases. When designing and running your test cases, always run the test-to-pass cases first.
 2. Designing and running test cases with the sole purpose of breaking the software is called testing-to-fail or error-forcing.

2.4**TEST CASE DESIGN FOR DESKTOP, MOBILE, WEB APPLICATION USING EXCEL**

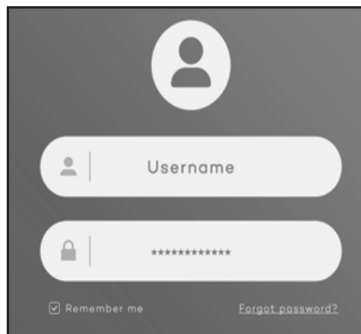
- In this section we will study test case design for the Desktop, Mobile and Web applications.
- A test case is a set of actions executed to verify a particular functionality of the software application. The test case also gives the idea about the error and fault present in the software application.
- The major purpose of test case designing are as follows:
 1. To require consistency in the test case execution.
 2. To make sure a better test coverage.

Test Case Design for Desktop Application:

- The test case design for the desktop applications is by reviewing the following case study for developing the test cases using MS-Excel.
- The all parameters of a test case in Fig. 2.2 are used for defining the test case for desktop application.
- The given tests cased are associated with specific problem of desktop application. Improved the functionality and security the defined MS-Excel workbook shares among with team members of desktop application.

Case Study:

- Prepare the test case for the Desktop application for the login window. Where, we must check the functional requirement of the username and password from the given login window of Desktop application.

**Fig. 2.4****User Requirement Specification:**

- Username must be in the Alphabet format.
- Username not in the digits or special symbols
- Password must be specified in the form of hidden forms.
- Password is not specified in the form of only alphabet
- Password is not specified in the form of only digits
- Password is not specified in the form of only special symbols

- Designing the Test case using MS-Excel for Desktop application is as follows:

A	B	C	D	E	F	G	H	I	J	K	L
Test Case ID	Test Case Name	Prerequisites	Objective	Description	Steps	Input	Expected Result	Actual Result	Status	Bug ID	Remark
1											
2	T101	User_Name_Checking	Login window must be Open	Username must be in the Alphabet format	1 Enter Alphabet in User name 2 Press Ok Button	ABC	Do not show any error Message	Do not showing error Message	Pass	-	-
3	T102	User_Name_Checking	Login window must be Open	Username must be in the Alphabet format	1 Enter Alphabet in User name 2 Press Ok Button	ABC123	Must show the error message	It show the error message	Pass	-	-
4	T103	Password_Checking	Login window must be Open	Password must be specified in the form of hidden characters.	1 Enter Content in Password 2 Press Ok Button	12345	Content represent in the form of hidden character	It show the Password character in hidden character like ****	Pass	-	-
5	T104	Password_Checking	Login window must be Open	Password must be specified in the form of hidden characters.	1 Enter Content in Password 2 Press Ok Button	12345	Content represent in the form of hidden character	It show the Password character not in hidden character like ****	Fail	Bug101	-
6	T105	Password_Checking	Login window must be Open	Password is not specified in the form of only alphabet	1 Enter Content in Password 2 Press Ok Button	ABC	Content represent in the form of Alphabet	Do not showing error Message	Pass	-	-
7	T106	Password_Checking	Login window must be Open	Password is not specified in the form of only alphabet	1 Enter Content in Password 2 Press Ok Button	ABC	Content represent in the form of Alphabet	Do not showing error Message	Pass	-	-
8	T107	Password_Checking	Login window must be Open	Password is not specified in the form of only digits	1 Enter Content in Password 2 Press Ok Button	123	Content not represent in the form of Digits	Do not showing error Message	Pass	-	-
9	T108	Password_Checking	Login window must be Open	Password is not specified in the form of only digits	1 Enter Content in Password 2 Press Ok Button	123	Content does not represent in the form of Digits	It Showing error Message	Pass	-	-
10	T109	Password_Checking	Login window must be Open	Password is not specified in the form of only special symbols	1 Enter Content in Password 2 Press Ok Button	\$%^	Content does not represent in the form of Special Symbols	It Showing error Message	Pass	-	-

Test Case Design for Mobile Application:

- The test case design for the Mobile applications is by reviewing the following case study for developing the test cases using MS-Excel.
- The all parameters of a test case in Fig. 2.2 are used for defining the test case for Mobile application.
- The given tests cased are associated with specific problem of Mobile application. Improved the functionality and security the defined MS-Excel workbook shares among with team members of Mobile application.

Case Study:

- Prepare the test case for the Mobile application for the starting the specific mobile application, where the following user requirement specification must check.

User Requirement specification

- Icon of the Mobile Application display in the beginning
- Check Menu list Display.
- Designing the test case using MS-Excel for Mobile application is as follows:

Refer to Page 2.11.

Test Case Design for Web Application:

- The test case design for the Web applications is by reviewing the following case study for developing the test cases using MS-Excel.
- The all parameters of a test case in Fig. 2.2 are used for defining the test case for Web application.
- The given tests cased are associated with specific problem of desktop application. Improved the functionality and security the defined MS-Excel workbook shares among with team members of Web application.

Case Study:

- Prepare the test case for the web application for the starting the specific Web application, where the following user requirement specification must check.

User Requirement specification

- Entering the URL of Web application.
- Web page navigation from Contact us page to home page.

A	B	C	D	E	F	G	H	I	J	K	L
Test Case ID	Test Case Name	Prerequisites	Objective	Description	Steps	Input	Expected Result	Actual Result	Status	Bug ID	Remark
1	Icon_Checking	Mobile Phone must On and install the mobile application	To verify the Icon of the Mobile Application display Correctly or not	Icon of the Mobile Application display in the beginning	1. Install the Application	.	Mobile application Must display after installation	It show the Mobile application in menu list	Pass	.	.
2	Menu_List_Checking	Mobile Phone must On and install the mobile application	To verify the Menu list of the Mobile Application display Correctly or not	Menu list display in the left top corner by three horizontal lines	1. Install the Application 2. Open the mobile application	.	Mobile application must display the menu list in the left corner by three horizontal lines	It show the Menu list icon correctly	Pass	.	.
3											

- Designing the Test case using MS-Excel for Web application is as follows:

A7											
f _K											
A	B	C	D	E	F	G	H	I	J	K	L
Test Case	Test Case Name	Prerequisites	Objective	Description	Steps	Input	Expected Result	Actual Result	Status	Bug ID	Remark
1	T101	URL_Checking	Internet connection must on	Entering the URL of Web application.	1. Type the URL in Address bar of the Browser. 2. Press Enter	URL of Mobile Application	After entering the URL must open the Web application.	The web application open successfully.	Pass	-	-
2			To Verify the URL is correctly working or not	Web page navigation from Contact us to home page.	1. Open the web application. 2. Visit the contact us page. 3. Click on home page.	-	Must open the home page form contact us page.	It show the home page of web application.	Pass	-	-
3	T102	Navigation_Checking	Internet Connection must on and open the web application								
4											
5											

2.5 WHITE BOX TESTING

- White box testing is a testing technique that examines the program structure and derives test data from the program logic/code.
- The white box testing is done on the basis of internal structures of software as defined by requirements, designs, coding standards and guidelines.
- The goal or objective of white box testing is to ensure that the test cases (developed by software testers by using white box testing) exercise each path through a program i.e., the test cases ensure that all internal structures in the program are developed according to design specifications.
- According to IEEE standards, white box testing, also known as structural testing or glass-box testing, is “testing that takes into account the internal mechanism of a system or component.”
- White box testing covers verification of software products as per structure, architecture, coding standards and guidelines of software. It mainly deals with the structure and design of the software product.
- The white box testing technique is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.
- In white box testing the software tester has access to the program's code and can examine it for clues to help him with his testing he/she can see inside the box.
- Based on what he/she sees, the tester may determine that certain numbers are more or less likely to fail and can tailor his testing based on that information.
- White-box testing is also called as structural testing or glass box testing or clear box testing. White-box testing is performed to test the program internal structure. Fig. 2.5 shows white box testing.

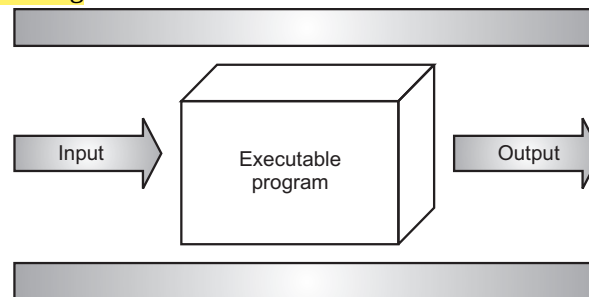


Fig. 2.5: White Box Testing

- White box testing involves looking at the structure of the code. When we know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification.

- And all internal components have been adequately exercised. In other words, White Box Testing (WBT) tends to involve the coverage of the specification in the code.
- Various types of which occur as part of white box testing are basis path testing, control structure testing and mutation testing.

Example for White Box Testing:

- A tester, usually a developer or engineer as well, studies the implementation code of a certain field on a Webpage, determines all legal (valid and invalid) and illegal inputs and verifies the outputs against the expected outcomes/results, which is also determined by studying the implementation code.

Advantages of White Box Testing:

1. The white box testing helps in removing the extra lines of code, which can bring in hidden defects.
2. In white box testing, as the knowledge of internal coding structure is prerequisite, it becomes very easy and simple to find out which type of input/data can help in testing the application effectively.
3. White box testing helps in optimizing the code.
4. In white box testing, the testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
5. In white box testing, due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Disadvantages of White-Box Testing:

1. In white box testing, as knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost.
2. In white box testing, it is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.
3. In white box testing, the tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
4. Test script maintenance can be a burden if the implementation changes too frequently.
5. It is difficult to maintain white-box testing as the use of specialized tools like code analyzers and debugging tools are required.

2.5.1 Basis Path Testing

- Basis path testing is a white box testing technique first proposed by Turn McCabe used for designing test cases.
- The basis path testing enables to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

- In the basic path testing, the **test** cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.
- Basis path testing, a **structured testing** or white box testing technique used for designing **test cases intended to examine all possible paths of execution at least once**.
- Basis path testing analyzes the control-flow graph of a program to find a set of linearly independent paths of execution.
- The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.
- Basis path testing generates test cases such that every path of the program has been exercised at least once.
- The basic path testing is used to specify the basic set of execution paths that are required to execute all the statements present in the program.

1. Flow Graph Notations:

- The flow graph depicts logical control flow using a diagrammatic notation. Each structured construct has a corresponding flow graph symbol.
- A simple notation for the representation of control flow, called a flow graph (or program graph).
- A flow graph $G = (V, E)$ is a direct graph, where V is a set of nodes and each node $v \in V$ represents a basic block. A basic **block** is a maximal sequence of statements, which satisfy all-or-nothing criterion.
- White box testing techniques rely on the structure of a program. Therefore, a representation of the program needs to be defined first.
- Flow graphs originating from compiler work are often widely used to depict logical control flowing of a program.
- The flow graph depicts logical control flow using the notation shown in Fig. 2.6. Each structured construct has a corresponding flow graph symbol.

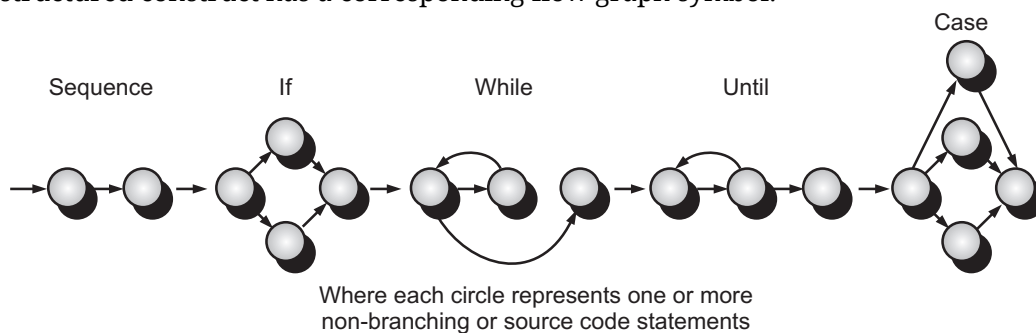


Fig. 2.6: Flow Graph Notations

- Referring to Fig. 2.6, each circle, called a flow graph node, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node.
- The arrows on the flow graph are called as edges or links. The arrows represent flow of control and are analogous to flowchart arrows.
- An edge in the flow graph must terminate at a node, even if the node does not represent any procedural statements (e.g., see the flow graph symbol for the if-then-else construct).
- Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.
- Fig. 2.7 shows conversion of flow chart to a flow graph.

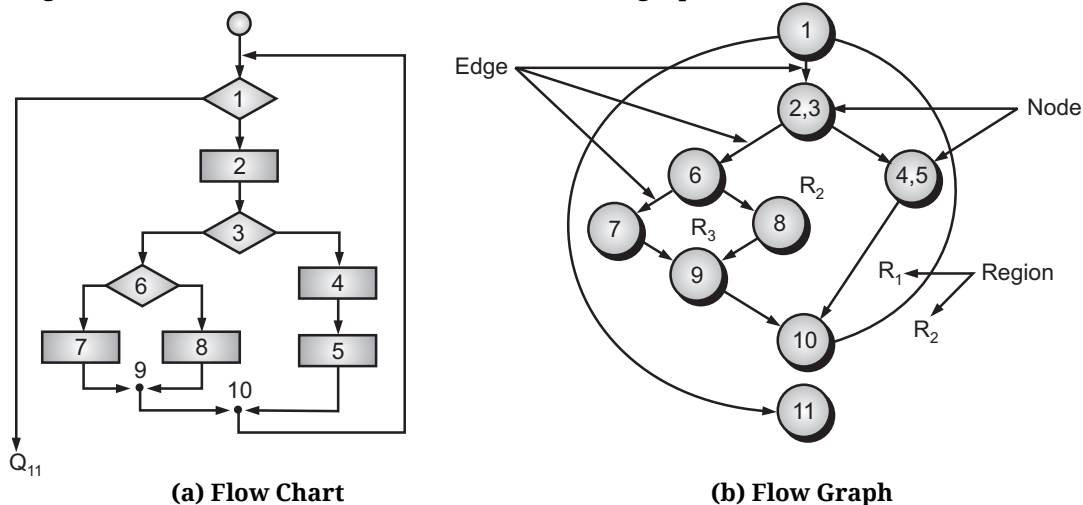


Fig. 2.7

2. Compound Logic:

- A compound condition occurs when one or more Boolean operators such as logical OR, AND, NAND, NOR etc., are present in a conditional statement.

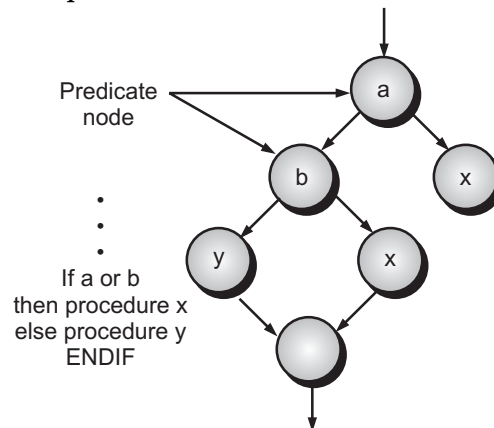


Fig. 2.8

- In Fig. 2.8, the Program Design Language (PDL) segment translates into the flow graph. Note that a separate node is created for each of the conditions a and b in the statement IF a OR b. Each node that contains a condition is called a predicate node and is characterized by two or more edges originating from it.

3. Independent Programs Paths:

- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
- When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.
- For example, a set of independent paths for the flow graph illustrated in Fig. 2.7 (b) is,

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

- Note that each new path introduces a new edge. The path,
1-2-3-4-5-10-1-2-3-6-8-9-10-1-11
is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges.
- Paths 1 through 4 constitute a basis set for the flow graph in Fig. 2.7 (b). That is, if you can design tests to force execution of these paths (a basis set), every statement in the program will have been guaranteed to be executed at least one time and every condition will have been executed on its true and false sides.
- It should be noted that the basis set is not unique. In fact, a number of different basis sets can be derived for a given procedural design.

4. Cyclomatic Complexity:

- Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.
- When used in the context of the basis path- testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- Cyclomatic complexity measures the amount of decision logic in the program module. Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.
- Cyclomatic complexity has a foundation in graph theory and provides you with an extremely useful software metric.

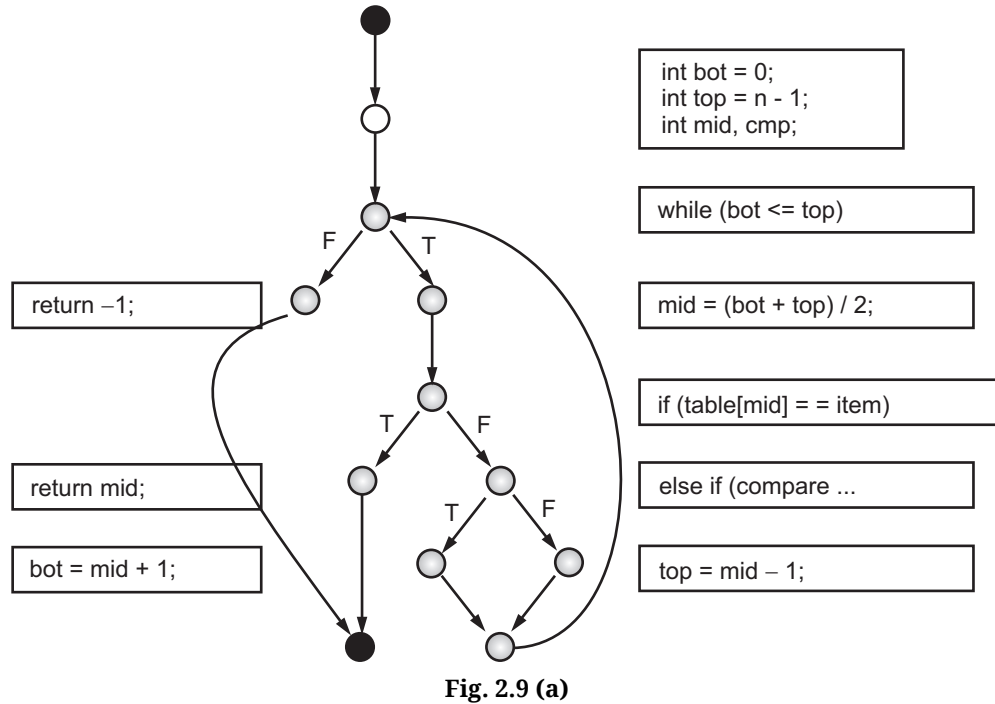
- Complexity is computed in one of three ways:
 - (i) The number of regions of the flow graph corresponds to the Cyclomatic complexity.
 - (ii) Cyclomatic complexity $V(G)$ for a flow graph G is defined as $V(G) = E - N + 2$ where, E is the number of flow graph edges and N is the number of flow graph nodes.
 - (iii) Cyclomatic complexity $V(G)$ for a flow graph G is also defined as $V(G) = P + 1$ where, P is the number of predicate nodes contained in the flow graph G .
- Referring once more to the flow graph in Fig. 2.7 (b), the Cyclomatic complexity can be computed using each of the algorithms just noted:
 - (i) The flow graph has four regions.
 - (ii) $V(G) = 11 \text{ edges} - 9 \text{ nodes} + 2 = 4$.
 - (iii) $V(G) = 3 \text{ predicate nodes} + 1 = 4$.
- Therefore, the Cyclomatic complexity of the flow graph in Fig. 1.7 (b) is 4. More important, the value for $V(G)$ provides we with an upper bound for the number of independent paths that form the basis set and, by implication, an upper bound on the number of tests that must be designed and executed to guarantee coverage of all program statements.

Example:

- Consider simple program for Binary Search given below to calculate its cyclomatic complexity:

```
intBinSearch (char *item, char *table[], int n)
{
    int bot = 0;
    int top = n - 1;
    int mid, cmp;
    while (bot <= top) {
        mid = (bot + top) / 2;
        if (table[mid] == item)
            return mid;
        else if (compare(table[mid], item) < 0)
            top = mid - 1;
        else
            bot = mid + 1;
    }
    return -1; // not found
}
```

- The flow graph for this program is shown in Fig. 2.9 (a).



- In this example,

$$V(G) = E - N + 2$$

Where E no. of edges = 14

N no. of nodes = 12

$$V(G) = 14 - 12 + 2 = 4$$

OR

$$V(G) = P + 1$$

Where P predicates are no. of decision points = 3

$$= 3 + 1 = 4$$

The minimum number of test paths to achieve maximum coverage as shown in Fig. 2.9 (b).

- In this example the $V(G)$ is 4, thus there are 4 linearly independent paths and this is the minimum number of test paths to achieve maximum coverage,
 - Path 1 : 0-1-2-3-11
 - Path 2 : 0-1-2-4-5-6-11
 - Path 3 : 0-1-2-4-5-7-8-10-2-3-11
 - Path 4 : 0-1-2-4-5-7-9-10-2-4-5-6-11
- These test paths achieve maximum coverage and any path other than these will dependent on these paths.

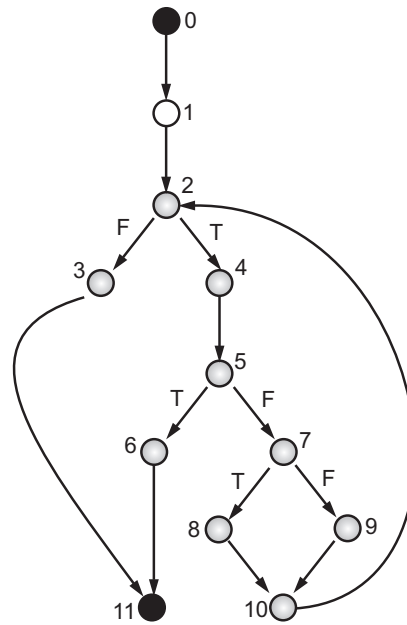


Fig. 2.9 (b)

5. Graph Matrix:

- The procedure for deriving the flow graph and even determining a set of basis paths is amenable to mechanization.
- To develop a software tool that assists in basis path testing a data structure called a graph matrix can be quite useful.
- A graph matrix is a square matrix whose size is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node and matrix entries correspond to connections between nodes.
- Graph matrix is a data structure that enables to develop a software tool that helps in carrying out basis path testing.
- Graph matrix is defined as, a data structure used to represent the flow graph of a program in a tabular form.
- The graph matrix is also used to evaluate the control structures present in the program during testing.
- Graph matrix is a square matrix of the size $N \times N$, where N is the number of nodes in the flow graph.
- An entry is made in the matrix at the intersection of i^{th} row and j^{th} column if there exists an edge between i^{th} and j^{th} node in the flow graph. Every entry in the graph matrix is assigned some value, known as link weight.
- Adding link weights to each entry makes the graph matrix a useful tool for evaluating the control structure of the program during testing.

- The procedure for deriving the flow graph and even determining a set of basis paths is amenable to mechanization.
- To develop a software tool that assists in basis path testing, a data structure called a graph mod can be quite useful.
- A graph matrix is a square matrix whose size (i.e., number of rows and columns) is equal to the number of nodes on the flow graph.
- Each row and column corresponds to an identified node, and matrix entries correspond to connections (an edge) between nodes.
- A simple example of a flow graph and its corresponding graph matrix is shown in Fig. 2.10.
- In Fig. 2.10 each node on the flow graph is identified by numbers, while each edge is identified by letters.
- A letter entry is made in the matrix to correspond to a connection between two nodes. For example, node 3 is connected to node 4 by edge b.

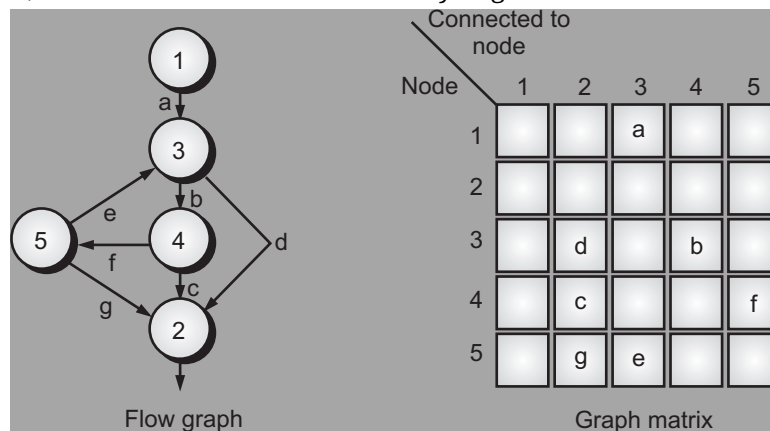


Fig. 2.10

- At this point, the graph matrix is nothing more than a tabular representation of a flow graph. However, by adding a link weight to each matrix entry, the graph matrix can become a powerful tool for evaluating program control structure during testing.
- The link weight provides additional information about control flow. In its simplest form, the link weight is 1 (a connection exists) or 0 (a connection does not exist).
- But link weights can be assigned other, more interesting properties. Some of them are given below:
 - (i) The memory required during traversal of a link.
 - (ii) The probability that a link (edge) will be executed.
 - (iii) The resources required during traversal of a link.
 - (iv) The processing time expended during traversal of a link.

6. Deriving Test Cases:

- In deriving test cases, basis path testing is presented as a series of steps and test cases are developed to ensure that all statements within the program get exercised at least once while performing testing.
- While performing basis path testing, initially the basis set (independent paths in the program) is derived. The basis set can be derived using the following steps:

Step 1: Draw the Flow Graph of the Program: A flow graph is constructed using symbols previously discussed. For example, a program to find the greater of two numbers is given below:

```

procedure greater;
    integer: a, b, c = 0;
    enter the value of a;
    enter the value of b;
    if a > b then
        c = a;
    else
        c = y;
    end greater
  
```

Fig. 2.11 shows the flow graph for above program.

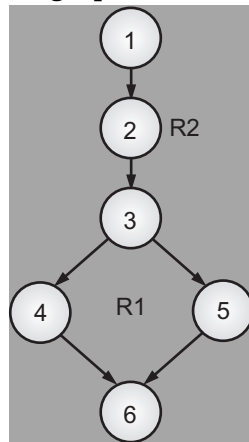


Fig. 2.11: Flow Control to find the Greater between Two Numbers

Step 2: Compute Cyclomatic Complexity: Cyclomatic complexity of the program can be computed using the flow graph as shown in Fig. 2.11.

CC = 2 as there two regions R1 and R2

or

CC = 6 edges - 6 nodes + 2 = 2

or

CC = 1 predicate node + 1 = 2

Step 3: Determine all Independent Paths through the Program: For the flow graph depicted in Fig. 2.11, the independent paths are:

P1: 1-2-3-4-6

P2: 1-2-3-5-6

Step 4: Prepare Test Cases: Test cases are prepared to implement the execution of all independent paths in the basis set. The program is then tested for each test case and the produced output is compared with the desired output.

2.5.2 Control Structure Testing

- Control structure testing is used to enhance the coverage area by testing various control structures (which include logical structures and loops) present in the program.
- Note that basis path testing is used as one of the techniques for control structure testing.
- Various types of testing performed under control structure testing are condition testing, data flow testing and loop testing.
 - The **Condition testing** is a test case design method that exercises the logical conditions contained in a program module.
 - The **data flow testing** method selects test paths of a program according to the locations of definitions and uses of variables in the program.
 - The **loop testing** is a white box testing technique that focuses exclusively on the validity of loop constructs. The four classes of loops can be defined, Simple loops, Concatenated loops, Nested loops and Unstructured loops.

Condition Testing:

- It is a test case design technique that exercises or evaluates the logical conditions contained in a program module.
- In condition testing, test cases are derived to determine whether the logical conditions and decision statements are free from errors.
- The value of a variable would determine the flow of the software application or product or system which can easily be understood with an example as in the case of Boolean operators, such as True or False.
- If the logical condition is True then the software application traverses along a specific path, if the logical condition is False it will traverse along another path.
- Similarly the value of an integer variable like the Age can make the application behave in a certain manner.
- For example, if Age is 21 or above certain rules may be triggered and if Age is less than 21 certain other conditions may be triggered.

- The common types of logical conditions that are tested using condition testing are given below:
 1. A relational expression such as $E1 \text{ op } E2$, where $E1$ and $E2$ are arithmetic expressions and 'op' is an operator.
 2. A simple condition such as any relational expression preceded by a NOT (\sim) operator. For example, $(\sim E1)$ where, $E1$ is an arithmetic expression.
 3. A compound condition which is formed by combining two or more simple conditions using Boolean operators. For example, $(E1 \ \& \ E2) \ | \ (E2 \ \& \ E3)$, where $E1$, $E2$ and $E3$ are arithmetic expressions and '&' and '|' represent AND and OR operators, respectively.
 4. A Boolean expression consisting of operands and a Boolean operator such as AND, OR or NOT. An example, $A|B$ is a Boolean expression, where A and B are operands and $|$ represents the OR operator.
- Condition testing is performed using different strategies namely, branch testing, domain testing and branch and relational operator testing.
 1. **Branch testing** executes each branch (such as 'if' statement) present in the module of a program at least once to detect all the errors present in the branch.
 2. **Domain testing** tests relational expressions present in a program. For this, domain testing executes all statements of a program that contain relational expressions.
 3. **Branch and Relational Operator Testing** tests the branches present in the module of a program using condition constraints. For example,


```
if a > 20
then
    print big
```

In this case, branch and relational operator testing verifies that when the above code is executed, it produces the output 'big' only if the value of variable a is greater than 20.
- A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (\neg) operator. A relational expression takes the following form,

$$E1 \text{ <relational-operator> } E2$$

where, $E1$ and $E2$ are arithmetic expressions and <relational-operator> is one of the $<$, \leq , $=$, \neq (non-equality), $>$ or \geq .
- A compound condition is composed of two or more simple conditions, Boolean operators (include OR ($|$), AND ($\&$) and NOT (\neg)) and parentheses.
- A condition without relational expressions is referred to as a Boolean expression. Therefore, the possible types of elements in a condition include a Boolean operator, a Boolean variable, a pair of parentheses, a relational operator or an arithmetic expression.

- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include Boolean operator errors (incorrect/missing/extra Boolean operators), Boolean variable errors, Boolean parenthesis errors, relational operator errors and arithmetic expression errors.
- The condition testing method focuses on testing each condition in the program to ensure that it does not contain errors.

Data Flow Testing:

- A program is actually all about manipulating data which can be generally classified into two categories namely, data that defines the value of a variable and data that refers to the value of a variable.
- In data flow testing, test cases are derived to determine the validity of variables definitions and their uses in a program. This testing ensures that all variables are used properly in a program.
- To specify test cases, data flow testing technique uses information such as location at which the variables are defined and used in a program.
- The data flow testing technique selects test paths of a program according to the locations of definitions and uses of variables in the program.
- To illustrate the data flow testing technique, assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables.
- For a statement with S as its statement number,
$$DEF(S) = \{X \mid \text{statement } S \text{ contains a definition of } X\}$$
$$USE(S) = \{X \mid \text{statement } S \text{ contains a use of } X\}$$
- If statement S is an if or a loop statement, its DEF set is empty and its USE set is based on the condition of statement S.
- The definition of variable X at statement S is said to be live at statements S' if there exists a path from statement S to statement S' that contains no other definition of X.
- A Definition-Use (DU) chain of variable X is of the form [X, S, S'], where S and S' are statements numbers, X is in DEF(S) and USE(S') and the definition of X in statement S is live at statement S'.
- For performing data flow testing, a definition use graph is built by associating the program variables with nodes and edges of the control flow graph.
- Once, these variables are attached, test cases can easily determine which variable is used in which part of a program and how data is flowing in the program. Thus, data flow of a program can be tested easily using specified test cases.
- One simple data flow testing strategy is to require that every DU chain be covered at least once. We refer to this strategy as the DU testing strategy.

- It has been shown that DU testing does not guarantee the coverage of all branches of a program.
- However, a branch is not guaranteed to be covered by DU testing only in rare situations such as if-then-else constructs in which the if-then part has no definition or any variable and the else part does not exist.
- In this situation, the else branch of the statement is not necessarily covered by DU testing.
- A number of data flow testing strategies have been studied and compared. The interested reader is urged to consider these other references.

Loop Statement:

- Loop testing is a white box testing technique. Loop testing is used to check the validity of loops present in the program modules.
- Loops are the cornerstone for the vast majority of all algorithms implemented in software. And yet, we often pay them little heed while conducting software tests.
- Loop testing focuses exclusively on the validity of loop constructs.
- Fig. 2.12 shows four different types of loops.

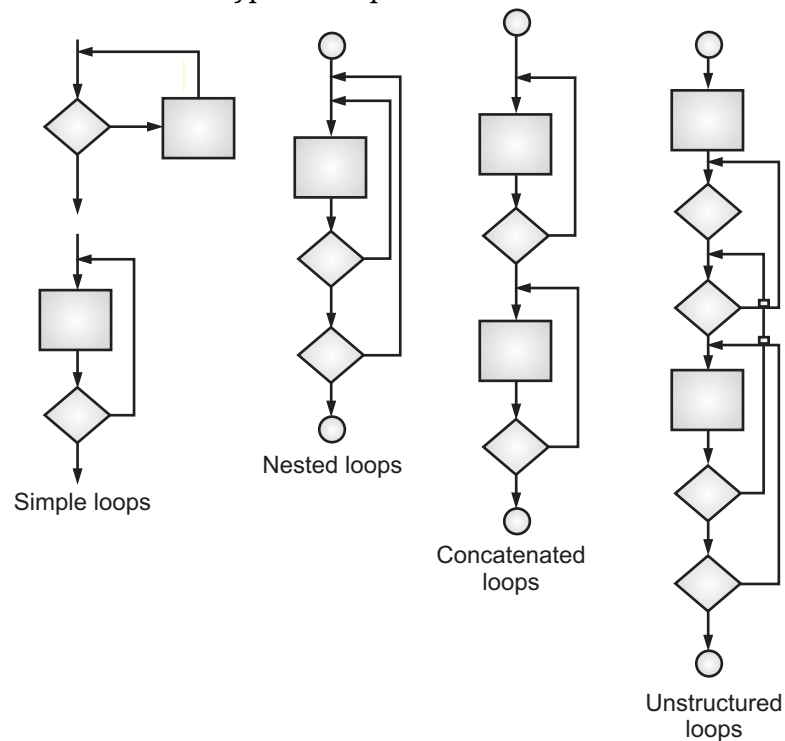


Fig. 2.12: Types of Loops

Simple Loops:

- The simple loop refers to a loop that **has no other loops in** it. Consider a simple loop of size n . Size n of the loop indicates that the loop can be traversed n times, i.e., n number of passes are made through the loop.
- The following sets of tests can be applied to simple loops, where 'n' is the maximum number of allowable passes through the loop:
 1. **Skip the loop entirely.**
 2. Only one pass through the loop.
 3. Two passes through the loop.
 4. 'm' passes through the loop where $m < n$.
 5. $n-1$, n , $m+1$ passes through the loop.

Nested Loops:

- Loops within loops are known as nested loops.
- If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increased. This would result in an impractical number of tests.
- Boris Beizer suggests an approach that will help to reduce the number of tests. These suggestions are given below:
 1. Start at the innermost loop. Set all other loops to minimum values.
 2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values add other tests for out-of-range or excluded values.
 3. Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to 'typical' values.
 4. Continue until all loops have been tested

Concatenated Loops:

- The loops containing several loops that may be dependent or independent. In case the loops are dependent on each other the steps in nested loops are followed.
- On the other hand, if the loops are independent of each other, the steps in simple loops are followed.
- Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other.
- However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.
- When the loops are not Independent, the approach applied to nested loops is recommended.

Unstructured Loops:

- Such loops are difficult to test; therefore, they should be redesigned so that the use of structured programming constructs can be reflected.

2.6 BLACK BOX TESTING

- Black box (or functional) testing checks the functional requirements and examines the input and output data of these requirements.
- Once, the specified function for which the software product or system has been designed is known, tests are performed to ensure that each function is working properly. This is referred to as black box testing.
- Black box testing, also known as behavioral testing focuses on the functional requirements of the software.
- Black box testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester.
- The black box testing is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.
- In black box testing, the tester only knows what the software is supposed to do he/she cannot look in the box to see how it operates.
- If he/she types in a certain input, he/she gets a certain output. He/she does not know how or why it happens, just that it does.
- The black box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Fig. 2.13 shows black box testing.

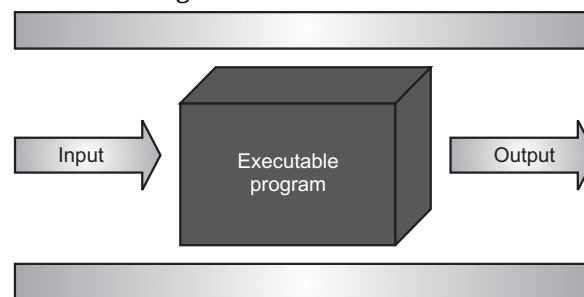


Fig. 2.13: Black box Testing

Example for Black Box Testing:

- A tester, without knowledge of the internal structures of a Website, tests the web pages by using a browser; providing inputs using clicks, keystrokes etc., and verifying the outputs against the expected outcome.

- Black box testing attempts to find errors in the following categories:
 1. Incorrect or missing functions.
 2. Interface errors.
 3. Errors in data structures.
 4. Behavior errors.
 5. Initialization errors.
- Various methods used in black box testing are equivalence class partitioning, boundary value analysis and cause effect graphing.
 1. In boundary value analysis, the boundary values of the equivalence classes are considered and tested.
 2. In equivalence class partitioning, the test inputs are classified into equivalence classes such that one input checks (validates) all the input values in that class.

Advantages of Black Box Testing:

1. Black box testing is efficient when used on large systems.
2. The tester and developer in black-box testing are independent of each other so testing is balanced and unprejudiced.
3. Tester can be non-technical.
4. There is no need for the tester to have detailed functional knowledge of system.
5. Tests will be done from an end user's point of view, because the end user should accept the system.
6. Black-box testing helps to identify vagueness and contradictions in functional specifications.
7. Test cases in black-box testing can be designed as soon as the functional specifications are complete.

Disadvantages of Black-Box Testing:

1. In black-box testing cases in black-box testing are challenging to design without having clear functional specifications.
2. In black box testing, it is difficult to identify tricky inputs if the test cases are not developed based on specifications.
3. It is difficult to identify all possible inputs in limited testing time. As a result, writing test cases may be slow and difficult.
4. There are chances of having unidentified paths during the testing process.
5. In black box testing, there is a high probability of repeating tests already performed by the programmer.

2.6.1 Boundary Values Analysis

- The simplest view of software is to divide its world into two parts i.e., the data (or its domain) and the program.
- The data is the keyboard input, mouse clicks, disk files, printouts, and soon. The program is the executable flow, transitions, logic and computations.
- A common approach to software testing is to divide up the test work along the same lines.
- The amount of data handled by even the simplest programs can be much more. Remember all the possibilities of input data for performing simple addition on a calculator? Say 1+1, 1+2, 1+3, 11+11, ... 1+99999, 1+9999999999999999....., or a word processor, a missile guidance system, or a stock trading program.
- These test cases need to be reduced to the effective test cases by equivalence partitioning based on a few key concepts like boundary conditions, sub-boundary conditions.
- A boundary value is any input or output value on the edge of an equivalence partition. Boundary Value Analysis (BVA) is a black-box test design technique where test case is designed by using boundary values.
- Boundary Value Analysis (BVA) is a software testing technique that selects the input data which represents boundary values. These data includes both boundary and each side of boundary which should be in the smallest increment.
- BVA is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition).
- A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries. Normally, BVA is part of stress and negative testing.
- Fig. 2.14 shows a simple example for boundary value analysis.
- Suppose you have very important tool at office, accept valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Valid range 8-12, Invalid range 7 or less than 7 and Invalid range 13 or more than 13.

Invalid Partition	Valid Partition	Invalid Partition
Less than 8	8 - 12	More than 12

Fig. 2.14

- Write test cases for valid partition value, invalid partition value and exact Boundary Value (BV).
 - **Test Cases 1:** Consider password length less than 8.
 - **Test Cases 2:** Consider password of length exactly 8.
 - **Test Cases 3:** Consider password of length between 9 and 11.
 - **Test Cases 4:** Consider password of length exactly 12.
 - **Test Cases 5:** Consider password of length more than 12.
- The best way to describe boundary condition testing is explained by following example. If software can operate on the edge of its capabilities, it will almost certainly operate well under normal conditions.
- Boundary conditions are special because programming, by its nature, is vulnerable to problems at its edges.
- The below code shows how a boundary condition problem can make its way into a very simple program.

A simple basic Program Demonstrating a Boundary Condition Bug:

```
1: Rem Create a 10 element integer array
2: Rem Initialize each element to -1
3: Dim data (10) As Integer
4: Dim i As Integer
5: For i = 1 To 10
6: data (i) = -1
7: Next i
8: End
```

- The purpose of this code is to create a 10-element array and initialize each element of the array to 1. It looks fairly simple. An array data of 10 integers and a counter i are created i.e. data (10). A for loop runs from 1 to 10, and each element of the array from 1 to 10 is assigned a value of 1. Where's the boundary problem?
- In most BASIC scripts, when an array is dimensioned with a stated range in this case, Dimdata (10) as Integer the first element created is 0, not 1. This program actually creates a data array of 11 elements from data (0) to data (10). The program loops from 1 to 10 and initializes get initialized.
- When the program completes, the array values look like this:

data (0) = 0	data (6) = 1
data (1) = 1	data (7) = 1
data (2) = 1	data (8) = 1
data (3) = 1	data (9) = 1
data (4) = 1	data (10) = 1
data (5) = 1	

- Notice that data (0)'s value is 0, not 1. If the same programmer later forgot about, or a different programmer wasn't aware of how this data array was initialized, he might use the first element of the array, data (0), thinking it was set to 1. Problems such as this are very common and, in large complex software, can result in very nasty bugs.
- **Types of Boundary Conditions:** For the following types:
 1. Numeric Speed
 2. Character Location
 3. Position Size
 4. Quantity.
- Following are the boundary conditions that need to be tested:
 1. First/Last
 2. Min/Max
 3. Start/Finish
 4. Empty/Full
 5. Shortest/Longest
 6. Slowest/Fastest
 7. Largest/Smallest
 8. Highest/Lowest.
- Testing outside the boundary is usually as simple as adding one, or a bit more, to the maximum value and subtracting one, or a bit more, from the minimum value. For example:
 1. First1/Last+1
 2. Start1/Finish+1
 3. Less than Empty/More than Full
 4. Largest+1/Smallest1
 5. Min1/Max+1.

Examples:

1. May be the software has a data-entry field for a 9-digit ZIP code. Try 00000-0000, the simplest and smallest. Try entering 99999-9999 as the largest. Try entering one more or one less digit than what's allowed.
2. If a program allows us to print multiple pages onto a single page, try printing just one (the standard case) and try printing the most pages that it allows. If we can, try printing zero pages and one more than it allows.
3. If a text entry field allows 1 to 255 characters, try entering 1 character and 255 characters as the valid partition. We might also try 254 characters as a valid choice. Enter 0 and 256 characters as the invalid partitions.

2.6.2 Equivalence Partitioning

- Equivalence Partitioning (EP) is a black box testing technique. Equivalence partitions are also known as Equivalence Class Partition (ECP).
- Equivalence partitioning testing divides the input domain of a program into classes of data from which test cases can be derived.
- The goal of equivalence partitioning is to reduce the set of possible test cases into a smaller, manageable set that still adequately tests the software.
- Equivalence partitioning is the process of methodically reducing the huge (infinite) set of possible test cases into a much smaller, but still equally effective, set.
- Equivalence partitioning is a software testing technique to minimize number of permutation and combination of input data.
- In equivalence partitioning, data is selected in such a way that it gives as many different output as possible with the minimal set of data.
- If software behaves in an identical way for a set of value, then the set is termed as equivalence class or a partition. It can be assumed safely that functionality of the software will be same for any data value from the equivalence class or partition.
- In equivalence partitioning, input data is analyzed and divided into equivalence classes which produce different output.
- Now, data from these classes can be representative of all the input values that your software expects.
- For equivalence classes, it can be assumed that software will behave in exactly same way for any data value from the same partition.
- So essentially, there are two steps that we need to follow if you want to use equivalence partitioning in your projects - * Identifying equivalence classes or partition * Picking one value from each partition for the complete coverage.
- Main advantage of using equivalence partitioning technique is that testing efforts are minimized and at the same time coverage is also ensured.
- Using equivalence partitioning technique, redundancy of test cases is removed by eliminating data which does not produce different output.

For example, consider a very simple function for awarding grades to the students. These programs follow this guideline to award grades,

- Marks 00 - 39 ----- Grade D
- Marks 40 - 59 ----- Grade C
- Marks 60 - 70 ----- Grade B
- Marks 71 - 100 ----- Grade A

Based on the equivalence partitioning techniques, partitions for this program could be as follows:

- Marks between 0 to 39 - Valid Input
- Marks between 40 to 59 - Valid Input
- Marks between 60 to 70 - Valid Input
- Marks between 71 to 100 - Valid Input
- Marks less than 0 - Invalid Input
- Marks more than 100 - Invalid Input
- Non numeric input - Invalid Input

From the example above, it is clear that from infinite possible test cases, (any value between 0 to 100, Infinite values for >100, <0 and non numeric) data can be divided into seven distinct classes. Now even if you take only one data value from these partitions, your coverage will be good.

Fig. 2.15 shows the Calculator's Edit menu selected to display the copy command. There are five ways to perform this function. For copy:

- we click the Copy menu item,
- type c or C when the menu is displayed,
- or press Ctrl+c or Ctrl+Shift+c.

We could partition these five input paths into three:

- Clicking the command on the menu,
- typing a c, or
- pressing Ctrl+c.

Consider the possibilities for entering a filename in the standard Save As dialog box, (See Fig. 2.16).

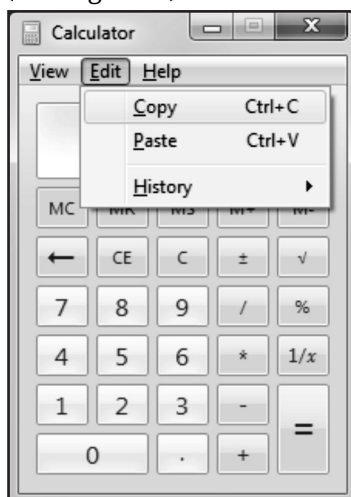


Fig. 2.15

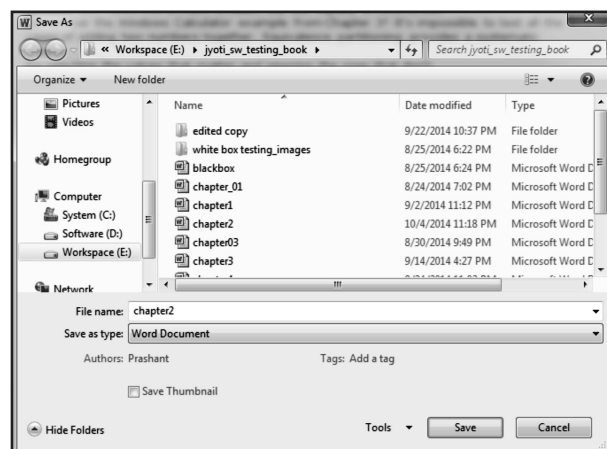


Fig. 2.16

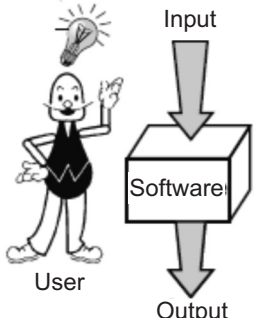
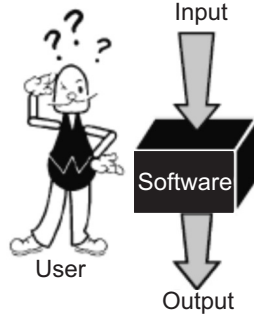
- A Windows filename can contain any characters except \/: * ? " <> and |. Filenames can have from 1 to 255 characters. If you're creating test cases for filenames, you will have equivalence partitions for
 - valid characters,
 - invalid characters,
 - valid length names, names that are too short, and
 - names that are too long.

2.7 DIFFERENCES BETWEEN WBT AND BBT

- Following table differentiate between White Box Testing (WBT) and Black Box Testing (BBT):

Sr. No.	White Box Testing	Black Box Testing
1.	The purpose of white box testing is to test the internal structure of software.	The purpose of black box testing is to test the functionality of software.
2.	White box testing performed in the early stages of testing.	Black box testing performed in the later stages of testing.
3.	In white box testing knowledge of the internal structure of a program is required for generating test case.	In black box testing no knowledge of the internal structure of a program is required to generate test case.
4.	In white box testing test cases are generated on the basis of the internal structure or code of the module to be tested.	In black box testing internal structure of modules or programs is not considered for selecting test cases.
5.	White box testing also known as clear box testing, structural testing or code based testing.	Black box testing is also known as closed box testing, data driven testing and functional testing.
6.	Normally white box testing done by testers and developers.	Black box testing performed by end users and also by testers and developers.
7.	In white box testing internal workings are fully known and the tester can design test data accordingly.	In black box testing is based on external expectations - internal behavior of the application is unknown.
8.	White box testing is time consuming type of testing.	Black box testing is the least time consuming and exhaustive.

contd. ...

9.	White box testing is suited for algorithm testing.	Black box testing not suited to algorithm testing.
10.	In white box testing data domains and internal boundaries can be better tested.	In black box testing can only be done by trial and error method.
11.	Diagram: 	Diagram: 

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

- Which is a set of guidelines that explains test design and determines how testing needs to be done?
 - Test plan
 - Test Strategy
 - Test Report
 - None of the mentioned
- Software testing techniques include,
 - White box testing (or structural testing or glass box testing) is performed to test the program internal structure.
 - Black box (or functional testing) testing checks the functional requirements and examines the input and output data of these requirements.
 - Both (a) and (b)
 - None of the mentioned
- The term testability refers to,
 - The degree to which a software/system requirement is stated in terms that permits the establishment of the test criteria.
 - The performance of tests to determine whether those criteria have been met.
 - Both (a) and (b)
 - None of the mentioned

-
4. Test characteristics includes,
 - (a) High probability of detecting errors
 - (b) Choose the most appropriate test
 - (c) Moderate the test (considered good if it is neither too simple, nor too complex).
 - (d) All of the mentioned
 5. Which provides the description of inputs and their expected outputs to observe whether the software is working correctly or not?
 - (a) Test plan
 - (b) Test case
 - (c) Test strategy
 - (d) None of the mentioned
 6. Test case gives detailed information about,
 - (a) testing strategy and testing process
 - (b) Testing preconditions
 - (c) expected output
 - (d) All of the mentioned
 7. Which is software testing that examines the functionality of an application based on the specifications?
 - (a) black-box testing
 - (b) white-box testing
 - (c) gray-box testing
 - (d) None of the mentioned
 8. Which is software testing technique in which product's underlying/internal structure, design, and coding in order to verify input-output flow and improve design, usability and security?
 - (a) black-box testing
 - (b) white-box testing
 - (c) gray-box testing
 - (d) None of the mentioned
 9. In which testing technique the inputs are partitioned into groups or more literally partitions and only one input from every group is tested to find the results?
 - (a) Equivalence Partitioning
 - (b) Boundary Value Analysis
 - (c) Basic Path Testing
 - (d) None of the mentioned
 10. Which is a white box technique for designing test cases and analyzes the control-flow graph of a program to find a set of linearly independent paths of execution?
 - (a) equivalence partitioning
 - (b) control structure testing
 - (c) basis path testing
 - (d) boundary value analysis
 11. Which is software metric used to indicate the complexity of a program and quantitative measures of the number of linearly independent paths through a program's source code?
 - (a) Cyclomatic complexity
 - (b) Test complexity
 - (c) Strategy complexity
 - (d) None of the mentioned
-

12. Control flow Which testing determines the execution order of statements or instructions of the program through a control structure?

- (a) Data flow (b) Branch flow
(c) Control flow (d) None of the mentioned

Answers

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

Q. II Fill in the Blanks:

- The test _____ describes the approach that the test team will use to test the software.
- To perform _____ testing, the tester should have a thorough knowledge of the program internals along with the purpose of developing the software.
- _____ testing enables to generate test cases such that every path of the program has been exercised at least once.
- A _____ graph represents the logical control flow within a program.
- _____ structure testing is used to enhance the coverage area by testing various control structures (which include logical structures and loops) present in the program.
- _____ testing is performed using different strategies, namely, branch testing, domain testing and branch and relational operator testing.
- _____ testing executes each branch (like 'if statement) present in the module of a program at least once to detect all the errors present in the branch.
- _____ testing tests relational expressions present in a program. For this, domain testing executes all statements of the program that contain relational expressions.
- In data flow testing, test cases are derived to determine the validity of _____ definitions and their uses in the program.
- _____ testing is used to check the validity of loops present in the program modules.
- Loops within loops are known as _____ loops.
- In equivalence class partitioning, the test inputs are classified into equivalence _____ such that one input checks (validates) all the input values in that class.
- In boundary value analysis, the _____ values of the equivalence classes are considered and tested.
- To perform _____ in a planned and systematic manner, software testing strategy is developed.
- The _____ is a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not.

16. _____ testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.
17. _____ partitioning testing technique divides the input values which are provided to the software into different groups or classes.
18. _____ complexity is a software metric (a software metric is a numerical measurement of a software attribute's time/schedule, quality, size, and cost) used to describe the complexity of a program.
19. _____ is a popular black box testing technique tests the boundary value of each input group (including both valid and invalid inputs).
20. Software testing _____ helps in designing better test cases, which are a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

Answers

1. strategy	2. white box	3. Basis path	4. flow
5. Control	6. Condition	7. Branch	8. Domain
9. variables	10. Loop	11. nested	12. classes
13. boundary	14. testing	15. test case	16. black box
17. Equivalence	18. Cyclomatic	19. BVA	20. techniques

Q. III State True or False:

1. The test strategy defines guidelines for test approach to be followed in order to achieve the test objectives and execution of test in the software.
2. White box testing techniques analyze the internal structures the used data structures, internal design, code structure for testing process.
3. Black box testing is also called glass box testing or clear box testing or structural testing.
4. In basis path testing control flow graphs are made from code and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.
5. Cyclomatic complexity is a measure of the logical complexity of the software and is used to define the number of independent paths.
6. In the loop testing, we will test the loops such as while, for and do-while, etc. and also check for ending condition if working correctly or not.
7. A testing strategy should be developed with the intent to provide the most effective and efficient way of testing the software.
8. White box testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

9. Test case designing includes preconditions, case name, input conditions, and expected result.
10. A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly or not.
11. The purpose of a test case is to determine if different features within a system are performing as expected and to confirm that the system satisfies all related standards, guidelines and customer requirements.
12. Loop testing is a white box testing technique used to test loops in the program.
13. In the concatenated loops, if two loops are independent of each other then they are tested using simple loops or else test them as nested loops.
14. Black box testing is a method of software testing that examines the functionality of an application based on the specifications.
15. Boundary value analysis is one of the widely used case design technique for white box testing.
16. Control flow testing is a testing technique that comes under white box testing, used to determine the execution order of statements or instructions of the program through a control structure.
17. Software testing techniques are basically certain procedures which help every software development project improve its overall quality and effectiveness.
18. A control flow graph is a directed graph that depicts a program's or module's control structure. V number of nodes/vertices and E number of edges make up a control flow graph (V, E).
19. The Cyclomatic complexity $V(G)$ is said to be a metric for a program's logical complexity.
20. Control flow graph is a graphical representation of control flow shows all the paths that can be traversed during a program execution.
21. Software testing strategy is an approach which incorporates planning of the steps to test the software along with the planning of time, effort & resources that will be required to test the software.
22. Basis path testing is a technique of selecting the paths in the control flow graph that provide a basis set of execution paths through the program or module.
23. Basis path testing in software engineering is a black box testing method in which test cases are defined based on flows or logical paths that can be taken through the program.
24. Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.

25. Testability is the degree to which a system facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

Answers

1. (T)	2. (T)	3. (F)	4. (T)	5. (T)	6. (T)	7. (T)	8. (F)	9. (T)	10. (T)
11. (T)	12. (T)	13. (T)	14. (T)	15. (F)	16. (T)	17. (T)	18. (T)	19. (F)	20. (T)
21. (T)	22. (T)	23. (F)	24. (T)	25. (T)					

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define test strategy.
2. Define test case.
3. List techniques for software testing.
4. Define data flow testing.
5. Define white box testing.
6. What is basic path testing?
7. Define loop testing.
8. Define data flow testing.
9. Define equivalence partitioning.
10. List any two test characteristics.
11. Define black box testing.
12. Define testability.
13. Define Cyclomatic complexity.
14. What is the purpose of BVA?

(B) Long Answer Questions:

1. What is test strategy? What is its purpose in testing? Also state its characteristics.
2. What is test case? How to create it? Explain with example.
3. What are the characteristics of test?
4. What is testability? What are its characteristics? Explain two of them in detail.
5. With the help of diagram and example describe white box testing. Also state its advantages and disadvantages.
6. What is black box testing? Describe with diagram and example.
7. How to design a test case in MS-Excel? Design test cases for mobile and web applications.
8. What is control flow testing? How it works? Explain its types.

9. Describe basic path testing with example.
10. What is loop testing? Explain with example.
11. With the help of example describe test case template.
12. Differentiate between white box testing and black box testing.
13. What is data flow testing? Describe in detail.
14. With the help of example describe BVA and EP? Also compare them.
15. What is Cyclomatic complexity and Graph matrix? Explain with example.



Levels of Testing

Objectives...

- To understand Levels of Testing
- To learn Unit Testing, System Testing and Integration Testing
- To study Usability Testing and Accessibility Testing

3.0 INTRODUCTION

- We already known software testing is a process of executing a program or application with the intent of finding the software bugs.
- Levels of testing include the different methodologies that can be used while conducting software testing.
- Software is tested at different levels. Initially, individual units are tested and once they are tested, they are integrated and checked for interfaces established between them.
- After this, the entire software is tested to ensure that the output produced is according to user requirements.
- Fig. 3.1 there are four levels of software testing, namely, unit testing, integration testing, system testing and acceptance testing.

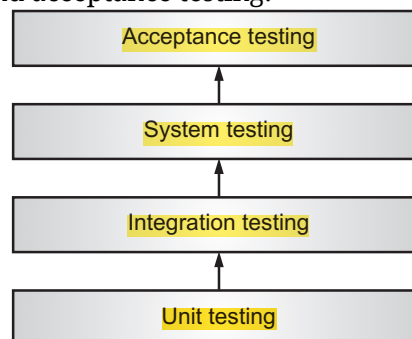


Fig. 3.1: Levels of Testing

- Following are the main levels of software testing as shown in Fig. 3.1.
 1. **Unit Testing:** Unit testing is the **basic level of testing**. Unit testing is a level of the software testing process where individual units/components of a software/system are tested. The purpose of unit testing is to **validate that each unit of the software performs as designed**.

2. **Integration Testing:** In integration testing, all unite tested modules are integrated and tested for compatibility. Integration testing is a level of the software testing process where individual units are combined and tested as a group. The purpose of integration testing level of testing is to expose faults in the interaction between integrated units.
3. **System Testing:** It is a level of the software testing process where a complete, integrated system/software is tested. The purpose of system testing is to evaluate the system's compliance with the specified requirements.
4. **Acceptance Testing:** It is a level of the software testing process where a system is tested for acceptability. The purpose of acceptance testing is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

3.1 A STRATEGIC APPROACH TO SOFTWARE TESTING

- Testing is the process of exercising or evaluating a system (or its component) by manual or automated means to verify that it satisfies specified requirements.
- The Institute of Electrical and Electronics Engineers (IEEE) defines testing is the process, as a sequence of steps performed for a given purpose.
- Software process is a set of activities, methods, practices, and transformations that people use to develop and maintain software products.
- Testing is a set of activities that can be planned in advance and conducted systematically.
- For this reason a template for software testing – a set of steps into which we can place specific test case design techniques and testing methods – should be defined for the software process.
- A number of software testing strategies have been proposed in the literature. All these strategies provides the software developer/engineer with a template for testing and all have the following generic characteristics:
 - Testing process begins at the component level and works "outward" towards the integration of the entire computer-based system.
 - Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
 - To perform effective testing, a software team should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
 - Testing is conducted by the developer of the software and (for large projects) an independent test group.
- A strategy for software testing may also be viewed in the context of the spiral as shown in Fig. 3.2.

- Unit testing process begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code.
- Unit testing exercises a specific control path to detect errors in each software unit or component. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture.
- Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed.
- Validation testing strategy assures that software validation criteria fulfills all functional behavioral and performance requirements which are established during the requirement specification.
- Finally, we arrive at system testing, where the software and other system elements are tested as a whole. System testing strategy verifies that all system elements functions properly with best performance.
- To test computer software product or application, we spiral out along streamlines that broaden the scope of testing with each turn.

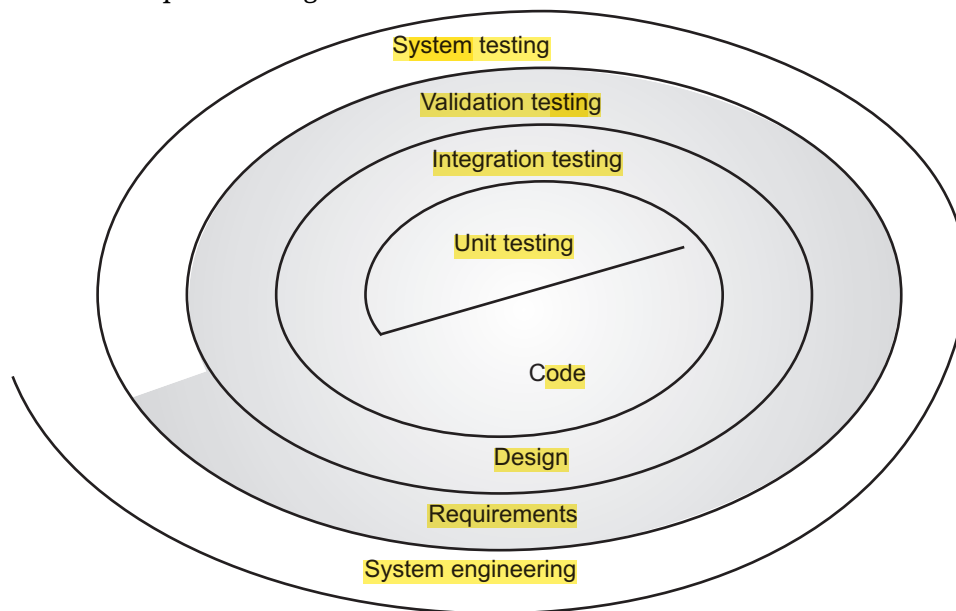


Fig. 3.2: Software Testing Strategy

- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well high-level tests that validate major system functions against customer requirements.
- A software testing strategy must provide guidance for the practitioner and a set of milestones for the manager.

- Because the steps of the software test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible.
- A strategy for software testing is developed by the software project manager, software engineer and testing specialist.

Verification and Validation (V&V):

- Software V&V is a disciplined approach to assessing software products throughout the product life cycle.
 - **Verification:** It refers to the set of activities that ensure that software correctly implements a specific function.
 - **Verification:** It refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- B. W. Boehm states V&V in following way:
 - **Verification:** "Are we building the product right?"
 - **Validation:** "Are we building the right product?"
- Verification confirms that work products properly reflect the requirements specified for them. Verification ensuring that we are doing things right through a process.
- Validation confirms that the software product, as provided, will fulfill its intended use. Validation ensures that we are building right things through requirements.

A Software Testing Strategy for Object-Oriented Architectures:

- The testing of Object-Oriented (OO) systems presents a different set of challenges for the software engineer/developer.
- The definition of testing must be broadened to include error/bug/defect/fault/failure discovery techniques (e.g., formal technical reviews) that are applied to analysis and design models.
- The completeness and consistency of object-oriented representations must be assessed as they are built. Unit testing in software testing loses some of its meaning and integration strategy change significantly.
- Both i.e., testing strategies and testing tactics must account for the unique characteristics of object-oriented software.
- The overall testing strategy for object-oriented software is identical in philosophy to the one applied for conventional architectures, but differs in approach. We begin with "testing in the small" and work outward toward "testing in the large".
- However, our focus when "testing in the small" changes from an individual module to a class that encompasses attributes and operations and implies communication and collaboration.
- As classes are integrated into an object-oriented architecture, a series of regression tests are run to uncover errors due to communication and collaboration between classes (components) and side effects caused by the addition of new classes (components).
- Finally, the system as a whole is tested to ensure that errors in requirements are uncovered.

Organizing for Software Testing:

- For every software project, there is an inherent conflict of interest that occurs as testing begins. The people who have built or develop the software are now asked to test the software.
- This seems harmless in itself; after all, who knows the computer program better than its developers? Unfortunately, these same developers have a vested interest in demonstrating that the program is error/bug free, i.e. works according to customer requirements and that it will be completed on schedule and within budget. Each of these interests mitigate against through testing.
- The software developer is always responsible for testing the individual units (components/elements) of the program, ensuring that each performs the function or exhibits the behavior for which it was designed.
- The basic role of an ITG (Independent Test Group) is to remove the inherent problems associated with letting the builder test the thing that has been built.
- Independent testing removes the conflict of interest that may otherwise be present. After all, ITG personnel are paid to find/detect errors.
- However, the software engineer/developer does not turn the program over to ITC and walk away.
- The software developer and the ITG work closely throughout a software project to ensure that thorough tests will be conducted.
- While testing is conducted, the developer must be available to correct errors that are uncovered.

3.2 TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

- There are a number of strategies that can be used to test software product or system. At one extreme, a software testing team could wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors.
- This approach, although appealing, simply does not work and will result in buggy software that disappoints the customer and end-user.
- At the other extreme, a software engineer could conduct tests on a daily basis, whenever any part of the system is constructed.
- This approach, although less appealing to many, can be very effective. Unfortunately, most software developers hesitate to use it. What to do?
- A testing strategy that is chosen by most software teams falls between the following two extremes:
 1. takes an incremental view of testing, beginning with the testing of individual program units, moving to tests designed to facilitate the integration of the units.
 2. culminating with tests that exercise the constructed system. Each of these classes of tests is described in the sections that follow.

3.2.1 Unit Testing

- Unit testing is a type of software testing in which the smallest testable units/components of the software are tested. The testable module can be a function or procedure.
- The purpose of unit testing is to validate that each such module of the software performs as desired.
- Unit testing is done to validate the code developed by the developer meets the requirements and performs in the expected manner. It is the testing of individual software component.
- Testing that occurs at the lowest level is called unit testing or module testing. Unit testing is performed to test the individual units of software. Unit is the smallest part of software system which is testable.
- Unit testing may be including code file, classes, methods which can be tested individually for correctness.
- The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect.
- In unit testing each unit is tested separately before integrating them into modules to test the interfaces between modules.
- Unit testing is a very important type of software testing that tests individual software components. Unit testing of software applications are performed during the development (encoding) of an application.
- The goal of unit testing is to isolate part of the code and verify its correctness. In procedural programming, a unit can also be a single function or procedure.
- Unit tests are usually done by the developer. Fig. 3.3 shows concept unit testing.

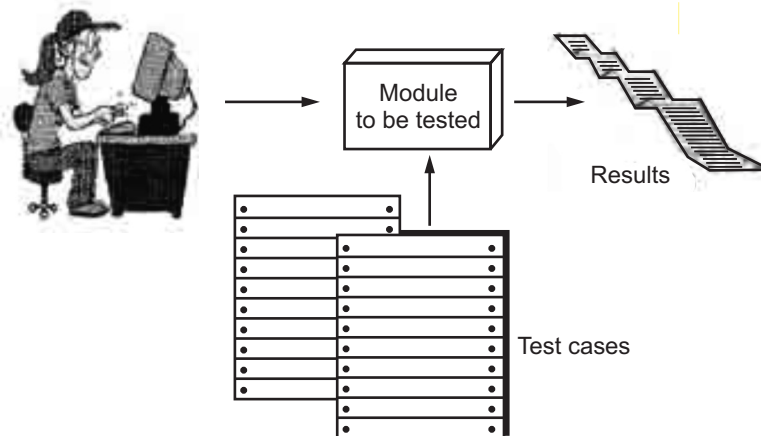


Fig. 3.3: Unit Testing

- Unit testing performs the following functions:
 1. Unit testing tests all control paths to uncover maximum errors that occur during the execution of conditions present in the unit being tested.
 2. Unit testing ensures that all statements in the unit have been executed at least once.
 3. It tests data structures that represent relationships among individual data elements.
 4. Unit testing checks the range of inputs given to units i.e. a maximum and minimum value.

Unit Tests Considerations:

- The tests that occur as part of unit tests are shown schematically in Fig. 3.4.
 - In unit testing, the module interface is tested to ensure that information properly flows into and out of the program unit under test.
 - In unit testing, the local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
 - In unit testing, all independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once.
 - In unit testing, the boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. And finally, all error handling paths are tested.
 - In unit testing, the tests of data flow across a module interface are required before any other test is initiated. If data in testing process do not enter and exit properly, all other tests are moot.
 - In addition, local data structures in unit test consideration should be exercised and the local impact on global data should be ascertained (if possible) during unit testing.
 - Selective testing of execution paths in unit test consideration is an essential task during the unit test.
 - In unit test consideration, the test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons or improper control flow.
 - Among the more common errors in computation are given below:
 1. Incorrect initialization.
 2. Incorrect symbolic representation of an expression.
 3. Precision inaccuracy.
 4. Mixed mode operations.
 5. Incorrect arithmetic precedence.

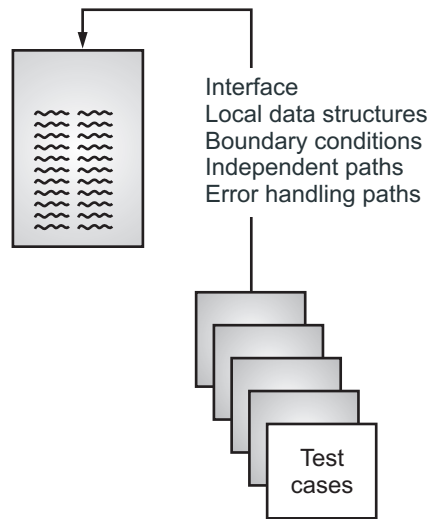


Fig. 3.4: Unit Test

- Test cases should uncover errors such as given below:
 1. Improperly modified loop variables.
 2. Incorrect comparison of different data types.
 3. Failure to exit when divergent iteration is encountered.
 4. Incorrect logical operators or precedence.
 5. Expectation of equality when precision error makes equality unlikely.
 6. Incorrect comparison different of variables.
 7. Nonexistent to **improper** loop termination.
- One of the most important unit testing tasks is Boundary testing. Software often fails at its boundaries i.e., errors often occur when the n^{th} element of an n -dimensional array is processed, when the i^{th} repetition of a loop with i passes is invoked, when the maximum or minimum allowable value is encountered.
- Test cases that exercise data structure, control flow and data values just below, at and just above maxima and minima are very likely to uncover errors.

Unit Test Procedures:

- In software testing, unit testing is normally considered as an adjunct to the program coding step.
- The design of unit tests can be performed before coding begins (a preferred agile approach) or **after source code has been generated**.
- A review of design information provides guidance for establishing test cases that are likely to uncover errors. Each test case should be coupled with a set of expected results.
- Because a component or **unit is not a stand-alone program, driver and/or stub software must be developed for each unit test.**

- The unit test environment is shown in Fig. 3.5, which shows stub and driver concept in unit testing. The unit testing requires drivers and stubs to be written.
- Drivers and stubs are special-purpose arrangements in unit testing, generally code, required to test units individually which can act as an input to the unit/module and can take output from unit/module.
- In number of applications a driver is nothing more than a main program that accepts test case data, passes such data to the component (to be tested) and prints relevant results.
- Stubs in unit test serve to replace modules that are subordinate to (called by) the component to be tested.
- A stub (or dummy subprogram) uses the subordinate module's interface, may do minimal data manipulation, provides verification of entry and returns control to the module undergoing testing.
- The driver in unit testing simulates a calling unit and the stub simulates a called unit. A component is not a stand-alone program; driver and/or stub software must often be developed for each unit test.
- Both i.e., stub and driver must be written (formal design is not commonly applied) but that is not delivered with the final software product.

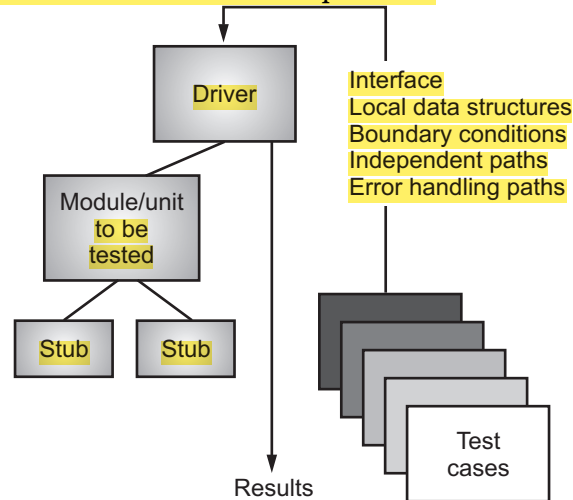


Fig. 3.5: Concept of Stub and Driver

Difference between Driver and Stub:

Sr. No.	Driver	Stub
1.	Drivers mainly created for integration testing such as bottom-up approach.	Stubs are mainly created for integration testing such as top-down approach.

contd. ...

2.	Very simple to develop.	It is also very simple to develop.
3.	A driver is basically a program that accepts test case data and passes that data to the module that is being tested.	Stubs are also programs that are used to replace modules that are subordinate to the module to be tested.
4.	A driver is a piece of software program which calls the functions in the unit under test.	A stub is a small program routine that substitutes for a longer program, possibly to be loaded later or that is located remotely.
5.	Stub is the piece of code emulating the called function.	Driver is piece of code emulating a calling function.

Advantages of Unit Testing:

1. Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach.
2. Unit tests find problems early in the software development cycle.
3. Unit testing allows the software programmer/developer to re-factor code at a later date, and make sure the module still works correctly.

Disadvantages of Unit Testing:

1. Unit testing requires initial time to develop them.
2. Unit testing will not catch every error in the program, since it cannot evaluate every execution path in any but the most trivial programs.
3. Unit testing only tests the functionality of the units themselves. Unit testing will not catch integration errors or broader system-level errors.
4. Unit testing can only show the presence or absence of particular errors. It cannot prove a complete absence of errors.
5. Writing the unit tests in unit testing is the difficulty of setting up realistic and useful tests.

3.2.2 Integration Testing

- Integration testing is a software testing methodology, during which the units of the software or the unit modules of the software integrated are validated.
- Integration testing is a level of software testing where individual units are combined and tested as a group.
- The purpose of this level of testing is to expose faults in the interaction between integrated units.
- A typical software package project consists of multiple software modules, coded by different programmers. Integration testing focuses on checking interactions amongst these modules.
- Integration testing may start at module level where different units and components come together to form a module and so up to system level.

- Integration testing is considered as structural testing. Fig. 3.6 shows a system levels schematically with integration testing.
- Integration testing mainly focuses on I/O protocols, parameters passing between different unit's modules and/or system etc.

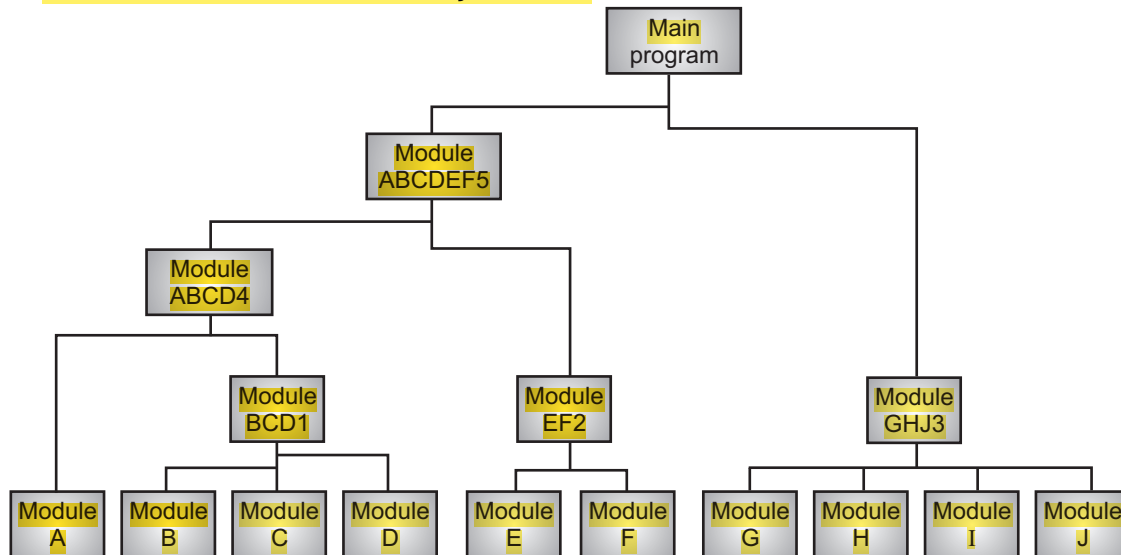


Fig. 3.6: Integration Testing with System Levels

Working of Integration Testing:

- Integration testing is a systematic software testing technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- The goal of integration testing is to take unit tested components and build a program structure that has been dictated by design.
- Once, unit testing is complete, integration testing begins. In integration testing the units/components validated during unit testing are combined to form a subsystem.
- The integration testing is aimed at ensuring that all the modules work properly as per the user requirements or expectations when they are put together i.e. integrated.
- The objective of integration testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.
- Fig. 3.7 shows working of integration testing.
- There is often a tendency to attempt non-incremental integration i.e., to construct the program using a Big-Bang integration testing strategy.
- The software testing approach, in which all the components/units of the systems are integrated simultaneously at once is called as Big-Bang integration.

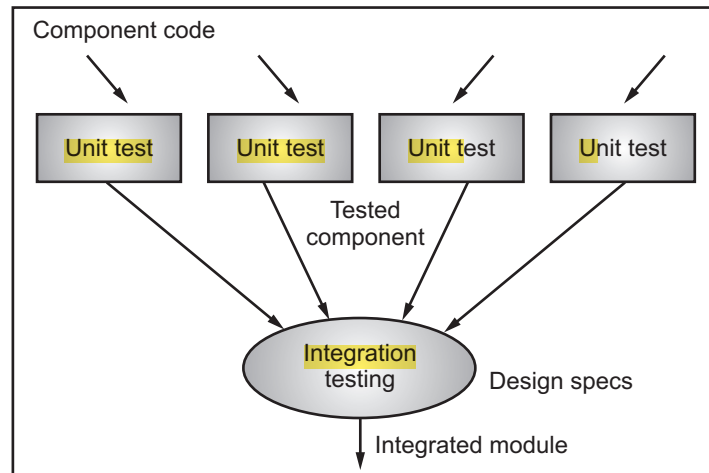


Fig. 3.7: Concept of Integration Testing

- In Big-Bang integration testing strategy, the entire program is tested as a whole. In Big-Bang integration testing all units are linked at once, resulting in a complete system.
- When Big-Bang integration testing strategy is adopted, it is difficult to isolate any errors/bugs found, because attention is not paid to verifying the interfaces across individual units.

Advantages of Integration Testing:

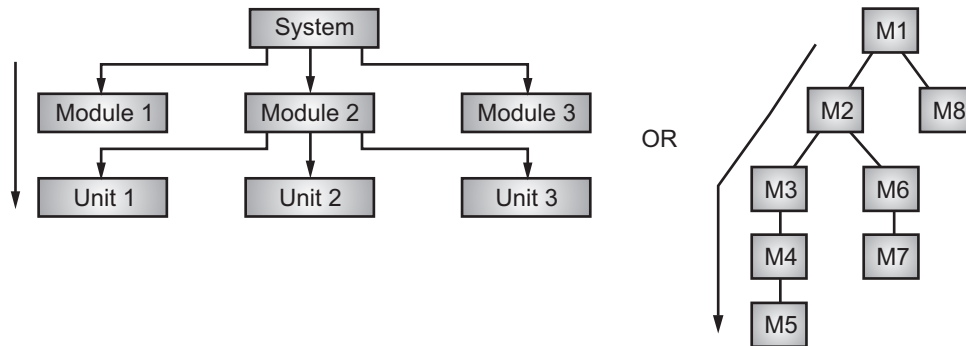
1. The integration testing is efficient and convenient for small systems.
2. In integration testing, it is easy to fix the error when compared to the system testing.
3. In integration testing, the interfaces should be checked thoroughly if any error messages are shown.
4. The integration testing is a systematic technique of integrating system software is practiced.
5. In integration testing, interfacing errors between different modules of a system are un-rooted.
6. In integration testing, the functionality of the system is enhanced before the product is developed. This can be considered as the prototype.

Disadvantages of Integration Testing:

1. The time used for developing, integrating, testing of all the modules is significant. This leads to delay in testing team execution time.
2. In integration testing, numerous modules are to be interfaced, which needs to be tested in a system. This could lead to missing of an interface.

3.2.2.1 Top-Down Integration

- In top-down integration the top level of the software application or system is tested first and then it goes downward till it reaches the final component of the system.
- In top-down integration testing, the software is developed and tested by integrating the individual modules, moving downwards in the control hierarchy.
- In top-down integration testing, initially only one module, known as the main control module is tested.
- After this, all the modules called by it are combined with it and tested. This process continues till all the modules in the software are integrated and tested.
- The top-down integration testing is also referred as incremental integration testing technique which begins by testing the top-level module and progressively adds in lower level module one-by-one.
- Lower level modules in top-down integration testing are normally simulated by stubs which mimic functionality of lower level modules. As you add lower level code, stubs will replace with actual components.
- The integration testing technique, in which 'Stubs' are used as the called modules at the lower-level in order to simulate the behavior of the same, is known as the top-down integration testing.
- Integration begins with the top modules. To perform top-down integration testing, the following steps are used:
 - Step 1:** The main control module is used as a test driver and all the modules subordinate to the main control module are replaced with stubs.
 - Step 2:** The subordinate stubs are then replaced with actual modules, one stub at a time.
 - Step 3:** As each new module is integrated, tests are conducted.
 - Step 4:** After each set of tests is complete to replace another stub with actual module.
 - Step 5:** In order to ensure no new errors have been introduced, regression testing may be performed.
- In this approach, top-level components are the user interfaces which are created first, to elicit user requirements or creation of prototype.
- Approaches like proto-typing, formal proof of concept, test driver development etc. uses top-down approach for testing.
- Top-down integration can be performed and tested in breadth first or depth first manner. The top-down integration and testing can be shown in Fig. 3.8.
- In depth first integration all modules on a construct path are integrated first and breadth first integration all modules directly subordinate at each level are integrated together.

**Fig. 3.8: Top-Down Integration****Advantages of Top-Down Integration Testing:**

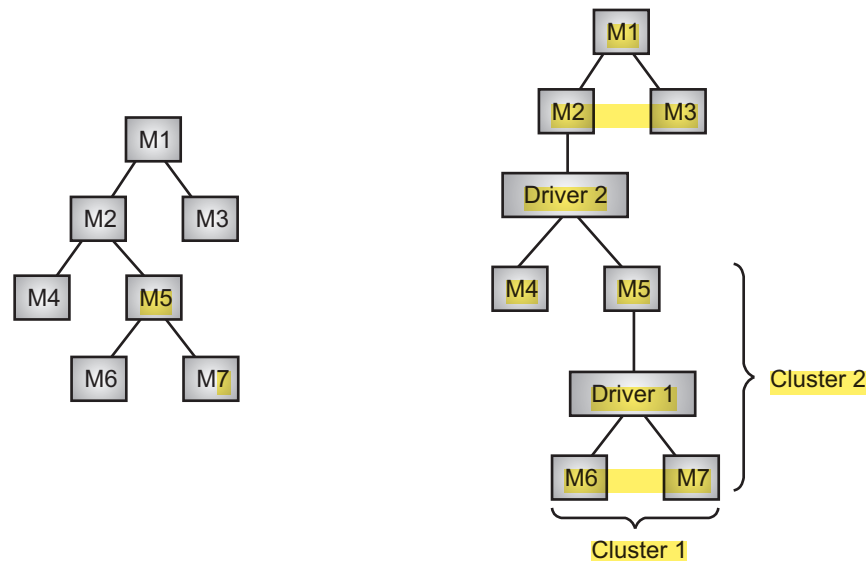
1. In top-down integration testing, feasibility of an entire program can be determined easily and simply at a very early stage as the top most layers, generally User Interface (UI) is made first.
2. Number of times top-down integration testing does not need drivers as the top layers are available first which can work as drivers for the layers below.
3. The top-down integration testing can detect major flaws in system designing by taking inputs from user. Prototyping is used extensively in agile application development where user requirement can be clarified by preparing a model.
4. The top-down integration testing provides early working module of program and so design defects can be found and corrected early.

Disadvantages of Top-Down Integration Testing:

1. In top-down integration testing units and modules are rarely tested alone before their integration. There may be few problems in individual units/modules which may get compensated in testing.
2. The compensating defects cannot be found in top-down integration.
3. In top-down integration testing the absence of lower-level routines, many times it may become difficult to exercise the top level routines in the desired manner since the lower-level routines perform several low-level functions such as Input/Output functions.
4. Top-down integration testing can create a false belief that software can be coded and tested before design is finished.
5. In top-down integration testing, stubs are to be written and tested before they can be used in integrated testing. Stubs must be as simple as possible and must not introduce any defect in the software. Large number of stubs may be required which are to be thrown at the end.
6. In top-down integration testing, it is difficult to have other people or third parties to perform this testing; mostly developers will have to spend time on this.

3.2.2.2 Bottom-Up Integration

- In bottom-up integration testing, individual modules are integrated starting from the bottom and then moving upwards in the hierarchy.
 - The bottom-up integration testing combines and tests the modules present at the lower levels proceeding towards the modules present at higher levels of the control hierarchy.
 - Bottom-up integration testing approach focuses on testing the bottom part/individual units and modules and then goes upward by integrating testing and working units and modules for system testing and inter-system testing.
 - In bottom-up integration each sub-system is tested separately and then the full system is tested. A sub-system must consist of many modules which communicate among each other through well-defined interfaces.
 - The integration testing technique where integration of modules begins from the lowest level and test drivers are used to drive and pass the data to lower modules is known as the bottom-up integration.
 - In this approach control and data interfaces are tested. Bottom-up integration testing starts at the atomic modules level. Atomic modules are the lowest levels in the program structure.
 - Bottom-up approach generally used for object oriented design and general purpose utility routines.
 - A bottom-up integration is implemented with the following steps:
 - Step 1:** Low level modules are combined into clusters that perform a specific software sub-function. These clusters are sometimes called builds.
 - Step 2:** A driver (a control program for testing) is written to coordinate test case input and output.
 - Step 3:** The build/cluster is tested.
 - Step 4:** Drivers are removed and clusters are combined moving upward in the program structure.
 - Fig. 3.9 shows how the bottom-up integration is done.
 - Whenever, a new module is added to as a part of bottom-up integration testing, the program structure changes. There may be a new data flow paths, some new Input/Output (I/O) or some new control logic.
 - These **new** changes may cause problems with functions in the tested modules, which were working fine previously. To detect these errors regression testing is done.
-



(a) Program Module

(b) Bottom-up Integration Applied to (a)

Fig. 3.9

Advantages of Bottom-up Integration Approach:

1. In this approach each components and unit is tested first for its correctness. If it found to be working correctly then only it goes for further integration.
2. Incremental integration testing is useful where individual components can be tested of integration.
3. This approach makes a system more robust since individual units are tested and confirmed as working.

Disadvantages of Bottom-up Integration Approach:

1. In this approach, top-level components are most important but tested last, where the pressure of delivery may cause problem of not completing testing.
2. There can be major problems during integration of interface testing or system level functioning may be a problem.
3. In bottom-up integration, objects are combined one at a time, which may need longer time and result into slow testing. Time required for complete testing may be very long and thus, it may disrupt entire delivery schedule.
4. Designing and writing stubs and drivers for testing is waste of work as they do not form part of final system.
5. In this approach, stub and drivers are to be written and tested before using them in integration testing. One needs to maintain review and test records of stubs and driver to ensure that they do not introduce any defect.
6. For initial phases, in this approach one may need both stubs and drivers. As one goes on integrating units, original stubs may be used while large number of new drivers may be required.

3.2.3 System Testing

- System testing is the process of attempting to demonstrate that a software program/system does not meet its original requirement/goals/objectives as stated in the requirement specifications.
- System testing is the level of testing where integrated and completed software is validated to verify the systems functionality and reliability in compliance to the specified requirements.
- It ensures that all the integrated modules of the system software are functioning accurately, including the external peripherals of the software.
- System testing represents the final testing done on a software system before it is delivered to the customer.
- In system testing, the system is tested against functional/non-functional requirements such as accuracy, reliability, and speed defined by the user/customer in software requirement specification.
- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.
- System testing is a method to monitor and evaluate the behavior of the complete and fully integrated software product or system. System testing involves the external view of how the software works from the user's perspective.
- The goal of system testing is to find or detect defects in features of the software system compared to the way it has been defined in the software system requirements and the test object is the fully integrated system.
- The system testing ensures that all the integrated modules of the software system are functioning accurately or not.
- IEEE defines system testing as, 'a testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement.'
- Fig. 3.10 shows system testing concept.

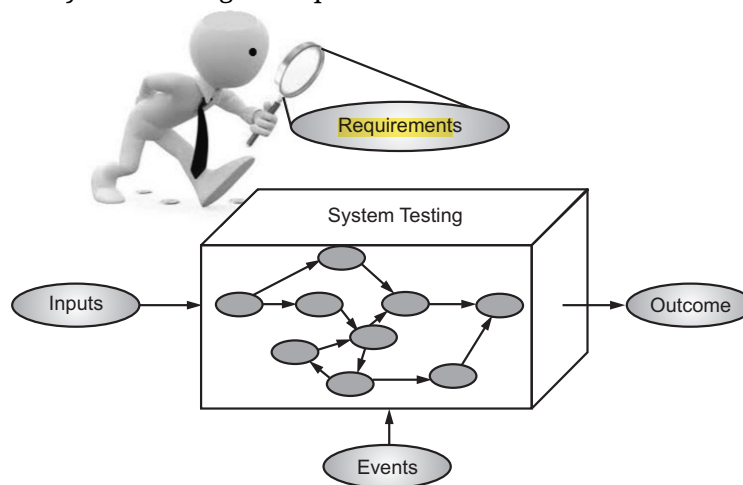


Fig. 3.10: Concept of System Testing

- There are several types of system testing. The types of system testing that are commonly used in a large software development company are listed below:
 1. **Regression Testing:** The regression testing is a type of software testing that confirms that a recent program or code change did not affect existing functionality. It also ensures that over time no old errors occur by adding new software modules.
 2. **Usability Testing:** Usability testing is a non-functional testing methodology that checks how easy the system can be used by end users. The focus is on application usability, control flexibility, and the ability of the system to achieve its goals.
 3. **Load Testing:** Load testing is a type of performance testing that determines the performance of a system under real load conditions. It is important to know that a software solution provides the performance required under real load.
 4. **Hardware/Software Testing:** Hardware/software testing is performed when the tester focuses on the hardware-software interaction during the system test.
 5. **Migration Testing:** Migration testing ensures that software can be easily transferred from legacy system infrastructures to current system infrastructures.
 6. **Function Testing:** Function testing, also known as the function completeness testing, tests all functions that the System should provide and identifies possible missing functions. Testers may compile a list of additional features that a product needs to improve the system during the functional testing.
 7. **Recovery Testing:** Recovery tests are performed to demonstrate that a software solution is reliable, trustworthy and can successfully resolve possible crashes.

Advantages of System Testing:

1. System tests help clearly specify how the application should behave.
2. System tests can be run automatically (for example, each night) so that testing is done as the application is being developed.
3. System tests help us to test that the application is working correctly from the point of view of a user.
4. System tests help us to test that changes made to one part of the application haven't created a bug somewhere else.
5. It helps in identifying and fixing of bugs before the code is pushed to production.
6. Root cause analysis is performed resulting to increased efficiency.

Disadvantages of System Testing:

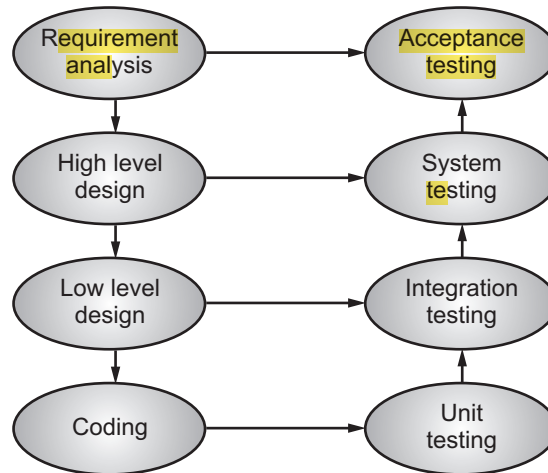
1. Process could involve redundant testing.
2. Possibility of missing out on the logical errors of the software or system.

3.2.4 Acceptance Testing

- Acceptance testing, a software testing technique performed to determine whether or not the software system has met the requirement specifications.
- User acceptance testing performed by the customer to certify the system in relation to the agreed requirements.
- The main purpose of acceptance testing is to evaluate the system's compliance with the organizational requirements and verify if it has met the required criteria for delivery to end users.
- Acceptance testing is the formal testing process conducted to determine whether a software or system satisfies its acceptance criteria.
- IEEE defines acceptance testing as, 'a formal testing with respect to user needs, requirements and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.' **OR**
- Acceptance testing is, "a level of the software testing process where a system is tested for acceptability."
- Acceptance testing is designed to:
 1. Determine whether the software is fit for the user/customer.
 2. Making confident about the software product.
 3. Determine whether the software product satisfies its acceptance criteria.
- There are following various forms of acceptance testing:
 1. **User Acceptance Testing (UAT):** UAT consists of a process of verifying that a solution works for the user. UAT is also known as beta testing, application testing or end user testing. It is conducted by the customer to ensure that system satisfies the contractual acceptance criteria before being signed-off as meeting user needs.
 2. **Business Acceptance Testing (BAT):** BAT is undertaken within the development organization of the supplier to ensure that the system will eventually pass the user acceptance testing.
 3. **Alpha Testing:** The alpha test is the first stage of testing and will be performed amongst the teams. Alpha testing takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers.
 4. **Beta Testing:** The beta test is performed after Alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta testing takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers.

Acceptance Testing in SDLC:

- Fig. 3.11 shows the placement of acceptance testing in the software development life cycle.
- The acceptance test cases in acceptance testing are executed against the test data or using an acceptance test script and then the results are compared with the expected ones.

**Fig. 3.11****Acceptance Criteria:**

- Microsoft press describes acceptance criteria as, **the conditions that a software product must satisfy to be accepted by a user, customer or other stakeholder.**
- Requirements must pass acceptance criteria at each phase, so that they can be taken for creating designs which may be as per entry criteria for design (high level design and low level design).
- Designs must fulfill acceptance criteria so that it can be taken for coding and coding must satisfy acceptance criteria so that system testing can be started and so forth.
- Google defines acceptance criteria as, "pre-established standards or requirements a product or project must meet". **OR**
- Acceptance criteria is defined as, **"the list of requirements that must be satisfied prior to the customer accepting delivery of the product"**.

Advantages of Acceptance Testing:

1. This testing gives user an opportunity to ensure that software meets user requirements, before actually accepting it from the developer.
2. It is easier and simpler to run an acceptance test compared to other types of test.
3. It enables both users and software developers to identify and resolve problems in software.

4. This testing determines the readiness (state of being ready to operate) of software to perform operations.
5. It decreases the possibility of software failure to a large extent.

Disadvantages of Acceptance Testing:

1. The users may provide feedback without having proper knowledge of the software.
2. Users are not professional testers, they may not be able to either discover all software failures or accurately describe some failures.

3.2.5 Performance Testing

- Performance testing is used to check the software system is working under the heavy load. Performance testing is a non-functional testing technique.
- Performance testing is performed to determine the system parameters in terms of responsiveness and stability under various workloads. It measures the quality attributes of the system such as reliability, scalability and resource usage.
- Performance testing is a form of software testing that focuses on how a system running the system performs under a particular load.
- Performance testing is used to test several factors that play an important role in improving the overall performance of the computer system such as **Stability** (how long a system is able to prevent itself from failure. Performance testing verifies whether the system remains stable under expected and unexpected loads), **Speed** (how quickly a system is able to respond to its users. Performance testing verifies whether the response is quick enough) and **Scalability** (the extent to which a system is able to handle the load given to it. Performance testing verifies whether the system is able to handle the load expected by users).
- Performance testing is a type of non-functional testing technique in which the performance of an application is evaluated under simulated expected or higher than expected workload.
- Various types of performance testing are explained below:
 1. **Load Testing:**
 - Load testing is usually conducted to understand the behavior of the system under a specific expected load.
 - This load in the system can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration.
 2. **Stress Testing:**
 - This testing is normally used to understand the upper limits of capacity within the system.
 - The goal or objective of stress testing is to measure the software stability. At what point does software fail, and how does the software recover from failure?

- The **stress testing** is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.
- The **stress testing also measure system performance outside of the parameters of normal working conditions**. The software is given more users or transactions that can be handled.

3. Spike Testing:

- This type of performance testing is done by suddenly **increasing or decreasing the load generated by a very large number** of users and observing the behavior of the system.
- The goal of spike **testing is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load**.
- Spike testing evaluates **software** performance when workloads are substantially increased quickly and **repeatedly**. The workload is beyond normal expectations for short amounts of time.

4. Configuration Testing:

- The **software tests are created to determine the effects of configuration changes to the system's components on the system's performance and behavior**.

5. Soak Testing:

- Soak testing is usually done to determine if the **software system can sustain the continuous expected load**.
- Soak testing increases the **number of concurrent users and monitors the behavior of the system over a more extended period**.
- The objective of the **soak testing is to observe if intense and sustained activity over time shows a potential drop in performance levels, making excessive demands on the resources of the system**.

Performance Testing Process:

- Performance testing can help demonstrate that the software system meets certain pre-defined performance criteria. It can also help identify parts of your software system which degrade its performance.
- Fig. 3.12 shows process of performance testing.

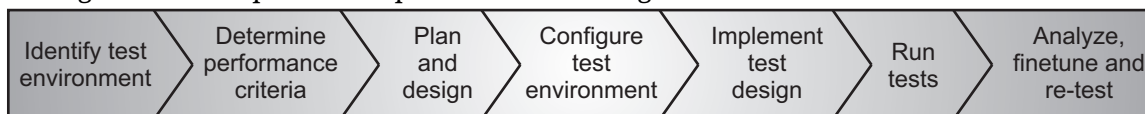


Fig. 3.12: Steps in Performance Testing

- Fig. 3.12 shows following steps for performance testing:
 1. **Identify Testing Environment:** Know the physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before begin the testing process.

2. **Identify the Performance Acceptance Criteria:** This step includes goals and constraints for throughput, response times and resource allocation in testing process.
3. **Plan and Design Performance Tests:** This step determines how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
4. **Configuring the Test Environment:** This step prepares the testing environment before execution. Also, arrange tools and other resources.
5. **Implement Test Design:** This step creates the performance tests according to your test design.
6. **Run the Tests:** This step executes and monitors the tests.
7. **Analyze Tune and Retest:** This step consolidate, analyze and share test results.

Advantages of Performance Testing:

1. It assess whether a component of system complies with specified performance requirements.
2. It compares different systems to determine which system performs better.
3. Performance testing also checks system characteristics such as its reliability.

3.2.6 Regression Testing

- Regression testing is designed to test the application after the introduction of changes. Regression testing consists of re-executing those tests that are impacted by the software code changes.
- The purpose of regression testing is to ensure that changes made to software, such as adding new features or modifying existing features, have not adversely affected features of the software that should not change.
- Regression testing is the selective retesting of a software system to verify that modification or improvements has not caused unintended effects and that the system still complies with its specified requirements.
- Regression testing, 're-tests' the software system or part of it to ensure that the components, features and functions, which were previously working properly, do not fail as a result of the error correction process and integration of modules.
- The regression test suite (the subset of tests to be executed) contains three different classes of test cases:
 1. Tests that check the modified software modules.
 2. Tests that check all functions of the software.
 3. Tests that check the functions that can be affected due to changes.

- Regression testing is a form verification test because it will help us determine if the software works as expected even after undergoing critical modifications to its architecture.

Benefits of Regression Testing:

1. Regression testing ensures the software is working very well even after introducing many changes and modifications.
2. The regression testing ensures that the unchanged parts of software system work properly according to specification.
3. This testing ensures that all errors that have occurred in software system due to modifications are corrected and do not further affect the working of the software.

Disadvantages of Regression Testing:

1. The regression testing requires knowledge about the software system and how it affects by the existing functionalities.
2. Regression testing is time-consuming and expensive activity.

3.2.7 Load Testing

- Load testing is used to examine the behavior of a software system when subject to both normal and extreme expected load conditions.
- When a software system is tested with a load that causes it to allocate its resources in maximum counts it is called as load testing.
- Load testing is a non-functional testing technique which is conducted to understand the behavior of the application under a specific expected load.
- Load testing helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation.
- Example, if the number of users is increased then how much CPU, memory will be consumed, what is the network and bandwidth response time.
- Load testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system.
- Load testing involves simulating real-life user load for the target application and helps us to determine how the application behaves when multiple users hits it simultaneously.
- The goal of load testing is to define the maximum amount of work a system can handle without significant performance degradation.

Process of Load Testing:

- Fig. 3.13 shows the process of load testing.

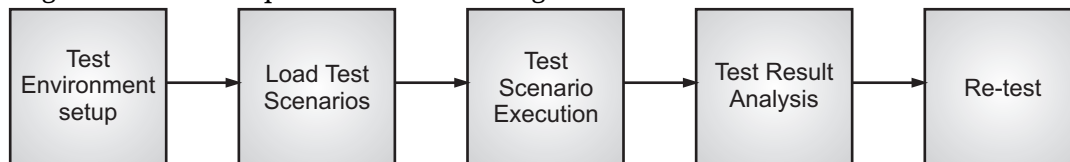


Fig. 3.13

- The load testing process will be completed in the following steps:
 - Step 1: Test Environment Setup:** In this step, we will set up the test environment to execute the load testing to ensure the testing can be done appropriately.
 - Step 2: Load the Test Scenario or Specify the Performance Criteria:** In this step, we will define the performance criteria, which contain the response time, reasonable limits on throughput, and the load test transaction. Then we create the load test scenarios, which ensure the success criteria are finalized.
 - Step 3: Execution of Test Scenarios:** In this step, we will execute the particular test scenarios.
 - Step 4: Analysis of the Test Result:** After executing the load test scenarios, we will analyze the test results.
 - Step 5: Re-test:** This is the last step of the load testing, depends on the test result because if the test fails, we have to perform the same process repeatedly until the test result is passed and all the issues and bottlenecks are fixed.

Examples of Load Testing:

1. Subjecting a server to a large amount of traffic.
2. Downloading a series of large files from the Internet.
3. Running multiple applications on a computer or server simultaneously.

Advantages of Load Testing:

1. Load testing will also prevent software failures, because it can predict how the system will react when it is given large loads of files and large amount of tasks.
2. Load testing allows having an idea of the scalability and the performance of the software.
3. Load testing will expose the bugs such as undetected memory overflow and memory management bugs in the system.

3.2.8 Stress Testing

- Stress testing is a software testing technique designed to determine the behavior of the software under abnormal situations.
- Stress testing executes a software or system in a manner that demands resources in abnormal quantity, frequency or volume.
- For example,
 - Special tests may be designed that generate ten interrupts per second, when one or two is the average rate
 - Input data rates may be increased by an order of magnitude to determine how input functions will respond
 - Test cases that require maximum memory or other resources are executed
 - Test cases that may cause memory management problems are designed
 - Test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to overwhelm the program.

- In stress testing, the test cases are designed to execute the system in such a way that abnormal conditions arise.
- IEEE defines stress testing as, 'testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.'
- Stress testing is running the software under less-than ideal conditions-low memory, low disk space, slow CPU's, slow modems and so on.
- Stress testing is a form of software testing that is used to determine the stability of a given system and it put greater emphasis on robustness, availability and error handling under a heavy load, rather than on what would be considered correct behavior under normal circumstances.
- The goals of stress testing to ensure the software do not crash in conditions of insufficient computational resources (such as memory or disk space).
- The purpose behind stress testing is to determine the failure of system and to monitor how the system recovers back gracefully.
- The challenge here is to set up a controlled environment before launching the test so that we could precisely capture the behavior of system repeatedly, under the most unpredictable scenarios.
- Stress testing is testing technique used to check the accessibility and robustness of software beyond usual functional limits.
- Stress testing mainly considers for critical software but it can also be used for all types of software applications.

Process of Stress Testing:

- Fig. 3.14 shows the process for stress testing.

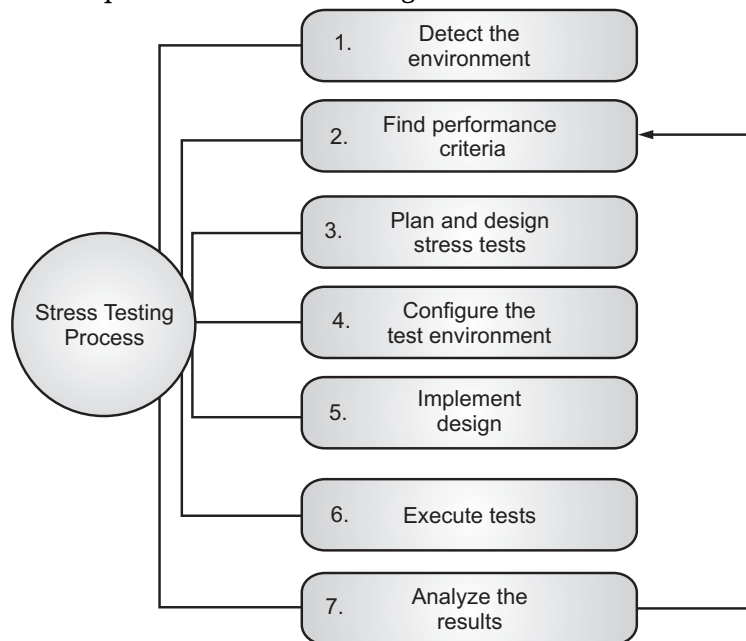


Fig. 3.14

- The stress testing process will be completed into the following steps:
 - Step 1: Detect the Testing Environment:** In this step, we will identify the network, software, and hardware configurations and tools available to achieve the stress test.
 - Step 2: Find Performance Acceptance Criteria:** In this step, we will find the performance acceptance criteria, which help us categorize the metrics used to test the application's performance under stress.
 - Step 3: Plan and Design Stress Tests:** In this step, we will plan and design a stress test plan, identify test scenarios etc.
 - Step 4: Configure the Test Environment:** In this step, we will move to next step where we create the test environment, tools and resources essential to perform each approach as features and components become available for test.
 - Step 5: Implement Test Design:** In this step, we will develop the stress tests resulting the test design best performs.
 - Step 6: Execute Tests:** In this step, we will execute the particular test, observe and confirm the tests along with test data and output collection.
 - Step 7: Analyze the Results:** In this step, we will analyze the outcomes, combine and share the respective teams' output data.

Examples of Stress Testing:

1. E-commerce website/ application.
2. News website at the time of some major/viral event.
3. Education Board's result website.

Advantages of Stress Testing:

1. Stress testing technique indicates the expected behavior of a system when it reaches the extreme level of its capacity.
2. This testing technique executes a system till it fails. This enables the testers to determine the difference between the expected operating conditions and the failure conditions.
3. Stress testing technique determines the part of a system that leads to errors.
4. Stress testing technique determines the amount of load that causes a system to fail.
5. Stress testing technique evaluates a system at or beyond its specified limits of performance.

Disadvantages of Stress Testing:

1. Using stress testing requires additional resources, which makes this testing costlier.
2. To writing the stress testing script or test cases, the tester should have enough scripting knowledge of the language supported by the particular tool.

3.2.9 Security Testing

- Any computer-based system that manages sensitive information and causes actions that can incorrectly harm or benefit individuals is a target for improper or illegal penetration.
- Penetration widens a broad range of activities like hackers who attempt to penetrate systems for confidential data of banks, employees who attempt to penetrate for confidential data of offices, and many more can be the reasons.
- Security testing attempts to verify that protection mechanisms built into a system will protect it from improper penetration.
- The system's security must be tested for invulnerability from front attack - but must also be tested for invulnerability from side or back attack.
- The objective of security testing is to determine if the security of the system can be attacked and breached.
- Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended.
- The security testing is used to determine whether or not the software can handle breaches or recover from them if security is breached.
- Security testing is an integral part of software testing, which is used to discover the weaknesses, risks or threats in the software application and also make sure the security of the software applications or systems.
- The goal of security testing is to find all the potential ambiguities and vulnerabilities of the application so that the software does not stop working.
- If we perform security testing, then it helps us to identify all the possible security threats and also help the programmer to fix those errors.
- Security testing verifies that the system accomplishes all the security requirements and validates the effectiveness of these security measures.
- Security testing also aims at verifying six basic principles as listed below:
 1. **Confidentiality:** Means preserving authorized restrictions on information access.
 2. **Integrity:** Means guarding against improper information modification/distribution.
 3. **Availability:** Ensuring timely and reliable access to and use of information.
 4. **Authentication:** Ensures that the individual is who he/she claims to be, but says nothing about the access rights of the individual.
 5. **Authorization:** Means giving someone permission to do or have something.
 6. **Non-repudiation:** Ensuring that the originators of messages cannot deny that they in fact sent the messages.

Advantages of Security Testing:

1. This type of testing determines whether proper techniques are used to identify security risks.
2. The security testing verifies that appropriate protection techniques are followed to secure the system.
3. It ensures that the system is able to protect its data and maintain its functionality.
4. It conducts tests to ensure that the implemented security measures are working properly.

3.2.10 Internationalization Testing

- The process of adapting software to a specific locale, taking into account its language, dialect, local conventions and culture, is called localization or sometimes internationalization. Testing the software is localization/internationalization testing.
- Internationalization testing is the process that ensures that software product's functionality is not broken and all the messages are properly externalized when used in different languages and locale.
- Internationalization testing is the process of verifying the application under test to work uniformly across multiple regions and cultures.
- The purpose of internationalization testing is to check if the code can handle all international support without breaking functionality that might cause data loss or data integrity issues.
- Globalization testing verifies if there is proper functionality of the product with any of the locale settings.
- The objective or goal of internationalization testing (also called localization testing) is to determine if the software has any problems related to transitioning it to other geographic locations.
- Internationalization testing includes adaptation to different languages, cultures, customs, and standards.
- The tester, in internationalization testing must be fluent in each language used to test the system in order to confirm proper translation of screens, reports, error messages, help, and other items.
- The internationalization testing is the procedure of developing and planning the software or the application (software product) or file content, which allows us to localize our application for any given language, culture, and region without demanding any changes in the source code.
- Internationalization testing is also known as I₁₈N testing and here 18 is the number between I and N in the Internationalization word.

Basic Terms in Internationalization Testing:**1. Internationalization (I_{18n}):**

- It is predominantly used as an umbrella term to mean all activities that are required to make the software available for international market. This includes both development and testing activities.
- In the short form I_{18n}, the subscript 18 is used to mean that there are 18 characters between "I" and the last "n" in the word "internationalization."
- The testing that is done in various phases to ensure that all those activities are done right is called internationalization testing or I_{18n} testing.

2. Localization (L_{10n}):

- It is a term used to mean the translation work of all software resources such as messages to the target language and conventions.
- In the short form L_{10n}, the subscript 10 is used to indicate that there are 10 characters between "L" and "n" in the word "localization."
- This translation of messages and documentation is done by a set of language experts who understand English and the target language into which the software is translated.
- In cases where an accurate translation is needed, the messages along with the context of usage is given to the L_{10n} team.

3. Globalization (G_{11n}):

- Globalization (G_{11n}) is a term that is not very popular but used to mean internationalization and localization.
- Globalization (G_{11n}) term is used by a set of organizations, which would like to separate the internationalization from the localization set of activities.
- This separation may be needed because the localization activities are handled by a totally different set of language experts (per language), and these activities are generally outsourced.
- Some companies/organizations use the term I_{18n}, to mean only the coding and testing activities, not the translation.

Phases in Internationalization Testing:

- Testing for internationalization requires a clear understanding of all activities involved and their sequence.
- Since, the job of testing is to ensure the correctness of activities done earlier by other teams, this section also elaborates on activities that are done outside the testing teams.
- Fig. 3.15 shows the various major activities involved in internationalization testing. Some important aspects of internationalization testing are given below:
 1. Testing the code for how it handles input, strings and sorting items.
 2. Display of messages for various languages.
 3. Processing of messages for various languages and conventions.

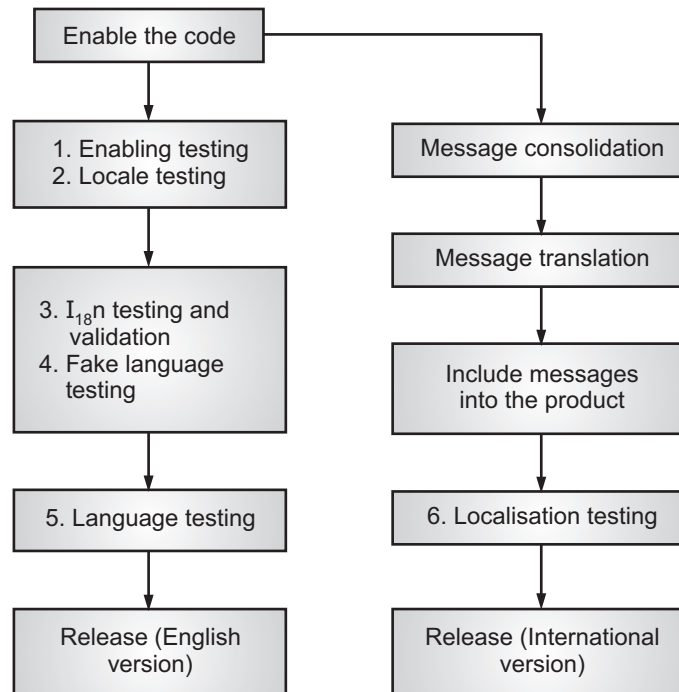


Fig. 3.15

- Localization testing process focused on adapting a globalized application to a particular culture/locale.

Advantages of Localization Testing

1. It has more flexibility and scalability.
 2. Helps in reducing the time and cost for testing.
- The market today is competitive; clients from all over the world are spreading globally with technology means an application needs to have global settings like functionality, view ability in cross-browsers, view ability in multiple platforms and readability.
 - Globalization testing is a testing method that checks proper functionality of the product with any of the culture/locale settings using every type of international input possible.

Advantages of Globalization Testing:

1. This testing technique ensures that the application could be used in various languages without need of rewriting the entire software code.
2. Globalization testing will increase the code design and quality of the software product.
3. Globalization testing will enhance the customer base around the world.
4. Globalization testing will help us to decrease the cost and time for localization testing.
5. Globalization testing will provide us more scalability and flexibility.

Disadvantages of Globalization Testing:

1. In Globalization testing, the test engineer might face the schedule challenge.
2. Needed the domain expert to perform globalization testing.
3. Need to hire a local translator, which makes this procedure costlier.

3.3 ALPHA AND BETA TESTING

- Alpha and Beta testing are the two types of testing used to test software which are based on feedback from real customers using real products in real environments.

3.3.1 Alpha Testing

- Alpha testing usually comes after system testing and involves both white and black-box testing techniques.
- The company employees test the software for functionality and give the feedback. After this testing phase any functions and features may be added to the software.
- The main **Features of Alpha testing** are:
 - Outside users are not involved while testing;
 - White box and black-box practices are used;
 - Developers are involved.
- Alpha testing is performed at the developer's sites, with the developer checking over the customer's shoulder as they use the system to determine errors.
- Alpha testing assesses the performance of the software in the environment in which it is developed.
- On completion of alpha testing, users report the errors to software developers so that they can correct them.
- Fig. 3.16 shows alpha testing.

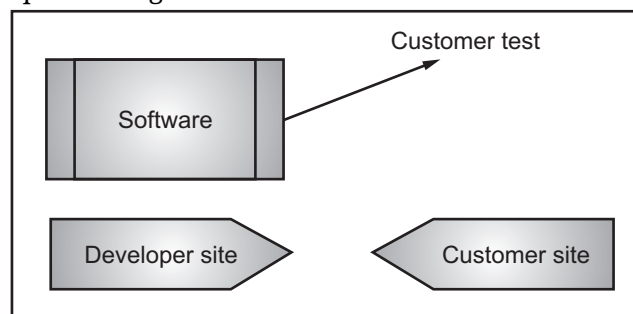


Fig. 3.16: Alpha Testing Approach

Advantages of Alpha Testing:

1. Alpha testing provides better view about the reliability of the software at an early stage.
2. Alpha testing helps simulate real time user behavior and environment.

3. Alpha testing detect many showstopper or serious errors.
4. Alpha testing has ability to provide early detection of errors with respect to design and functionality.

Disadvantages of Alpha Testing:

1. In alpha testing depth functionality cannot be tested as software is still under development stage.
2. In alpha testing, sometimes developers and testers are dissatisfied with the results of alpha testing.

3.3.2 Beta Testing

- Beta testing used to describe the external testing process in which the software is sent out to a select group of potential customers who use it in a real-world environment.
- Beta testing usually occurs toward the end of the software product development cycle and ideally should just be a validation that the software is ready to release to real customers.
- Beta tests in beta testing can be a good way to find compatibility and configuration bugs.
- Beta testing is 'live' software testing technique. Beta testing conducted in an environment, which is not controlled by the developer i.e., the testing is performed without any interference from the developer.
- Beta testing is performed to know whether the developed software satisfies user requirements or expectations and fits within the organizational processes.
- Fig. 3.17 shows the approach of beta testing.
- Both alpha and beta testing are very important while checking the software functionality and are necessary to make sure that all users' requirements are met in the most efficient way.

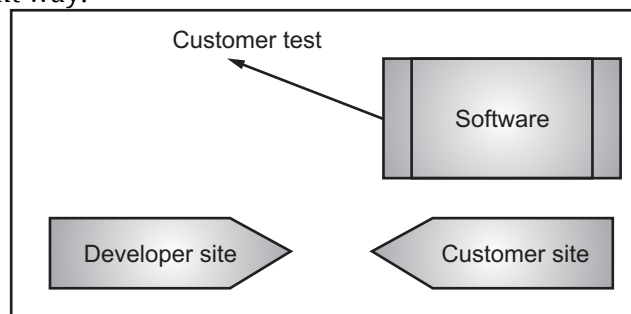


Fig. 3.17: Beta Testing Approach

Advantages of Beta Testing:

1. Beta testing allows an organization/firm to test post-launch infrastructure.
2. Beta testing reduces product failure risk via customer validation.
3. Beta testing improves product quality via customer feedback.
4. Beta testing creates goodwill with customers and increases customer satisfaction.

Disadvantages of Beta Testing:

1. Test management is an major issue in beta testing. As compared to other testing techniques which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control.
2. In beta testing, finding the right beta users and maintaining their participation could be a challenge.

Comparison Alpha Testing and Beta Testing:

Sr. No.	Alpha Testing	Beta Testing
1.	Alpha testing performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients or end users who are not employees of the organization.
2.	Alpha testing performed at developer's site.	Beta testing is performed at client location or end user of the product.
3.	Reliability and security testing are not performed in depth alpha testing.	Reliability, security, robustness are checked during beta testing.
4.	Alpha testing involves both the white-box and black-box techniques.	Beta testing typically uses black-box testing.
5.	Alpha testing requires lab environment or testing environment.	Beta testing doesn't require any lab environment or testing environment. Software is made available to the public and is said to be real time environment.
6.	Long execution cycle may be required for alpha testing.	Only few weeks of execution are required for beta testing.
7.	Critical issues or fixes can be addressed by developers immediately in alpha testing.	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product.
8.	Alpha testing is to ensure the quality of the product before moving to beta testing.	Beta testing also concentrates on quality of the product, but gathers users input on the product and ensures that the product is ready for real time users.

3.4 USABILITY AND ACCESSIBILITY TESTING

- Usability testing is used for measuring how easy and user-friendly a software application is. A small set of target end-users, use software application to expose usability defects.
- Usability testing mainly focuses on user's ease of using application, flexibility of application to handle controls and ability of application to meet its objectives.
- Accessibility testing is defined as a type of software testing performed to ensure that the application being tested is usable by people with disabilities like hearing, color blindness, old age and other disadvantaged groups.
- Accessibility testing is a subset of usability testing.

3.4.1 Usability Testing

- Usability testing is a software testing technique used in user-centered interaction design to evaluate a software product by testing it on users.
- Usability testing attempts to characterize the "look and feel" and usage aspects of the software, from the point of view of its users.
- Usability testing is done to find/detect out whether the general users are experiencing any difficulty in using the software.
- In other words, usability testing determines the user-friendliness or ease of usage of the software.
- These testing are done by various groups of users. So the alpha, beta and acceptance testing as discussed in the system testing can also be part of usability testing.
- Usability testing is mostly concerned with checking user interface specifications. During this testing, the various GUI interfaces and aspects related to the user interface requirements are tested.
- Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users. Usability is a quality attribute that assesses the easiness with which the user interfaces are used.
- The purpose of executing the usability testing is to check that the application should be easy to use for the end-user who is meant to use it.
- Usability testing is a technique for ensuring that the end-users of a product can carry out the intended tasks efficiently, effectively and satisfactory.
- Usability tests are performed in order to help verify that the software product or system is easy to use and that the user interface appearance is appealing. Usability tests consider the human element in system operation.
- Some of the characteristics of usability testing or usability validation are as follows:
 1. Usability testing tests the product from the user's point of view. It encompasses a range of techniques for identifying how users actually interact with and use the product.

2. Usability testing is for checking the product to see if it is easy to use for the various categories of users.
3. Usability testing is a process to identify discrepancies between the user interface of the product and the user requirements.

Process of Usability Testing:

- Fig. 3.18 shows process of usability testing.

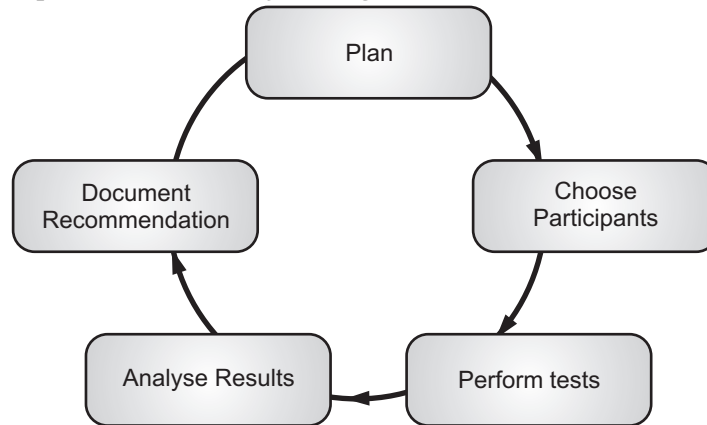


Fig. 3.18

- Steps in usability testing are explained below:
 - Step 1: Planning:** In this step, the team makes the test plan and generates some document samples that help the testing team complete the usability testing tasks.
 - Step 2: Team Recruiting:** In this step, we will move to the next step of usability testing, which team is recruiting. As the name suggests itself, here we will hire or recruit the end-user delegates and the participants or the test engineers as per the budget and density of the product.
 - Step 3: Test Execution:** In the test execution step, the test engineers execute the usability testing and implement their assigned responsibilities.
 - Step 4: Test Result Documentation:** The test result documentation step includes the results based on the test execution step and then proceeds for further analysis.
 - Step 5: Data Analysis:** In this step, the response or the feedback is obtained from usability testing evaluation in the data analysis phase. And the outcomes are classified and the patterns are acknowledged. In this step, the data from usability tests are wholly evaluated to get expressive implications and help us provide actionable suggestions to enhance the overall usability of the product.
 - Step 6: Reporting:** After performing all the above steps successfully, we will finally reach the last step of the usability testing process is reporting.

Advantages of Usability Testing:

1. In usability testing, the direct feedback from the potential user helps in creating a product that meets user's expectations.
2. In usability testing, increased user satisfaction with the software product leads to a meeting of organizational goals.
3. In usability testing, the issues are identified before the actual release of the software product, this maximizes the chances of the product's success.
4. In usability testing, enhanced user satisfaction with the software product ensures to deliver a good quality product.
5. Usability testing helps us to make the software more efficient and applicable.

Disadvantage of Usability Testing:

1. Usability testing technique requires many resources to establish a usability test.
2. In usability testing, hiring or recruiting a usability test engineer could be costly.

3.4.2 Accessibility Testing

- Accessibility testing is a subset of usability testing that takes into consideration both users with and without disabilities.
- In software testing, accessibility testing is widely used to check the application for disabled persons and make sure the developer will create the application which can be accessible by all types of users, like a regular user and physically challenged (color blindness, learning disabilities, and so on)
- The aim of accessibility testing is to determine whether the software or application is accessible for disabled people.
- Here, disability means deafness, colorblindness, mental disability, blindness, old age, and other disabilities.
- Various checks are performed, such as font size for the visually disabled, color and contrast for those with color blindness, etc.
- Accessibility is a subset of usability and should be included as part of usability test planning. Accessibility to the software product can be provided by following two means:
 1. **Basic Accessibility:** Making use of accessibility features provided by the underlying infrastructure (for example, operating system), called basic accessibility.
 2. **Product Accessibility:** Providing accessibility in the product through standards and guidelines, called product accessibility.
- According to the World Health Organization (WHO), 15% of the population has a disability, which is the reason accessibility testing has become important.
- Accessibility test is geared towards normal users as well as users with different types of disabilities.
- Accessibility aims to cater people of different abilities such as, Visual Impairments, Physical Impairment, Hearing Impairment, Cognitive Impairment, Learning Impairment and so on.

3.4.3 Configuration Testing

- Configuration testing is the process of testing the system with each one of the supported software and hardware configurations.
- The execution area supports configuration testing by allowing reuse of the created tests. Configuration testing is the process of checking the operation of the software we are testing with various types of hardware.
- Configuration testing is the process of testing the system with each one of the supported software and hardware configurations.
- Configuration testing is a software testing technique of testing a system under development on multiple machines that have different combinations or configurations of hardware and software.
- In configuration testing the software application is tested with multiple combinations of software and hardware in order to evaluate the functional requirements and find out optimal configurations under which the software application works without any defects or flaws.
- The performance of the system or an application is tested against each of the supported hardware and software configurations.
- For example, In a computer system, the Desktop applications will be of 2 tier or 3 tier, here we will consider a 3 tier Desktop application which is developed using ASP.Net and consists of Client, Business Logic Server and Database Server.
- Where each component supports following mentioned platforms:
 - **Client Platform:** Windows XP, Window7 OS, windows 8 OS , etc.
 - **Server Platform:** Windows Server 2008 R2, Windows Server 2008 R2, Windows Server 2012R2 etc.
 - **Database:** SQL Server 2008, SQL Server 2008R2, SQL Server 2012, etc.
- A tester has to test the Combination of Client, Server and Database with combinations of the above-mentioned platforms and database versions to ensure that the application is functioning properly and does not fail.
- Configuration testing is not only restricted to Software but also applicable for Hardware, where we test different hardware devices like Printers, Scanners, Web cams, etc. that support the application under test.

3.4.4 Compatibility Testing

- Compatibility helps to determine if a software product will function correctly with other hardware/software in the intended environment.
- Software testing requires functional validation across a wide variety of hardware platforms, operating systems, databases, browsers, servers, and various networks. Failure to test these can introduce expensive bugs.
- Compatibility testing checks the software applications or products compatibility against various aspects such as operating systems, hardware components, third-party software and internet browsers.

- Compatibility testing determines if the program can operate on various hardware, operating systems, applications, network settings or mobile devices.
- Compatibility testing cover compatibility between a different hardware, various operating systems' compatibility, networks, computers and application environments.
- Compatibility testing is non-functional software testing technique. Compatibility testing is carried out to verify whether an application or software can run on different devices, browsers, operating systems hardware and networks.
- Compatibility testing conducted on the application to evaluate the application's compatibility with the computing environment.
- Compatibility testing is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments.
- Computing environment may contain some or all of the following mentioned elements:
 - Computing capacity of Hardware Platform (IBM 360, HP 9000, etc.)
 - Bandwidth handling capacity of networking hardware.
 - Compatibility of peripherals (Printer, DVD drive, etc.)
 - Operating systems (Linux, UNIX, Windows. etc.)
 - Database (Oracle, Sybase, DB2, MySQL etc.)
 - Other System Software (Web server, Networking/messaging tool, etc.)
 - Browser compatibility (Firefox, Netscape, Internet Explorer, Safari, etc.)
- Fig. 3.19 shows process of compatibility testing.

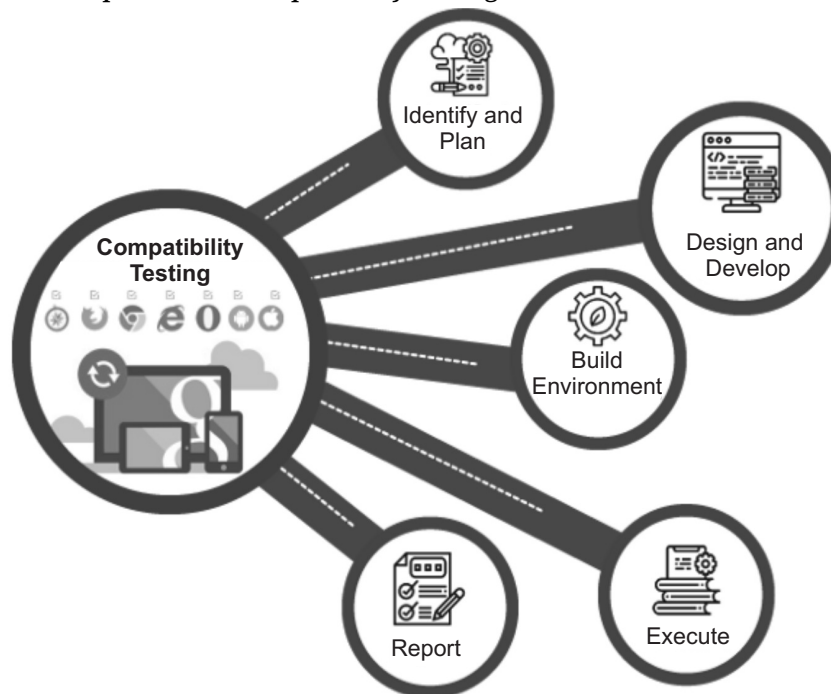


Fig. 3.19

- Various categories of compatibility testing are explained below:
 1. **Hardware Compatibility Testing**, a developed application or software is tested on various hardware configurations to make sure it works on them in the desired manner.
 2. **Network Compatibility Testing**, an application is checked on different networks such as WiFi, 3G, 4G, etc. for various parameters such as bandwidth, speed, etc.
 3. **Operating System Compatibility Testing**, an application or software is tested by running it on different operating systems such as Windows, Mac, Linux, etc.
 4. **Device Compatibility Testing** we can check an application's compatibility with various external devices such as Bluetooth, USB device, printer, etc.
 5. **Mobile Compatibility Testing**, an application is tested on various mobile devices with different operating systems such as Android, iOS, and Windows.
 6. **Browser Compatibility Testing** tests an application's working on various browsers such as Chrome, Firefox, Internet Explorer, etc. to make sure it is compatible with them. To know more about cross-browser testing, [click here](#).
 7. **Software Compatibility Testing**, an application's compatibility is checked with other software. For example, if an application has a feature that allows users to generate a PDF file, the user should be able to open this PDF file in adobe acrobat reader (being one of the most used pdf readers).
 8. **Version Compatibility Testing**, an application's compatibility with its older and newer versions is checked. It is further divided into following two types:
 - (i) **Backward Compatibility Testing** tests an application's compatibility with its older versions. For example, Office 2010's PowerPoint software should be able to open a presentation made in Office 2007. It is easier to test as the functions of older versions are known. In simple words, backward compatibility testing is performed to check the behavior and compatibility of the hardware or software with their older versions.
 - (ii) **Forward Compatibility Testing** is performed to make sure an application is compatible with its future versions as much as possible. Forward compatibility testing process is used to check the behavior and compatibility of the software or hardware with their recent or new versions. It is difficult to predict forward compatibility testing because the changes made in the newer versions are unknown.

Advantages of Compatibility Testing:

1. Compatibility testing enhances software development process.
2. Compatibility testing is capable of timely identification of the defects in mobile and Web apps even in the difficult areas.
3. Compatibility testing ensures to meet the fundamental expectation of end users.
4. The testing practices effectively perform system compatibility tests on different browsers, platforms, configurations, OS, hardware, etc.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which are the different testing levels helps to check behavior and performance for software testing?
 - (a) unit testing (used to test if software modules are satisfying the given requirement or not).
 - (b) integration testing (used to identify the defects at the interaction between integrated components or units).
 - (c) system testing (used to test the software's functional and non-functional requirements) and acceptance testing (used to evaluate whether a specification or the requirements are met as per its delivery)
 - (d) All of the mentioned
2. Which is a type of software testing where individual units or components of the software are tested?
 - (a) unit testing
 - (b) integration testing
 - (c) system testing
 - (d) acceptance testing
3. Which software testing combines the parts of an application to determine if they function correctly?
 - (a) unit testing
 - (b) integration testing
 - (c) system testing
 - (d) acceptance testing
4. Which software testing tests the system as a whole?
 - (a) unit testing
 - (b) integration testing
 - (c) system testing
 - (d) acceptance testing
5. Which software testing is performed to ensure that the functional, behavioral, and performance requirements of the software are met?
 - (a) unit testing
 - (b) integration testing
 - (c) system testing
 - (d) acceptance testing
6. Which software testing is designed to determine the performance of the software (especially real-time and embedded systems) at the run-time in the context of the entire computer-based system?
 - (a) Unit
 - (b) System
 - (c) Performance
 - (d) Integration
7. Which software testing is a non-functional software testing process in which the performance of software application is tested under a specific expected load?
 - (a) Stress
 - (b) Load
 - (c) Performance
 - (d) Unit
8. Which software testing includes testing the behavior of the software under abnormal conditions (extremely heavy load conditions)?
 - (a) Stress
 - (b) Load
 - (c) Performance
 - (d) Unit

9. Which is a software testing technique in which the software application is tested with multiple combinations of software and hardware in order to evaluate the functional requirements and find out optimal configurations under which the software application works without any defects or flaws?
(a) Stress (b) Load
(c) Performance (d) Configuration
10. Which is a type of software testing that seeks to uncover software errors after changes to the program (e.g. bug fixes or new functionality) have been made, by retesting the program?
(a) Stress (b) Regression
(c) Performance (d) Configuration
11. Which is used to identify the levels of testing which are to be applied along with the methods, techniques, and tools to be used during testing?
(a) a testing strategy (b) a testing plan
(c) a testing technique (d) None of the mentioned
12. Which testing is a process of ensuring the adaptability of software to different cultures (locale) and languages around the world accordingly without any modifications in source code?
(a) Stress (b) Regression
(c) Internationalization (d) Configuration
13. Which testing is considered as a form of internal acceptance testing in which the users test the software at the developer's site? ?
(a) Alpha (b) Beta
(c) Both (a) and (b) (d) None of the mentioned
14. Which testing is used to identify any error(s) and improvements in the software by observing the users through their usage and operation?
(a) Accessibility (b) Configuration
(c) Compatibility (d) Usability
15. Which testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view?
(a) Stress (b) Security
(c) Performance (d) Configuration
16. Which testing is a type of testing which determines the usability of a product to the people having disabilities (hearing problem, color blindness, mentally disabled etc).?
(a) Accessibility (b) Configuration
(c) Compatibility (d) Usability
17. Which testing is used to check whether the software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices?
(a) Accessibility (b) Configuration
(c) Compatibility (d) Usability

18. Which testing is the process of testing the system under each configuration of the supported software and hardware and ensures the system can work without any flaws?

- (a) Accessibility (b) Configuration
(c) Compatibility (d) Usability

Answers

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

Q. II Fill in the Blanks:

- _____ of testing include different methodologies that can be used while conducting software testing.
- The purpose of _____ testing is to validate that each unit of the software code performs as expected.
- _____ testing determines how the software application behaves while being accessed by multiple users simultaneously.
- A software testing _____ should contain complete information about the procedure to perform testing and the purpose and requirements of testing.
- The objective of _____ testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.
- Regression testing _____ the software or part of it to ensure that the components, features and functions, which were previously working properly, do not fail as a result of the error correction process and integration of modules.
- _____ testing is a testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement.
- _____ testing is a type of system testing in which the system is forced to fail in different ways to check whether the software recovers from the failures such as system crashes, hardware failures etc., without any data loss.
- _____ testing is also shortly known as i18n, in which 18 represents the number of characters in between I & N in the word Internationalization. Internationalization simply makes applications ready for localization.
- _____ testing specifies the way in which a system reacts when it is made to operate beyond its performance and capacity limits.
- _____ testing is a type of performance testing used to determine an application's behavior when exposed to extreme traffic variations.
- In system testing, the system is tested against _____ -functional requirements such as accuracy, reliability, and speed.

13. _____ testing is used to test several factors (speed (refers to the extent how quickly a system is able to respond to its users), scalability (refers to the extent to which the system is able to handle the load given to it) and so on) that play an important role in improving the overall performance of the system.
14. _____ testing is a formal testing with respect to user needs, requirements and business processes conducted to determine whether or not a system satisfies the acceptance criteria.'
15. _____ testing is performed to know whether the developed software satisfies user requirements and assesses the performance of the software at user's site.
16. Software _____ strategy is an approach which incorporates planning of the steps to test the software along with the planning of time, effort & resources that will be required to test the software.
17. _____ testing is a software testing method used to ensure that the software application can function in any culture or locale (language, territory or code page) by testing the software functionalities using each type of international input possible. The purpose of Globalization testing is to ensure that software can be used internationally or worldwide.
18. In _____ integration testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.
19. _____ testing determine whether the software application or product is proficient enough to run in different browsers, database, hardware, operating system, mobile devices and networks.
20. _____ testing is a technique used in user-centered interaction design to evaluate a product by testing it on users and checking the user-friendliness, efficiency, and accuracy of the software application.
21. A _____ testing helps validate the efficiency of software and system based on how it performs during heavy load.

Answers

1. Levels	2. Unit	3. Load	4. strategy
5. integration	6. re-tests	7. system	8. Recovery
9. Internationalization	10. Stress	11. Spike	12. non
13. Performance	14. Acceptance	15. Beta	16. testing
17. Globalization	18. top-down	19. Compatibility	20. Usability
21. soak			

Q. III State True or False:

1. The purpose of levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

2. Unit testing is performed to test the individual units of software.
3. The goal of load testing is to improve performance bottlenecks and to ensure stability and smooth functioning of software application before deployment.
4. Testing strategy also decides test cases, test specifications, test case decisions, and puts them together for execution.
5. In top-up integration testing each module at lower levels is tested with higher modules until all modules are tested.
6. The integration testing is aimed at ensuring that all the modules work properly as per the user requirements when they are put together (that is, integrated).
7. Localization testing is the software testing process for checking the localized version of a product for that particular culture or locale settings.
8. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage under various workload.
9. Accessibility testing is a subset of usability testing where in the users under consideration are specifically the people with disabilities such as difficulty to hear well or hear clearly, learning difficulties, poor memory, color blindness or poor eyesight and so on.
10. Validation testing focuses on the testing of software against the requirements specified by the customer.
11. Object-oriented testing is a combination of various testing techniques that help to verify and validate object-oriented software.
12. Unit testing is used to verify the code produced during software coding and is responsible for assessing the correctness of a particular unit of source code.
13. Security testing is a process intended to reveal flaws in the security mechanisms of an information system that protect data and maintain functionality as intended.
14. Configuration testing the performance of the system or an application is tested against each of the supported hardware and software configurations.
15. Soak testing is a non-functional performance testing under which an application is exposed to a continuous load for a pre-determined period. The term 'soak' itself explained the test's purpose.
16. Regression testing consists of re-executing those tests that are impacted by the code changes.

Answers

1. (T)	2. (T)	3. (T)	4. (T)	5. (F)	6. (T)	7. (T)	8. (T)	9. (T)	10. (T)
11. (T)	12. (T)	13. (T)	14. (T)	15. (T)	16. (T)				

Q. IV Answer the following Questions:**(A) Short Answer Questions:**

1. What software testing strategy?
2. List levels for testing.
3. Define system testing.
4. What is the purpose of load testing?
5. Define acceptance testing?
6. Compare alpha and beta testing any two points.
7. Define security testing.
8. What is the purpose of usability testing?
9. Define configuration testing?
10. Define performance testing.
11. List types of integration testing.
12. Define load testing.
13. Define compatibility testing?
14. Give the purpose of accessibility testing.
15. Define internationalization testing.

(B) Long Answer Questions:

1. What is meant by testing levels? Explain with the example.
2. Write a short note on: A strategic approach to software testing.
3. What is unit testing? How it works? Also state its advantages and disadvantages.
4. What is alpha and beta testing? Differentiate between them. Any four points.
5. What is performance testing?
6. What is system testing? How it test the system? Also list its different types.
7. What is load testing? State its advantages and disadvantages.
8. What is usability testing? Explain its process diagrammatically.
9. What is integration testing? How it works? Explain with example. Also list its types.
10. Explain compatibility testing with example.
11. Describe configuration testing with the help of example.
12. What is internationalization testing? Also explain its phases diagrammatically.
13. Differentiate between system, performance, load testing.
14. What is stress testing? Describe its process diagrammatically.
15. Write a short note on: Test strategic for conventional software.
16. What is regression testing? State its advantages and disadvantages.



Testing Web Applications

Objectives...

- To understand Testing of Web Applications
 - To learn Dimension of Quality
 - To study Testing Strategy for WebApp
-

4.0 INTRODUCTION

- Web applications are the software applications that reside on Web servers and are accessed using a Web browser (client).
 - These Web applications are developed using popular technologies such as HTML, CSS, JavaScript and are utilized through multiple Browsers such as Windows Explorer, Chrome, Safari, Firefox, etc.
 - A Web application (or Web app) is application software or program that runs on a Web server.
 - A Web application is stored on a remote server and delivered over the Internet through a browser interface.
 - The purpose of a Web apps are efficiently communicating and exchanging information with its users while being compliant with a variety of browsers and Operating Systems (OSs).
 - Web app testing also known as Web testing. Web testing is a software testing practice/technique that helps ensure the quality and functionalities of the app according to the requirements.
 - Web testing must detect all underlying issues, such as functional discrepancies, security breaches, integration problems, environmental issues, or traffic stress before it is delivered.
 - A Web-based application is an application which can be accessed and used over the network like Internet. Internet is the worldwide collection of interconnected networks.
 - A Web application is a client-server program in which the client runs or request in a Web browser.
-

- Web servers pass requests to Web application servers, which in turn pass results back to the requesting browser.

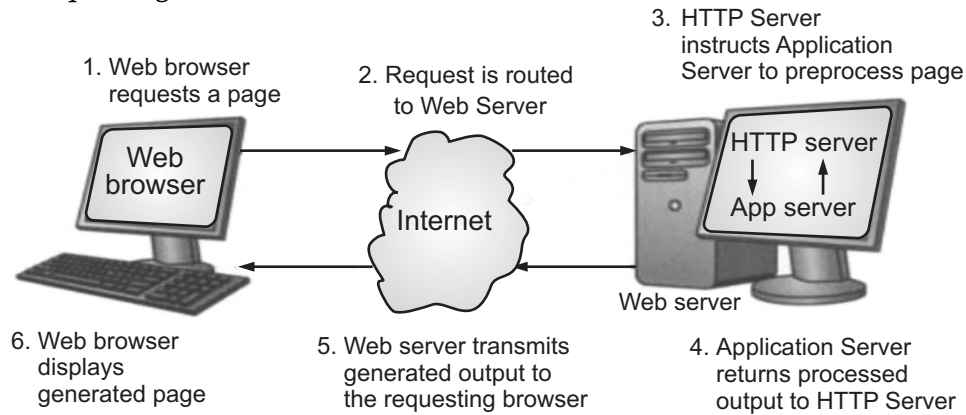


Fig. 4.1: Working of Web Application

- In general, a user sends a request to the Web server using Web browsers such as Google Chrome, Microsoft Edge, Mozilla Firefox etc. over the Internet.
- Then, the request of client is forwarded to the appropriate Web application server by the Web server.
- Web application server performs the requested operations/tasks like processing the database, querying the databases; produces the result of the requested data.
- The obtained result is sent to the Web server by the web application server along with the requested data/information or processed data.
- The Web server responds to the user with the requested or processed data/information and provides the result to the user's computer.
- Web testing is a software practice that ensures quality by testing that the functionality of a given web application is working as intended or as per the requirements.
- Web testing or website testing is checking the Web application or Website for potential bugs before its made live and is accessible to general public.
- Web testing checks for functionality, usability, security, compatibility, performance of the Web application or Website.
- The goal of Web testing is to exercise each of the many dimensions of Web application quality with the intent of finding errors or uncovering issues that may lead to quality failures.
- Web testing focuses on content, function, structure, usability, navigability, performance, compatibility, interoperability, capacity and security.
- Web testing is the process used to validate that a website application can meet specific functionality, security, usability, accessibility, visual, and performance criteria.

Techniques for Web Application Testing:

- Testing is a fundamental challenge when developing/creating Web applications. Different types of browsers, interfaces, security threats and overall app integration are just a few of the issues faced by developers.
- Since, testing is a crucial phase in the development process, the developer should expect to run into unforeseen issues associated with both the Web application and the testing process itself.
- Following are the various testing techniques used in Web application:

1. Functionality Testing:

- Functional testing can be considered as a way of ensuring that all the functionalities of features of the software are working as required.
- The functionality testing is one of the most common tests for Web applications. The functionality testing checks if the initial build works as per its design.
- The functional testing ensures that the functionalities of a Web application are properly functioning or not.
- The functionality testing often covers link testing, form validation, cookie testing, HTML and CSS validation and database connection checkup.
- Functionality testing of a Website is a process has several testing parameters such as user interface, APIs, database testing, security testing, client and server testing and basic website functionalities.
- In simple words, the functional testing is performed to test the functionalities of each feature on the website.

2. Interface Testing:

- Interface testing is used to validate whether systems pass data and control information properly over a computer network.
- This testing examines how the Web interface responds to emulated interruptions, as well as its compatibility and interaction between different servers.
- Three key areas to focus on interface testing are the application Server, Web server and Database server.

3. Compatibility Testing:

- Website compatibility is very important for the overall Web testing aspect. The compatibility include Brower compatibility, Operating Systems (OSs) compatibility, Mobile browsing compatibility and Printing options.
- Compatibility testing checks whether or not the Web application design is compatible with a variety of browsers and devices.

4. Performance Testing:

- Performance testing is used to ensure that the Websites or Web applications will perform well under their expected workload.

- A specific Web application is tested in terms of how better it can perform under stress conditions and heavy workload.
- It also ensures how the application is able to perform under different internet speeds, networks and browsers are also worked upon.
- In other words, performance testing is load testing for Web apps. Besides the tests on network traffic load, stress tests and scalability tests are also crucial to the web performance, especially when it is potentially released to a large audience.

5. Security Testing:

- Security testing is performed to verify if the application is secured on Web as data theft and unauthorized access. Website security activities includes:
 - (i) Tasting if Web application is protected from unauthorized users/hackers.
 - (ii) Tasting Website for unauthorized access.

4.1 DIMENSIONS OF QUALITY

- A quality software product is a product that meets the expectations of the customers or end users. Quality is incorporated into a Web application as a consequence of good analysis and design.
- It is evaluated by applying a series of technical reviews that assess various elements of the design model and by applying a testing process.
- Both reviews and testing examine one or more of the following quality dimensions:
 1. **Structure** is assessed to ensure that it properly delivers WebApp content and function, that it is extensible and that it can be supported as new content or functionality is added.
 2. **Content** is evaluated at both a syntactic and semantic level. At the **syntactic level**, spelling, punctuation, and grammar are assessed for text-based documents. At a **semantic level**, correctness (of information presented), consistency (across the entire content object and related objects), and lack of ambiguity are all assessed.
 3. **Interoperability** is tested to ensure that the WebApp properly interfaces with other applications and/or databases.
 4. **Function** is tested to uncover errors that indicate lack of conformance to customer requirements. Each WebApp function is assessed for correctness, instability, and general conformance to appropriate implementation standards (e.g., Java or Ajax standards).
 5. **Usability** is tested to ensure that each category of user is supported by the interface and can learn and apply all required navigation syntax and semantics.
 6. **Compatibility** is tested by executing the WebApp in a variety of different host configurations on both the client and server sides. The intent is to find errors that are specific to a unique host configuration.

7. **Security** is tested by assessing potential vulnerabilities and attempting to exploit each. Any successful penetration attempt is deemed a security failure.
 8. **Navigability** is tested to ensure that all navigation syntax and semantics are exercised to uncover any navigation errors (e.g., dead links, improper links and erroneous links).
 9. **Performance** is tested under a variety of operating conditions, configurations and loading to ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.
- A strategy and tactics for WebApp testing has been developed to exercise each of above quality dimensions.

4.2 ERRORS WITHIN A WEBAPP ENVIRONMENT

- The intent of Web application testing is to uncover errors and correct them. Errors encountered as a consequence of successful WebApp testing have a number of unique characteristics.
- These characteristics can have a profound impact on testing and error discovery.
 - A WebApp is created in a number of various configurations and within different environments. For this reason it may be difficult or impossible to reproduce an error outside the environment in which the error was originally encountered.
 - Because number of types of WebApp tests uncover problems that are first evidenced on the client side i.e., via an interface implemented on a specific browser or a personal communication device like PDA or mobile phone, we often see a symptom of the error, not the error itself.
 - Because Web applications reside within a client-server system, errors can be difficult to trace across three architectural layers namely, the client, the server or the network itself.
 - Some errors in Web application encountered due to the static operating environment i.e., the specific configuration in which testing is conducted, while others errors are attributable to the dynamic operating environment i.e., instantaneous resource loading or time-related errors.
 - Although some errors in Web application encountered due to result of incorrect design or improper HTML or other programming language (such as JavaScript) coding, many errors can be traced to the WebApp configuration.
- Above errors attributes suggest that environment plays an important role in the diagnosis of all errors uncovered during the WebApp testing.
- In number of situations (e.g., content testing), the site of the error is obvious, but in many other types of WebApp testing (e.g., navigation testing, performance testing, security testing) the underlying cause of the error may be considerably more difficult to determine.

4.3 TESTING STRATEGY FOR WEBAPP

- WebApp testing is more challenging than (conventional) software testing because of the WebApps works in a dynamic environment, (e.g., interoperates on several OS, browsers, hardware platforms, communication protocols and so on).
- The strategy for Web application testing adopts the basic principles for all software testing and applies a strategy and tactics that have been recommended for Object-Oriented (OO) systems.
- The following steps summarize the approach:
 - Step 1:** The content model for the WebApp is reviewed to uncover errors.
 - Step 2:** The interface model is reviewed to ensure that all use cases can be accommodated.
 - Step 3:** The design model for the WebApp is reviewed to uncover navigation errors.
 - Step 4:** The user interface is tested to uncover errors in presentation and/or navigation mechanics.
 - Step 5:** Selected functional components are unit tested.
 - Step 6:** Navigation throughout the architecture is tested.
 - Step 7:** The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
 - Step 8:** Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
 - Step 9:** Performance tests are conducted.
 - Step 10:** The WebApp is tested by a controlled and monitored population of end users; the results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns and WebApp security, reliability and performance.
- Because number of WebApps evolves continuously, the testing process is an ongoing activity, conducted by WebApp support staff.

4.4 TEST PLANNING

- Test planning the most important activity in WebApp testing. A test plan in Web testing provides a road map so that the Websites can be evaluated through requirements.
- Test planning ensures that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project.
- A test plan is a document that guides the whole process of testing. Test planning helps to estimate the time schedule and efforts needed for testing.
- Test planning also establishes a test environment, getting test personnel and test process/procedure before any testing can actually starts.

- Test plan is a detailed document/template that describes the test strategy, objectives, schedule, estimation and deliverables, and resources required for testing.
- It also helps us 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 which is minutely monitored and controlled by the test manager.
- The test planning establishes a systematic process for all work that the testing follows. In their book on WebApp testing, Steven Splaine, Stefan P. Jaskiel state:
“Except for the simplest of websites, it quickly becomes apparent that some sort of test planning is needed. All too often, the initial number of bugs found from ad hoc testing is large enough that not all of them are fixed the first time they are detected. This puts an additional burden on people who test websites and applications. Not only must they conjure up imaginative new tests, but they must also remember how previous tests were executed in order to reliably re-test the website/application, and ensure that known bugs have been removed and that no new bugs have been introduced.”
- The questions are:
 - How do we conjure up imaginative new tests?
 - What should those tests focus on?
- The answers to above questions are contained within a test plan. A WebApp test plan identifies:
 - The task set to be applied as testing consequences.
 - The work product to be produced as each testing task is executed.
 - The manner, in which the results of testing are evaluated, recorded and reused when regression testing is conducted.
- In some cases, the test plan is integrated with the project plan. In others, the test plan is a separate document.

4.5 THE TESTING PROCESS - AN OVERVIEW

- Testing plays a crucial role in the overall Web development process. However, more often than not, testing and evaluation is a neglected aspect of Web development.
- We begin the WebApp testing process with tests that exercise content and interface functionality that are immediately visible to end users.
- As testing proceeds, aspects of the design architecture and navigation are exercised. Finally, the focus shifts to tests that examine technological capabilities that are not always apparent to end users - WebApp infrastructure and implementation/installation issues.
- Fig. 4.2 shows the WebApp testing process with the design pyramid for WebApps.

- As the testing flow proceeds from left to right and top to bottom, user-visible elements of the WebApp design (top elements of the pyramid are tested first, followed by infrastructure design elements).

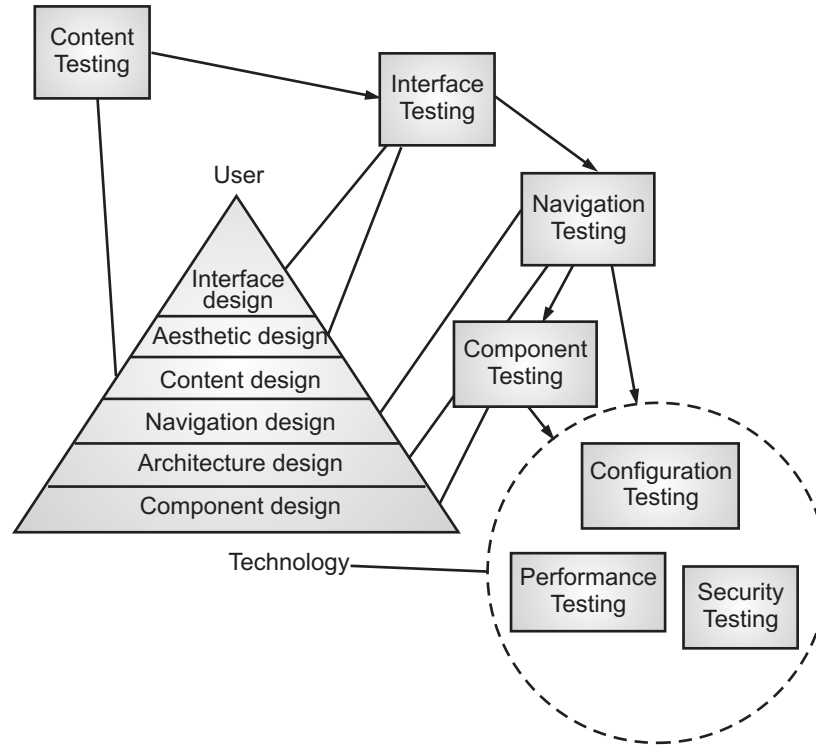


Fig. 4.2: Testing Process

- Various types of testing in Fig. 4.2 are explained below:
- Contents Testing:**
 - The content is a crucial part of any WebApp and playing a major role in making WebApps.
 - Content in WebApp should be meaningful. Web content can be used in the form of text, images, videos, articles, animations blogs, case studies and so on.
 - Errors in WebApp content can be as significant as incorrect information, improper organization or violation of intellectual property laws or as trivial as minor typographical mistakes.
 - Content testing attempts to uncover these errors and many other problems before the user encounters them. Content testing combines both reviews and the generation of executable test cases.
 - Review:** It is applied to uncover semantic errors in content.
 - Executable Testing:** It is used to uncover content errors that can be traced to dynamically derived content that is driven by data acquired from one or more databases.

- The important objectives of Content testing are given below:
 1. to uncover syntactic errors e.g., typos (a mistake made in the typing the text), grammar mistakes in text-based documents, graphical representations, and other media.
 2. to uncover semantic errors i.e., errors in the accuracy or completeness of information in any content object presented as navigation occurs.
 3. to find errors in the organization or structure of content that is presented to the end user.
- 2. **Database Testing:**
 - Database is one critical component of the Web application. Modern WebApps do much more than present static content objects.
 - In number of application domains, WebApps interface with sophisticated database management systems and build dynamic content objects that are created in real time using the data acquired from a database.
 - For example, a financial service WebApp can produce complex text-based, tabular, and graphical information about a specific equity (e.g., a stock or mutual fund).
 - The composite content object that presents this information is created dynamically after the user has made a request for information about a specific equity.
 - To accomplish this, the following steps are required:
 - Step 1:** An equities database is queried.
 - Step 2:** Relevant data are extracted from the database.
 - Step 3:** The extracted data must be organized as a content object
 - Step 4:** this content object (representing customized information requested by on end user) is transmitted to the client environment for display.
 - Errors can and do occur as a consequence of each of above steps. The objective/goal of database testing is to uncover these errors, but database testing is complicated by a variety of factors.
 - Some of them factors are given below:
 - The original client-side request for information is rarely presented in the form e.g., SQL that can be input to a DBM, for this reason tests should be designed to uncover errors made in translating the user's request into a form that can be processed by these DBMS.
 - The database may be remote to the server that houses the WebApp, for this reason tests that uncover errors in communication between the WebApp and the remote database must be developed.
 - Raw data acquired from the database must be transmitted to the WebApp server and properly formatted for subsequent transmittal to the client, for this reason tests that demonstrate the validity of the raw data received by the WebApp server

- must be developed and additional tests that demonstrate the validity of the transformations applied to the raw data to create valid content objects must also be created.
- Three dynamic content object(s) must be transmitted to the client in a form that can be displayed to the end user. Therefore, a series of tests must be designed to:
 - Uncover errors in the content object format.
 - Test compatibility with different client environment configurations.
 - Considering above four factors, test-case design methods should be applied for each of the “layers of interaction” shown in Fig. 4.3.
 - Testing should ensure that:
 - (i) Valid information is passed between the client and server from the interface layer.
 - (ii) The WebApp processes script correctly and properly extract or format user data.
 - (iii) User data are passed correctly to a server-side data transformation function that formats appropriate queries (e.g., SQL).
 - (iv) Queries are passed to a data management layer that communicates with database access routines (potentially located on another machine).
 - The data transformation, data management, and database access layers shown in Fig. 4.3, are often constructed with reusable components that have been validated separately and as a package.
 - If this is the case, WebApp testing focuses on the design of test cases to exercise the interactions between the client layer and the first two server layers (WebApp and data transformation) shown in the Fig. 4.3.

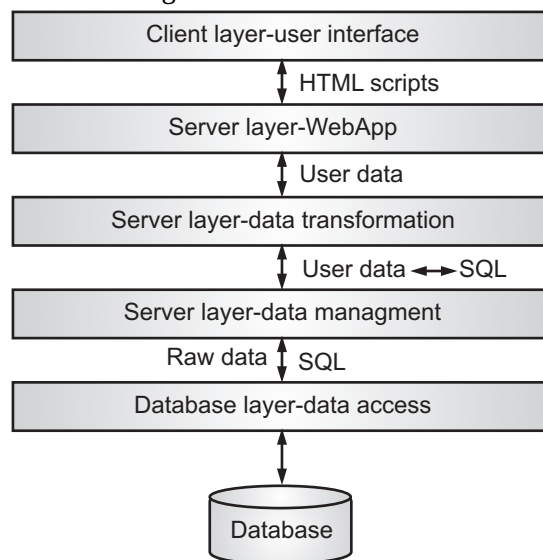


Fig. 4.3

- The user interface layer in the Fig. 4.3 is tested to ensure that scripts are properly constructed for each user query and properly transmitted to the server side.
- The WebApp layer in the Fig. 4.3 on the server side is tested to ensure that user data are properly extracted from scripts and properly transmitted to the data transformation layer on the server side.
- The data transformation functions are tested to ensure that the correct SQL (Structured Query Language) is created and passed to appropriate data management components.

3. User Interface Testing:

- Interface testing is performed to verify the interface and the dataflow from one system to other. Verification and Validation (V&V) of a WebApp user interface occurs at the following three distinct points:
 - During requirements analysis, the interface model is reviewed to ensure that it conforms to stakeholder requirements and to other elements of the requirements model.
 - During design the interface design model is reviewed to ensure that generic quality criteria established for all user interfaces have been achieved and that application-specific interface design issues have been properly addressed.
 - During testing, the focus shifts to the execution of application-specific aspects of user interaction as they are manifested by interface syntax and semantics. In addition; testing provides a final assessment of usability.

Strategy for Interface Testing:

- Interface testing exercises or evaluates interaction mechanisms and validates aesthetic aspects of the User Interface (UI). The overall strategy for interface testing is to uncover errors:
 - related to specific interface mechanisms (e.g., errors in the proper execution of a menu link or the way data are entered in a form).
 - in the way the interface implements the semantics of navigation, WebApp functionality, or content display.
- To accomplish interface testing strategy, a number of following tactical steps are initiated:

Step 1: Interface features are tested to ensure that design rules, aesthetics and related visual content are available for the user without error.

Step 2: Individual interface mechanisms are tested in a manner that is analogous to unit testing. For example, tests are designed to exercise all forms, client-side scripting, dynamic HTML, scripts, streaming content and application-specific interface mechanisms (e.g., a shopping cart for an e-commerce application).

Step 3: Each interface mechanism is tested within the context of a use case for a specific user category.

Step 4: The complete interface is tested against selected use cases to uncover errors in the semantics of the interface. It is at this stage that a series of usability tests are conducted.

Step 5: The interface is tested within a variety of environments e.g., browsers to ensure that it will be compatible.

Mechanisms for Testing Interface:

- When a user interacts with a WebApp, the interaction occurs through one or more following interface mechanisms:
 - **Dynamic HTML (DHTML):** DHTML is based on the properties of the HTML, JavaScript, CSS and DOM (Document Object Model) which helps in making dynamic content. Each and every Web page that contains DHTML is executed to ensure that the dynamic display is correct. In addition, a compatibility test should be conducted to ensure that the dynamic HTML works properly in the environmental configurations that support the WebApp.
 - **Links:** Links or Hyperlinks are the basic concept in the WebApp which allows linking documents to other documents or resources, link to specific parts of documents, or making apps available at a web address. Each and every navigation link in a WebApp is tested to ensure that the proper content object or function is reached. Testing includes links associated with the interface layout (e.g., menu bars, index items, links within each content object, and links to external WebApps).
 - **Client-side Scripting:** A client-side scripts are small programs which are downloaded, compiled and run by the browser. JavaScript is an example of client-side scripting language and widely used in dynamic websites. In WebApp testing the black-box tests are conducted to uncover any errors in processing as the script is executed. These black-box tests are often coupled with forms testing, because script input is often derived from data provided as part of forms processing.
 - **Cookies:** A cookie (Web cookie or, Browser cookie) is a small piece of data files that a server sends to a user's web browser. Both server-side and client-side testing are required cookies. On the server-side, tests should ensure that a cookie is properly constructed (contains correct data) and properly transmitted to the client-side when specific content or functionality is requested. On the client-side, tests determine whether the WebApp properly attaches existing cookies to a specific request (sent to the server).
 - **Forms:** A form in WbApp is used for entering or selecting information on a Website and it permits users to interact with a website. A form on a web page allows a user to enter data that is sent to a server for processing. At a macroscopic

level, tests are performed to ensure that, labels correctly identify fields within the form and that mandatory fields are identified visually for the user, the server receives all information contained within the form and that no data are lost in the transmission between client and server, appropriate defaults are used when the user does not select from a pull-down menu or set of buttons, scripts that perform error checking on data entered work properly and provide meaningful error messages. At a more targeted level, tests should ensure that form fields have proper width and data types, the form establishes appropriate safeguards that preclude the user from entering text strings longer than some predefined maximum, all appropriate options for pull-down menus are specified and ordered in a way that is meaningful to the end user etc.

- **Pop-up Windows:** A pop-up Window is a Web browser window that is smaller than standard windows and without some of the standard features such as toolbars or status bars. A series of tests ensure that the pop-up is properly sized and positioned, the pop-up does not cover the original WebApp window, the aesthetic design of the pop-up is consistent with the aesthetic design of the interface, scroll bars and other control mechanisms appended to the pop-up are properly located and function as required.
- **Streaming Content:** For streaming contents the tests should demonstrate that streaming data are up-to-date, properly displayed and can be suspended without error and restarted without difficulty.
- **CGI Scripts:** For CGI script, the black-box tests are conducted with an emphasis on data integrity (as data are passed to the CGI script) and script processing (once validated data have been received). In addition, performance testing can be conducted to ensure that the server-side configuration can accommodate the processing demands of multiple invocations of CGI scripts.

Semantics Interface Testing:

- Once each interface mechanism has been “unit” tested the focus of interface testing changes to a consideration of interface semantics.
- Interface semantics testing evaluates how well the design takes care of users, offers clear direction, delivers feedback, and maintains consistency of language and approach.

4. Usability Testing:

- Usability testing is used to verify how the WebApps are easy to use with.
- Usability testing evaluates the degree to which users can interact effectively with the WebApp and the degree to which the WebApp guides users’ actions, provides meaningful feedback, and enforces a consistent interaction approach.

- Usability testing can occur at a variety of different levels of abstraction as given below:
 - (i) The usability of a specific interface mechanism (e.g., a form) can be assessed.
 - (ii) The usability of a complete Web page (encompassing interface mechanisms, data objects, and related functions) can be evaluated.
 - (iii) The usability of the complete WebApp can be considered.
- To identify a set of usability categories and establish testing objectives for each category is the first step in usability testing.
- The following test categories and objectives (written in the form of a question) illustrate usability testing approach:
 - (i) **Layout:** Are navigation mechanisms, content, and functions placed in a manner that allows the user to find them quickly?
 - (ii) **Time Sensitivity:** Can important features, functions, and content be used or acquired in a timely manner?
 - (iii) **Interactivity:** Are interaction mechanisms (e.g., pull-down menus, buttons, and pointers) easy to understand and use?
 - (iv) **Readability:** Is text well written and understandable? Are graphic representations easy to understand?
 - (v) **Accessibility:** Is the WebApp accessible to people who have disabilities?
 - (vi) **Aesthetics:** Do layout, color, typeface, and related characteristics lead to ease of use? Do users “feel comfortable” with the look and feel of the WebApp?
 - (vii) **Personalization:** Does the WebApp tailor itself to the specific needs of different user categories or individual users?
 - (viii) **Display Characteristics:** Does the WebApp make optimal use of screen size and resolution?
- A series of tests is designed within each of above categories. In some cases, the “test” may be a visual review of a Web page. In other cases interface semantics tests may be executed again, but in this instance usability concerns are paramount.
- For example, we consider usability assessment for interaction and interface mechanisms. Larry L. Constantine, Lucy A.D. Lockwood suggest that the list of interface features should be reviewed and tested for usability: animation, buttons,
- color, control, dialogue, fields, forms, frames, graphics, labels, links, menus, messages, navigation, pages, selectors, text, and tool bars.
- As each feature is assessed, it is graded on a qualitative scale by the users who are doing the testing.
- Fig. 4.4 shows a possible set of assessment “grades” that can be selected by users and these grades are applied to each feature individually, to a complete Web page, or to the WebApp as a whole.

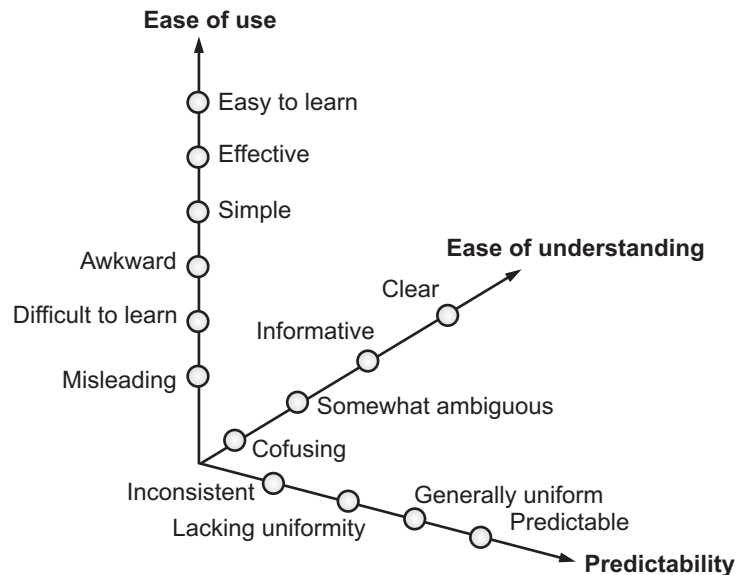


Fig. 4.4: Qualitative Assessment of Usability

5. Compatibility Testing:

- Compatibility tests ensure that the Web application displays correctly across different devices.
- This WebApp testing strives to uncover the following problems before the WebApp goes online:
 - (i) Different computers, display devices, operating systems, browsers, and network connection speeds can have a significant influence on WebApp operation.
 - (ii) Each computing configuration can result in differences in client-side processing speeds, display resolution, and connection speeds.
 - (iii) Operating system vagaries may cause WebApp processing issues.
 - (iv) Different browsers sometimes produce slightly different results, regardless of the degree of HTML standardization within the WebApp.
 - (v) Required plug-ins may or may not be readily available for a particular configuration.
- Steps in compatibility testing are given below:
 - (i) to define a set of “commonly encountered” client-side computing configurations and their variants. In essence, a tree structure is created, identifying each computing platform, typical display devices, the operating systems supported on the platform, the browsers available, likely Internet connection speeds, and similar information.
 - (ii) a series of compatibility validation tests are derived, often adapted from existing interface tests, navigation tests, performance tests, and security tests. The intent of these tests is to uncover errors or execution problems that can be traced to configuration differences.

6. Component-Level Testing:

- Component-level testing also called function testing. It focuses on a set of tests that attempt to uncover errors in WebApp functions.
- Each and every WebApp function is a software module (implemented in one of a variety of programming or scripting languages) and can be tested using black-box (and in some cases, white-box) techniques.
- Component-level test cases are often driven by forms-level input. Once forms data are defined, the user selects a button or other control mechanism to initiate execution.
- Equivalence Partitioning (EP), Boundary Value Analysis (BVA) and Path testing can be adapted for use in testing forms-based input and the functionality that is applied to it.
- In addition to component level test case design techniques, a technique called forced error testing is used to derive test cases that purposely drive the WebApp component into an error condition.
- The purpose is to uncover errors that occur during error handling (e.g., incorrect or nonexistent error messages, WebApp failure as a consequence of the error, erroneous output driven by erroneous input, side effects that are related to component processing).
- Each and every component level test case specifies all input values and the expected output to be provided by the component.
- The actual output generated/produced as a consequence of the test is recorded for future reference during support and maintenance.

7. Navigation Testing:

- Navigation tests analyze how users navigate through the Web application, given a specific task or goal.
- A user travels through a WebApp in much the same way as a visitor walks through a store. There are many pathways that can be taken, many stops that can be made, many things to learn and look at, activities to initiate, and decisions to make.
- This navigation process is predictable in the sense that every visitor has a set of objectives when he arrives.
- At the same time, the navigation process can be unpredictable because the visitor, influenced by something he sees or learns, may choose a path or initiate an action that is not typical for the original objective.
- The job of navigation testing is as follows:
 - to ensure that the mechanisms that allow the WebApp user to travel through the WebApp are all functional and
 - to validate that each Navigation Semantic Unit (NSU) can be achieved by the appropriate user category.

Testing Navigation Syntax:

- The first step of navigation testing in WebApp is actually begins during interface testing. Navigation are tested to ensure that each performs its intended function.
- Steven Splaine, Stefan P. Jaskiel suggest that each of the following navigation mechanisms should be tested:
 - links and anchors of all types,
 - redirects (when a user request a nonexistent URL),
 - bookmarks,
 - frames and frame sets,
 - site maps,
 - the accuracy of internal search facilities.
- Some of the tests noted can be performed by automated tools (such as link checking) while others are designed and executed manually. The intent throughout is to ensure that errors in navigation mechanics are found before the WebApp goes online.

Testing Navigation Semantics (NSU):

- It is a set of information and related navigation structures that collaborate in the fulfillment of a subset of related user requirements.
- Each and every NSU is defined by a set of navigation paths (called “ways of navigating”) that connect navigation nodes (e.g., Web pages, content objects or functionality).
- Taken as a whole, each NSU allows a user/client to achieve specific requirements defined by one or more use cases for a user category.
- Navigation testing exercises each NSU to ensure that these requirements can be achieved.
- We should answer the following questions as each NSU is tested:
 - If the NSU can be achieved using more than one navigation path, has every relevant path been tested?
 - Is the NSU achieved in its entirety without error?
 - Does the user understand his location within the content architecture as the NSU is executed?
 - If a function is to be executed at a node and the user chooses not to provide input, can the remainder of the NSU be completed?
 - Is every navigation node (defined for an NSU) reachable within the context of the navigation paths defined for the NSU?
- Like interface and usability testing the navigation testing, should be conducted by as many different constituencies as possible.
- We have responsibility for early stages of navigation testing, but later stages should be conducted by other project stakeholders, an independent testing team and ultimately, by nontechnical users. The intent is to exercise WebApp navigation thoroughly.

8. Configuration Testing:

- Configuration testing is the process of checking the operation of the WebApp with various types of hardware and software platforms and their different settings.
- Configuration instability and variability are the two important factors that make WebApp testing a challenge. Hardware, operating system(s), browsers, storage capacity, network communication speeds, and a variety of other client-side factors are difficult to predict for each user.
- In some cases, the configuration for a given user can change [e.g., Operating System (OS) updates, new ISP and connection speeds] on a regular basis.
- The result can be a client-side environment that is prone to errors that are both subtle and significant.
- One user's impression of the WebApp and the manner in which she interacts with it can differ significantly from another user's experience, if both users are not working within the same client-side configuration.
- The task of configuration testing is not to exercise or evaluate every possible client-side configuration.
- Configuration testing is to test a set of probable client/side and server/side configurations to ensure that the user experience will be the same on all of them and to isolate errors that may be specific to a particular configuration.

Server-Side Issues in Configuration Testing:

- On the server side, configuration test cases are designed to verify that the projected server configuration [i.e., WebApp server, database server, operating system(s), firewall software, concurrent applications] can support the WebApp without error.
- As server-side configuration tests are designed, you should consider each component of the server configuration.
- Among the questions that need to be asked and answered during server-side configuration testing are listed below:
 - Do server-side WebApp scripts execute properly?
 - Is the WebApp fully compatible with the server OS?
 - Have system administrator errors been examined for their effect on WebApp operations?
 - Are system files, directories, and related system data created correctly when the WebApp is operational?
 - Is the WebApp properly integrated with database software? Is the WebApp sensitive to different versions of database software?
 - Has the WebApp been tested with the distributed server configuration (if one exists) that has been chosen?

Client-Side Issues in Configuration Testing:

- On the client-side, configuration tests focus more heavily on WebApp compatibility with configurations that contain one or more permutations of the components like hardware, operating systems, browser software, User Interface (UI) components, plug-ins, and connectivity services (e.g., cable, DSL, WiFi).
- In addition to these components, other variables include networking software, the vagaries of the ISP, and applications running concurrently.
- To design client-side configuration tests, we must reduce the number of configuration variables to a manageable number. To accomplish this, each user category is assessed to determine the likely configurations to be encountered within the category.
- In addition, industry market share data may be used to predict the most likely combinations of components. The WebApp is then tested within these environments.

9. Security Testing:

- Security testing is performed to verify if the Web application is secured on Web as data theft and unauthorized access are more common issues.
- WebApp security is a complex subject that must be fully understood before effective security testing can be accomplished.
- WebApps and the client/side and server/side environments in which they are housed represent an attractive target for external hackers, disgruntled employees, dishonest competitors and anyone else who wishes to steal sensitive information, maliciously modify content, degrade performance, disable functionality, or embarrass a person, organization or business.
- Security tests are designed to probe vulnerabilities of the client-side environment, the network communications that occur as data are passed from client to server and back again, and the server-side environment.
- Each of these domains can be attacked, and it is the job of the security tester to uncover weaknesses that can be exploited by those with the intent to do so.
- On the client side, vulnerabilities can often be traced to preexisting bugs in browsers, e-mail programs, or communication software.
- On the server side, vulnerabilities include denial-of-service attacks and malicious scripts that can be passed along to the client side or used to disable server operations.
- In addition, server-side databases can be accessed without authorization (data theft). To protect against these (and many other) vulnerabilities, firewalls, authentication, encryption, and authorization techniques can be used.
- Security tests should be designed to probe each of these security technologies in an effort to uncover security holes.

- The design of security tests requires in-depth knowledge of the inner workings of each security element and a comprehensive understanding of a full range of networking technologies. In number of cases, security testing is outsourced to firms that specialize in these technologies.

10. Performance Testing:

- Performance testing is performed to verify the server response time and throughput under various load conditions.
- Performance testing is used to uncover performance problems that can result from a lack of server/side resources, inappropriate network bandwidth, inadequate database capabilities, faulty or weak operating system capabilities, poorly designed WebApp functionality, and other hardware or software issues that can lead to degraded client-server performance.
- The intent is twofold, to understand how the system responds as loading (i.e., number of users, number of transactions, or overall data volume) and to collect metrics that will lead to design modifications to improve performance.

Objectives Performance Testing:

- The performance testing checks speed, scalability and stability of a WebApp. The speed is determined through responsiveness of a WebApp i.e., how quickly it responds. The scalability determines maximum user's load that can handled by the WebApp. Stability tells about the stability of WebApp during varying loads.
- Performance tests in WebApp are designed to simulate real-world workloading situations or conditions such as the number of simultaneous WebApp users grows or the number of online transactions increases or the amount of data (downloaded or uploaded) increases, performance testing will help answer the following questions:
 - What system components are responsible for performance degradation?
 - What happens when loads that are greater than maximum server capacity are applied?
 - Does the server response time degrade to a point where it is noticeable and unacceptable?
 - What is the average response time for users under a variety of loading conditions?
 - Is WebApp reliability or accuracy affected as the load on the system grows?
- To develop answers to above questions, the two different performance tests are conducted namely, load testing (examines real-world loading at a variety of load levels and in a variety of combinations) and stress testing (forces loading to be increased to the breaking point to determine how much capacity the WebApp environment can handle).

Load Testing:

- Load testing is the simplest form of testing conducted to understand the behaviour of the system under a specific load.
- Load testing will result in measuring important business critical transactions and load on the database, application server, etc. are also monitored.
- The intent of load testing is to determine how the WebApp and its server-side environment will respond to various loading conditions.
-
- Load testing tells that how much load a WebApp can bear. As testing proceeds, permutations to the following variables define a set of test conditions:
 - N, number of concurrent users
 - T, number of online transactions per unit of time
 - D, data load processed by the server per transaction
- In every case, above variables are defined within normal operating bounds of the system. As each and every test condition is run, one or more of the measures are collected like average user response, average time to download a standardized unit of data, or average time to process a transaction.
- We should examine above measures to determine whether a precipitous decrease in performance can be traced to a specific combination of N, T and D.
- Load testing can also be used to assess recommended connection speeds for users of the WebApp. Overall throughput, P is computed in the following formula:
$$P = N \times T \times D$$
- For example, consider a popular sports news site. At a given moment, 20,000 concurrent users submit a request (a transaction, T) once every 2 minutes on average. Each transaction requires the WebApp to download a new article that averages 3K bytes in length. Therefore, throughput can be calculated as follows:
$$P = [20,000 \times 0.5 \times 3 \text{ Kbl}/60 = 500 \text{ Kbytes/sec.} = 4 \text{ megabits per sec.}$$
- The network connection for the server would therefore have to support this data rate/speed and should be tested to ensure that it does.

Stress Testing:

- Stress testing is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.
- As its name stress testing involves testing an WebApp under extreme stress and pressure to figure out how it handles excess traffic or data processing.
- Stress testing is a continuation of load testing, but in this instance the variables, N, T, and D are forced to meet and then exceed operational limits.

- The intent of stress tests is to answer each of the following questions:
 - Is data integrity affected as capacity is exceeded?
 - Does the system degrade “gently,” or does the server shut down as capacity is exceeded?
 - Does the server queue resource requests and empty the queue once capacity demands diminish?
 - What values of N, T and D force the server environment to fail? How does failure manifest itself? Are automated notifications sent to technical support staff at the server site?
 - Are transactions lost as capacity is exceeded?
 - Does server software generate “server not available” messages? More generally, are users aware that they cannot reach the server?
 - If the system does fail, how long will it take to come back online?
- A variation of stress testing is sometimes referred to as spike testing/bounce testing. In spike testing regime, load is spiked to capacity, then lowered quickly to normal operating conditions, and then spiked again.
- The spike testing, tests the WebApp’s reaction to sudden large spikes in the load generated by users.
- By bouncing system loading, we can determine how well the server can marshal resources to meet very high demand and then release them when normal conditions reappear (so that they are ready for the next spike).
- Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system.
- The objective of spike testing is to determine whether the system will be able to sustain the work load.

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. Which is a software practice that ensures quality by testing that the functionality of a given Web application is working as intended or as per the requirements?
 - (a) Web application testing
 - (b) Web functional testing
 - (c) Web security testing
 - (d) Web interface testing
2. Which type of testing ensures that the Web application is working as intended?
 - (a) application testing
 - (b) functional testing
 - (c) security testing
 - (d) interface testing
3. Which software testing ensures that all interactions between the web server and application server interfaces are running smoothly?
 - (a) Usability
 - (b) Compatibility
 - (c) Interface
 - (d) Performance

4. Quality dimensions for Web application testing includes,
 - (a) Structure and Content
 - (b) Usability and Compatibility
 - (c) Interoperability and Navigability
 - (d) All of the mentioned
5. Which testing ensures that the Web application testing makes sure that your application is protected against unauthorized access and harmful actions through viruses or other malicious software?
 - (a) security
 - (b) stress
 - (c) interface
 - (d) database
6. Which testing ensuring that the Web application's functionality is working properly on all browsers and devices, it is time to take a look at how it performs under heavy load?
 - (a) Usability
 - (b) Compatibility
 - (c) Interface
 - (d) Performance
7. Which testing ensuring the Web application is compatible with all browsers, operating systems and mobile devices and so on?
 - (a) Usability
 - (b) Compatibility
 - (c) Interface
 - (d) Performance
8. The Web application testing _____ describes the approach that will be used to manage the overall Web testing process.
 - (a) plan
 - (b) teplate
 - (c) strategy
 - (d) document
9. Errors encountered in WebApp because of,
 - (a) incorrect design or improper HTML for other programming language coding
 - (b) implementation of user interface on a specific browser
 - (c) different configurations and environments for WebApp development
 - (d) All of the mentioned
10. Which is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product?
 - (a) Test Plan
 - (b) Test Strategy
 - (c) Test Template
 - (d) None of the mentioned
11. Which testing analyze how users navigate through the Web application, given a specific task
 - (a) Load
 - (b) Navigation
 - (c) configuration
 - (d) Interface

Answers

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

Q. II Fill in the Blanks:

1. _____ application testing is to check a web application for potential bugs either before it moves to the production environment or once it is live on the web and accessible to end users.
2. A _____ documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements.
3. _____ testing is performed to verify if the application is secured on web as data theft and unauthorized access.
4. A _____ is a small piece of information sent from a website and stored on the users in the user's hard drive (in a text file) by the user's web browser. Testing of cookie is the process of verifying whether the cookies are working as intended or not.
5. A web _____ (or web app) is application software that runs on a web server
6. _____ testing is to test the interface between the web server and application server, application server and database server have proper interaction or not.
7. The Web application testing a software testing technique to test the functionality, usability, accessibility, compatibility, performance and security of the application which is hosted on the _____.
8. _____ testing involves verifying the integrity of data in the front end with the data present in the back end.
9. _____ testing is to ensure whether an application is compatible across different browsers and on a variety of devices.
10. A web application _____ in a web browser, rather than being installed on a user's device and runs on any device that can access the internet which includes desktop computers, tablets, and mobile phones.
11. Web application _____, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested.
12. Navigation testing analyze how users _____ through the Web application, given a specific action.

Answers

1. Web	2. test plan	3. Security	4. cookies
5. application	6. Interface	7. Web	8. Database
9. Compatibility	10. runs	11. testing	12. navigate

Q. III State True or False:

1. Web testing is software testing that focuses on web applications and also known as Web Testing or Website Testing.
2. Component-level testing in Web testing focuses on a set of tests that attempt to uncover errors in WebApp functions.
3. Compatibility testing confirms the website design is compatible across different browsers and also on a variety of devices.
4. Interface testing includes checking the communication processes as well as making sure that error messages are displayed correctly.
5. Test planning helps us determine the effort needed to validate the quality of the application under test.
6. The computing environment for Web applications is simple as internet is heterogeneous, distributed, multi-platform, multimedia, multilingual and cooperative wide area network.
7. A test plan is a document detailing the objectives, resources, and processes for a specific test for a software or hardware product. The plan typically contains a detailed understanding of the eventual workflow.
8. Web application testing ensures that an application is fully functional and running properly and securely.
9. Web applications are vulnerable to an array of attacks, making security testing essential.
10. A test plan is a document describing the scope, approach, resources, and schedule of intended test activities.

Answers

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

Q. IV Answer the following Questions:**(A) Short Answer Questions:**

1. Define Web application.
2. What is the purpose of
3. Define Web application testing.
4. Define test plan.
5. List dimensions of quality.
6. Define test planning.
7. Define strategy for Web applications.

(B) Long Answer Questions:

1. What is Web application testing? How to perform it? Explain in detail.
2. What is Web application? How it works? Explain diagrammatically.

3. What are the techniques used for Web application testing? Describe two of them in detail.
4. Write a short note on: Dimension of quality.
5. What is testing strategy for Web application? Describe in detail.
6. What is test planning? Explain test plan in detail.
7. Describe the term errors within a WebApp environment.
8. With the help of diagram describe testing process.
9. What is interface testing? List mechanisms for testing interface.
10. What is navigation testing? How to test navigation syntax and semantics?
11. Describe security and performance testing for Web application testing? Also compare them.
12. Explain content testing in Web testing in detail.

■■■

Agile Testing

Objectives...

- To understand Concept of Agile Testing
 - To learn Agile Principles and Values
-

5.0 INTRODUCTION

- Agile testing is a software testing practice that follows the principles of agile software development. Agile is the ability to create and respond to change.
- Agile testing involves all members of a cross-functional agile team, with special expertise contributed by testers, to ensure delivering the business value desired by the customer at frequent intervals, working at a sustainable pace.
- Agile testing is software testing that follows the best practices of Agile development. Agile development encourages us to solve our problems as a team.
- In agile testing, the word "Agile" primarily signifies something that can be performed quickly and immediately, but also in the area of software development.
- The agile testing means testing within the context of an Agile process such as Scrum and/or Extreme Programming (XP). The core-functional agile team implements/creates it in order to test the software product and its several modules.
- The implementation of agile testing makes sure to deliver a high quality software product as defects get detected in the initial stage of the project itself.
- An agile tester is a professional tester who embraces change, collaborates well with both technical and business people, and understands the concept of using tests to document requirements and drive development.
- In agile testing, the agile testers tend to have good technical skills, know how to collaborate with others to automate tests, and are also experienced exploratory testers.

5.1 AGILE TESTING

- Agile is an iterative development methodology, where both development and testing activities are concurrent.
-

- The Agile Manifesto was published by a team of software developers in 2001, highlighting the importance of the development team, accommodating changing requirements and customer involvement.
- Testing is very critical from agile perspective. Agile testing is a software testing practice that follows the principles of agile software development. Agile development talks about agile manifesto which works some principles.
- Agile development promotes iterative and incremental development with significant testing i.e., customer centric and welcomes change during the process.
- In other words, agile testing is a testing practice for projects using an agile methodology. Agile methodology focus on getting the software product developed with only the activities that are required.
- Agile methodology is a collection of values, principles and practices that incorporates iterative development, test and feedback into a new style of software application development.
- Agile testing involves all members of the project team, with special expertise contributed by testers.
- Testing is not a separate phase and is interwoven with all the development phases such as requirements, design and coding and test case generation. Testing takes place simultaneously through the development life cycle.
- A software testing practice that follows the principles of agile software development is called Agile Testing.
- Agile is an iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.

Features of Agile Testing:

1. The whole development process in agile testing is divided into smaller sprints, or they are called iterations that are delivered to the customers within the given timeframe.
2. Agile testing suggests smart work rather than hard work, hence facilitating the continuous delivery of products in a most efficient manner as each team works in a collaborative manner to accomplish the task.
3. Feedback is provided on an ongoing basis by the testing team to ensure continuous progress.
4. Testing is done during the development cycle to ensure that the deliverable should be in a stable state so that the tester can test major functionality with different perspectives.
5. All the bugs or issues which are recognized in one iteration are corrected by the agile team within that iteration itself. This simplifies the task of testing and fixing defects.

6. It promotes an efficient relationship between the testers and developers to gain the desired fruits. This also enhances the capability of individuals to adopt the change required to embrace work processes.
 7. Instead of being sequential like traditional techniques, agile testing is a continuous mechanism.
- Agile methodology allows organizing and managing the software development process. Agile methodology breaks the whole process into shorter tasks so that frequent assessment and adaptation of plans can result in high-quality deliverables.
 - Agile helps in mitigating the chances of failure or error by shifting the focus of development from less important factors to more important factors.
 - Agile Manifesto focuses on having collaboration within the teams to ensure the smooth execution of the product.

Methods of Agile Testing:**1. Acceptance Test-Driven Development (ATDD):**

- The ATDD is the methodology of agile testing, in which tests work on the basis of customers' point of view on how a software work shall.
- So, these acceptance tests exhibit the notions of the users so it ensures that the software works according to the demand for which it was created.
- The ATDD approach emphasizes the customer's requirements by involving the team members with different viewpoints.
- The team's members of development, testing, and the customers come together in order to develop the acceptance test from the customer's perspective.
- ATDD is a very customer-centered methodology. In ATDD the code is acquired along with the developed acceptance test case.
- The objective of using the ATDD methodology is to develop a program based on the user's view.

2. Exploratory Testing:

- Exploratory Testing is an important activity in an agile environment as it can help software testers to keep up with the rapid development pace of agile software projects.
- In exploratory testing, test design and test execution are done simultaneously. In exploratory testing, a tester does every hit and trial to break the system by using different user behaviors.
- In exploratory testing, no detailed documentation is given to the tester. They focus on the high-risk scenarios based on their experience.
- Exploratory testing is an testing approach to testing that values the tester as an integral part of the test process and shares the same values as the Agile Manifesto:
 - Individuals and interactions over processes and tools.
 - Responding to change over following a plan.
 - Working software over comprehensive documentation.

3. Risk-based Testing:

- The risk events could have occurred in the past or may be a concern for future occurrences. Risk-based testing is a software testing technique based on the probability of risk.
- Risk is the occurrence of an uncertain/ unforeseen event with a positive or negative effect/impact on the measurable success criteria of a software project/product.
- The risk events could have occurred in the past or may be a concern for future occurrences. Risks can serve as a reliable parameter to plan, schedule, and allocate tester effort in agile.
- The risk-based testing technique organizes testing efforts in ways that lower the residual level of product risk when the software goes into production.
- This risk-based testing strategy is useful for test analysis, planning, estimation, design, execution, and results reporting.
- Under risk-based testing technique tasks are prioritized based on the risks. The more critical areas which are prone to greater risks are tested and rectified first.
- Any failures in such risk-based testing areas may lead to heavy losses or complicated problems like server crashes.
- The comparatively less critical or smaller areas are kept for the last, even if there is any error or glitch in such areas, the losses are nominal and problems can be corrected with ease.

4. FIT Tests:

- As the name suggests, FIT (Framework Integrated Test) technique is an integration of the tasks of analysts, testers, developers, and even customers.
- When FIT technique is used, the result of the testing comes in three colors, red, yellow or green which depicts the levels of quality of the software.

5. Dynamic Software Development Method (DSDM) Agile Testing:

- DSDM is an iterative agile software development methodology based on based on the Rapid Application Development (RAD) methodology.
- DSDM is the effective method of agile testing. DSDM is a RAD approach that offers a delivery framework to agile projects.
- In other words, the DSDM technique is a correlate degree agile code development approach, which gives a framework for developing and maintaining systems.
- The DSDM testing approach can be used by users, development, and testing teams.

6. Behavior Driven Development (BDD):

- The purpose of BDD testing is to ensure that the software product or system that is built is working as expected. The BDD testing is conducted on the basis of how the system is supposed to work.

- The business/organizational analysts and testers communicate to know and understand each other's stake prior to the development process and design the software accordingly.
- Test scenarios in BDD testing technique are written in a format the Gherkin Given/When/Then syntax.
- The documentation of the scenarios in BDD testing helps to build tests that can be failed on the initial stage so that further they can build the software functionality that makes those scenarios pass.

7. **Session-Based Testing:**

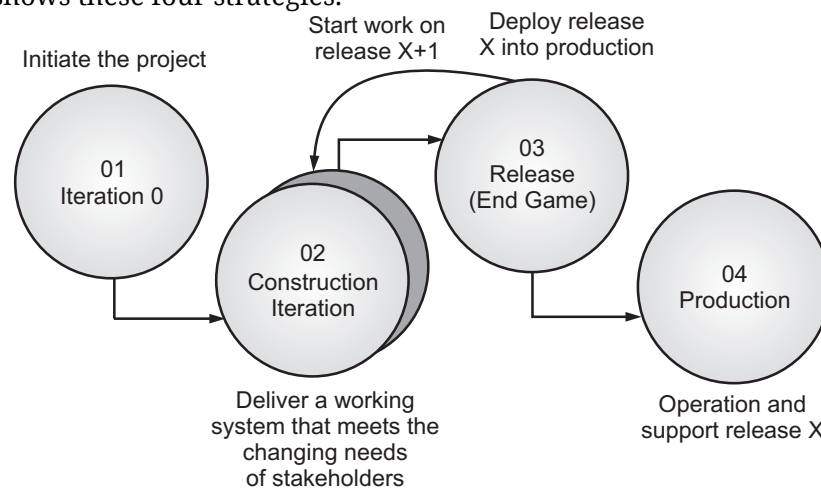
- Session-based testing was developed by Jonathan and James Bach in 2000. A session is a basic testing work unit (an uninterrupted block of reviewable, chartered test effort that lasts for a fixed time duration).
- Session-based testing is a software testing technique aims to provide rapid defect discovery, creative on-the-fly test design, management control and metrics reporting.
- The session-based testing is mainly is created on the values of exploratory testing. Though session-based testing contains some structure and on the other hand, exploratory testing is performed unexpectedly without any planning.
- The session-based testing is used to help us identify the hidden bugs/errors and defects in the particular software.

8. **Extreme Programming (XP):**

- XP is an agile methodology that supports frequent releases in short development cycles to improve software quality and allow developers to respond to changing customer requirements.
- XP is a software development methodology which is intended to improve software product quality and responsiveness to changing customer requirements.
- When there is a continuous modification in the user requirements, we will go for the XP methodology.
- The XP will helps to deliver a quality software product, which meets the customer's requirements that are made clear throughout the process of development and testing.
- XP teams in practice Test-Driven Development (TDD) technique that entails writing an automated unit test before the code itself.
- According to XP agile methodology, every piece of code must pass the test to be released. So, software engineers thereby focus on writing code that can accomplish the needed function. That's the way TDD allows programmers to use immediate feedback to produce reliable software.

Agile Testing Strategies:

- Agile testing has four different approaches (Iteration 0, Construction Iteration, Release End Game or Transition Phase and Production) which help us to enhance our product quality.
- Fig. 5.1 shows these four strategies.

**Fig. 5.1**

- Let us discuss the agile testing strategies one by one in detail:

1. Iteration 0:

- The iteration 0 is the first strategy of agile testing in which we execute the preliminary setup tasks such as finding people for testing, establishing testing tools, preparing resources or usability testing lab, etc.
- In iteration 0 strategy of agile testing, the below steps are accomplished:
 - Verifying a business case for the project and boundary situations, and the project scope.
 - Summarize the important requirements and use cases that will determine the strategic trade-offs.
 - Plan the initial project and cost valuation.
 - Detecting the risk.
 - Outline one or more candidate designs.

2. Construction Iteration:

- In construction iteration agile testing strategy, the majority of the agile testing is performed.
- The construction iteration is performed as a set of iterations on software product in order to create an increment of the solution.
- In construction iteration agile testing strategy, the agile team follows the listed requirement within each iteration where they can acquire the most significant business needs or requirements left behind from the work item stack and then execute them.

- The construction iteration process divided into two types of testing namely Confirmatory Testing and Investigative Testing.

Confirmatory Testing: This testing ensure that the software product meets all the stakeholders' requirements, we will execute the confirmatory testing. Confirmatory testing can be divided into two types of testing namely, Agile Acceptance Testing and Developer Testing.

- **Agile Acceptance Testing:** It is a combination of functional testing and acceptance testing. The agile acceptance testing can be executed by the development team and stakeholders together.
- **Developer Testing:** It is a combination of unit testing and integration testing. And it validates both the application code as well as the database schema.

Investigative Testing: In order to test deep and identify all the issues, which are overlooked in confirmatory testing, we will execute the investigative testing.

3. Release End Game or Transition Phase:

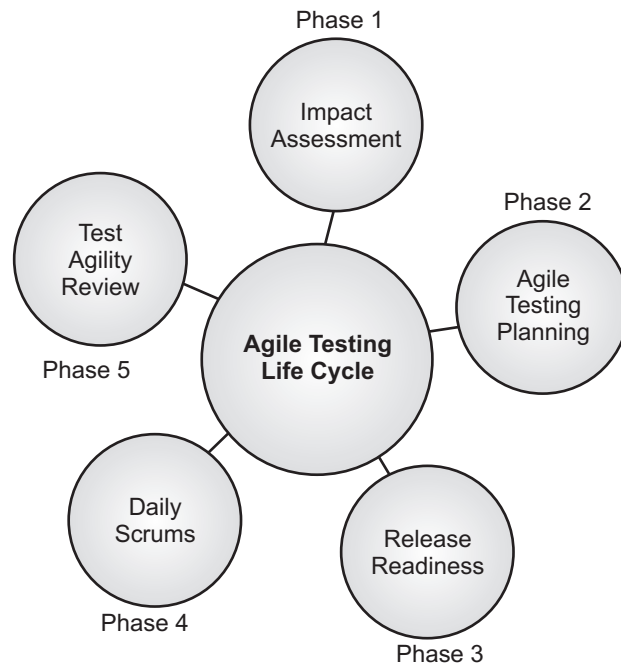
- The third strategy of agile testing is release. The objective of this strategy is to implement the software product or system effectively in production.
- The tester will be working on its defect stories in the end game. In the release end game or transition stage, we have the following activities:
 - Support Individuals.
 - Training of end-users.
 - Operational People.
- Similarly, it involves some additional activities as well:
 - Back-up and Restoration.
 - Marketing of the product release.
 - User documentation.
 - Completion of system.
- The last/final agile methodology testing stage encompasses whole system testing and acceptance testing. To complete the final testing phase without having any difficulties, we should have to test the product more thoroughly in construction iterations.

4. Production:

- The software product or system will move on to the production stage as soon as the release stage is completed.

Agile Testing Life cycle:

- Agile testing life cycle completed into five different phases as shown in Fig. 5.2.

**Fig. 5.2**

- Let understand all the phases in Agile testing life cycle detail:

Phase 1: Impact Assessment: In impact assessment phase, we collect the inputs and responses from users and stakeholders to execute the impact assessment phase. The impact assessment phase is also known as the feedback phase, which supports the test engineers to set the purpose for the next life cycle.

Phase 2: Agile Testing Planning: In agile testing planning phase, the developers, test engineers, stakeholders, customers, and end-users team up to plan the testing process schedules, regular meetings, and deliverables.

Phase 3: Release Readiness: In release readiness phase, where test engineers have to review the features which have been created entirely and test if they are ready to go live or not and which ones need to go back to the previous development phase.

Phase 4: Daily Scrums: Daily scrums phase, involves the daily morning meetings to check on testing and determine the objectives for the day. And in order to help the test engineers to understand the status of testing, the goals and targets of the day are set daily.

Phase 5: Test Agility Review: In test agility review phase, the test agility phase encompasses the weekly meetings with the stakeholders to evaluate and assess the progress against goals. In simple words, we can say that the agility reviews are implemented regularly in the development process to analyze the progress of the development.

Advantages of Agile Testing:

1. Agile testing gives way to get regular feedback and reviews directly from the end-user, which helps us enhance the software product's quality.
2. The agile testing will save lots of time and money, making the estimation of cost more transparent.
3. Agile testing reduces documentation or it requires less documentation to execute the agile testing.
4. Agile software testing is reducing errors/bugs/defects and enhancing software productivity.
5. Agile testing quick responses to changing requirements and accommodating them soon.
6. Agile testing is flexible to accommodate changes and advancements in technology.

Disadvantages of Agile Testing:

1. In agile testing, the changes are not always foreseeable, so they may not be adopted while releasing the next iteration. In this way, sometimes it becomes unpredictable for the users with regards to what new will come up in the next iteration.
2. Due to continuous change in requirements sometimes agile testing becomes difficult and complex to assess the actual effort required performing the specific task.
3. The repetitive release of parts in agile testing of the software product or system incurs higher expenses.

5.2**DIFFERENCE BETWEEN TRADITIONAL AND AGILE TESTING**

- Traditional software testing development uses a phased approach (e.g. requirement elicitation, specification, coding, testing and release).
- Testing takes place at the end of the software development, shortly before the release (shown schematically in the upper part of Fig. 5.3).
- Fig. 5.3 describes an ideal situation because it gives the impression that there is just as much time for testing as for coding. However, this is not the case with many software developments projects.
- Testing is 'squeezed' because software coding takes longer than expected and the teams end up going through a code-and-fix cycle.
- Traditional testing was the mainstream, but efficiency increases when an enterprise makes a shift from traditional to agile testing.
- Traditional testing aims to understand user needs and develop a software product. After development, testers test the software product and report bugs before deployment.

- The development team in traditional testing then works on them and fixes any errors using the best possible solution.
- Traditional testing works on the assumption that the processes are repetitive and predictable. In contrast, Agile testing seeks to correct the rigidity of traditional testing. It is a team-based approach that, unlike traditional testing, is interactive and dynamic.
- As a result, the delivery time shortens. Agile testing is iterative and incremental. This means that the testers test each code increment as soon as it is complete.
- Iteration can only take a week or a month. The team builds and tests a bit of code to make sure it is working properly, and then proceeds to the next part that needs to be created.
- Agile software development is shown in the lower part of Fig. 5.3. The label 'A', 'B', 'C', 'D', 'E' and 'F' represent block or unit of code in a software system.
- In comparison, in the Waterfall model, the testing process is more structured and there is a detailed description of the testing phase.
- Agile testing is well suited for small projects. On the other hand, Waterfall testing can be adopted for all sorts of projects.
- As testing begins at the start of the Agile project, errors can be fixed in the middle of the project.
- In the waterfall model, however, the product is tested at the end of the development. For any changes, testing must start from the beginning.
- We start by looking at similarities between agile testing and testing in traditional software development for this consider Fig. 5.3.
- In the phased traditional approach (See Fig. 5.3), it is clear that testing happens at the end, right before release. The diagram is idealistic, because it gives the impression there is as much time for testing as there is for coding.
- In number of projects, this is not the case. The testing gets 'squished' because coding takes longer than expected, and because teams get into a code-and-fix cycle at the end.
- Agile testing is iterative and incremental means that the testers test each increment of coding as soon as it is finished.
- Iteration might be as short as one week, or as long as a month. The team builds and tests a little bit of code, making sure it works correctly, and then moves on to next piece that needs to be built.
- Software programmers never get ahead of the testers, because a story is not 'done' until it has been tested.

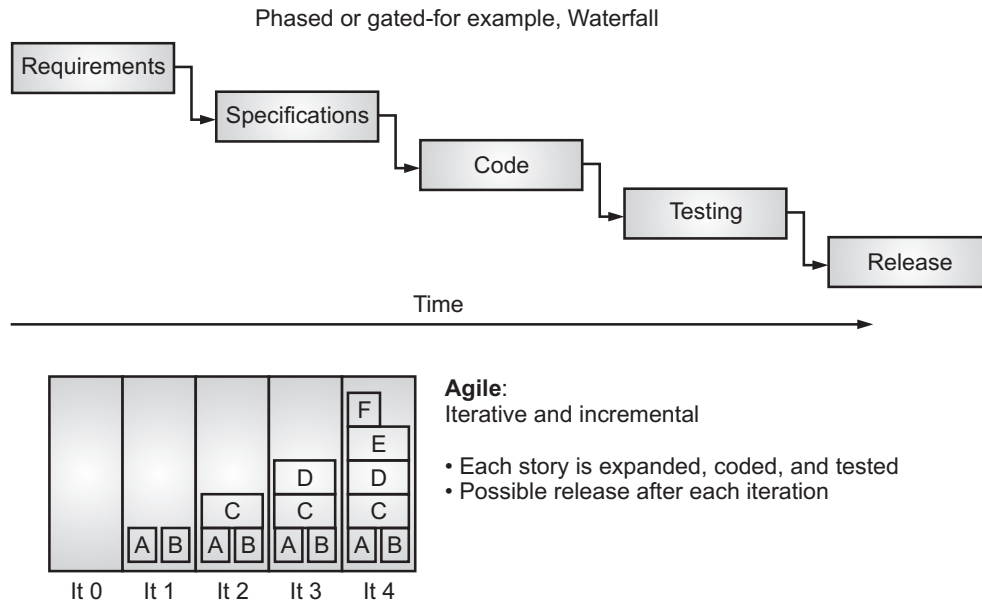


Fig. 5.3: Traditional Testing vs Agile Testing

- Unlike the traditional testing (Waterfall model), Agile testing can create or implement at the beginning of the project with endless incorporation between development and testing. It is not a sequential but the continuous process.
- The agile testing process is a smart way of testing complicated software, which accepts more effective results as compared to the traditional testing procedures.
- Some key differences between Traditional Testing and Agile Testing are given below:
 1. In traditional testing, there is no user feedback taken until testing is done. The agile approach follows short ongoing feedback cycles at the end of every sprint.
 2. The primary function of the traditional testing approach is to certify the quality of the products. The agile testing principles ensure the product's quality and fast delivery with minimal functionalities.
 3. In the traditional testing, required modifications are only done in the next release. While in the agile testing, process follows a continual improvement in software testing, where changes required are done in the next sprint of the testing cycle.
 4. Agile testing emphasizes on positive desire to implement a solution that passes the test, confirmation of user requirements while traditional testing focus on negative desire to break the solution, falsification of given solution.
- Here, are some of the other reasons why software testing done in an Agile environment is preferred over testing in a traditional setting:
 - **Transparency and Continuous Testing:** Agile testing technique teams perform tests regularly to make sure that the product is continuously progressing. Further, in this case, traditional testing is done in conjunction with development to bring in greater transparency in the process.

- **Faster Time to Market and Quick Product Releases:** The incremental and iterative model used in the agile testing approach minimizes the overall time taken between specifying test requirements and validating results. It leads to faster product releases without any delay.
- **Scope for Feedback:** In the Agile testing, the business team participates in each iteration. This kind of ongoing feedback helps to reduce the time taken to get feedback on software development work.
- **High-level Software Quality:** The Agile testing approach ensures that the teams test the software so that the code is clean and tight. Additionally, the software's regular testing allows for all the issues and vulnerabilities to be detected quickly and fixed in the same iteration as they are being developed.
- **Accountability and Tighter Alignment:** Agile testing is well-known for fixing defects instantly due to the teams of software testers and developers working collaboratively with each other, enabling them to share immediate feedback.
- **Better Collaboration:** With a strong team of developers, testers, architects, and coders working closely in the agile testing methodology, there is more face to face communication throughout the entire software testing life cycle. It eliminates the need for lengthy documentation processes leading to faster and quicker test results.

5.3 AGILE PRINCIPLES AND VALUES

- Agile values and principles in agile testing promote a focus on the people involved in a project and how they interact and communicate.
- An agile team that guides itself with agile values and principles will have higher team morale and better velocity than a poorly functioning team of talented individuals.

5.3.1 Agile Principles

- The Agile Manifesto also includes a list of principles that define how we approach software development. Our list of agile “testing” principles is partially derived from those principles.

Principle 1 - Testing is Not a Phase: Agile team tests alongside the development team to ensure that the features implemented during a given iteration are actually done. Testing is not kept for a later phase.

Principle 2 - Testing Moves the project Forward: Agile team tests continuously and continuous testing is the only way to ensure continuous progress. Agile testing provides feedback on an ongoing basis and the product meets the business demands/needs.

Principle 3 - Everyone Tests: In agile testing, the entire team including analysts, developers, and testers test the application. In conventional SDLC, only test team tests while in agile including developers and BA's test the application.

Principle 4 - Shortening Feedback Response Time: In Agile testing, the business team gets to know the product development for each and every iteration. They are involved in every iteration. Continuous feedback shortens the feedback response time and thus the cost involved in fixing it is less.

Principle 5 - Keep the Clean Code: Raised defects in agile testing are fixed within the same iteration and thereby keeping the code clean.

Principle 6 - Reduce Test Documentation: Instead of very lengthy documentation, agile testers use reusable checklist, focus on the essence of the test rather than the incidental details. The execution of agile testing requires less documentation as the agile teams or all the test engineers use a reusable specification or a checklist and the team emphasizes the test rather than the secondary information.

Principle 7 - Test Driven: In conventional approaches, testing is performed after implementation while in agile testing, testing is done while implementation. While performing the agile testing, we need to execute the testing process during the implementation that helps to decrease the development time. However, the testing is implemented after implementation or when the software is developed in the traditional testing process.

5.3.2 Agile Values

- Agile testing follows the standards of agile software development. The agile software development is an iterative framework for development and delivery of software products.
- Agile is based on ideas of using small teams in increments; delivering quality software products. The goal of agile testing is to do what is necessary to complete customer's requirements or expectations.
- In Agile, testing is integrated directly into the development process so that defects/bugs are discovered as early and as often as possible.
- As a result, agile testers can identify problems at every point in the software development process, moving the product quickly towards release.
- Agile testers in agile testing ask questions of both customers and developers early and often and help shape the answers into the right tests.
- The agile test plan contains those types of testing executed in a specific iteration, such as test environments, test data requirements, test results and infrastructure.
- The agile values as listed in Agile Manifesto are as follows:
 1. **Working Software over Comprehensive Document:** Working software product or system is valued more useful than presentation documents in client meetings. Self-explanatory code is valued more than hundreds of pages of documents.

2. **Individuals and Interactions over Processes and Tools:** Agile software development methodology lays emphasis on self-organization, motivation and interactions such as co-location and pair programming as compared to any process or tools.
3. **Responding to Change over Following a Plan:** Agile development focuses on quick response to change to minimize delivery to customer time.
4. **Customer Collaboration over Contract Negotiation:** Agile methodology accepts the fact that with the ever-changing world change to software development requirements is a continuous process and all requirements cannot be fully collected at the beginning of the software project. Therefore, continuous customer and stakeholder interaction is must at short periodic time intervals.

5.4 AGILE TESTING QUADRANTS

- The quadrants in agile testing, separate the whole process in four Quadrants and help to understand how agile testing is performed.
- Agile methodologies treat testing from a different perspective. Since agile projects are highly dynamic in nature and composed by short iterations, testing in these environments aims to provide information to drive planning for the following iterations and to increase the confidence of the development team in the value added by the product.
- Lisa Crispin, Janet Gregory developed, as an improvement to Marick's work, a classification model for the most commonly techniques for software testing for the most commonly techniques for software testing in agile development projects called Agile Testing Quadrants.
- Fig. 5.4 shows the agile testing quadrants that shows how each of the four quadrants reflects the different reasons we test.
- On one axis, we divide the matrix into tests that support the team and tests that critique the product. The other axis divides them into business-facing and technology-facing tests.
- The order in which we have numbered these quadrants has no relationship to when the different types of testing are done.
- The quadrants on the left side include tests that support the team as it develops the software product.
- This concept of testing to help the programmers is new to many testers and is the biggest difference between testing on a traditional project and testing on an agile project.
- The testing done in Quadrant 1 (Q1) and Quadrant 2 (Q2) are more requirements specification and design aids than what we typically think of as testing.

- The agile testing quadrants, based on original work by Brian Marick. The quadrants in agile testing are shown in Fig. 5.4.
- Fig. 5.4 shows a 2×2 box diagram with the X-axis representing the purpose of the tests (from supporting the team to critiquing the product) and the Y-axis representing whom the test is targeting (from technology facing to business facing).
- The resulting quadrants within the diagram are labeled Q1 to Q4, and no ordering is implied with this numbering system; this is purely for reference.
- The Quadrant 1 (Q1) is located in the position that is strongly supporting the team and technology facing, so the tests falling within this quadrant are unit and component tests.
- These types of tests can act as scaffolding around which the development team creates the software. They can also shape the design and the architecture.
- For example, by using TDD, you can ensure that there are also two consumers of functionality namely, the original consumer component within the application and a test.
- Tests falling in Quadrant 1 (Q1) are highly automatable. We should be running these tests not only within a build pipeline, but also as part of minute-by-minute local builds and, ideally, through an automated process that watches for code changes and runs appropriate tests.
- Infinitest is one such example of a continuous testing tool that operates as a plugin for Eclipse and IntelliJ.
- Quadrant 2 (Q2) is also strongly supporting the agile team, but is oriented toward being business- or customer-facing.
- Tests that fall into this category include functional tests, examples, and story tests. Tests within this quadrant are often referred to as acceptance tests, and are a focus of Specification by Example or BDD.
- Quadrant 3 (Q3), at the top right-corner, is also business facing, but switches the purpose toward critiquing the product.
- In Quadrant 3 (Q3), we try to explore how the end user will feel when using the product. Is it appealing? Is it intuitive? Is it accessible by all types of users and devices?
- This kind of test cannot be easily automated, since the expected right answer is not always known before the test.
- However, this does not mean that the tests aren't important, since failing to address these questions can lead to the product's failure.
- Quadrant 4 (Q4) deals with critiquing the product from a technical point of view. These tests are usually difficult to write and tend to require special tools.
- Also, although their execution can be automated, their evaluation is a little more subjective.

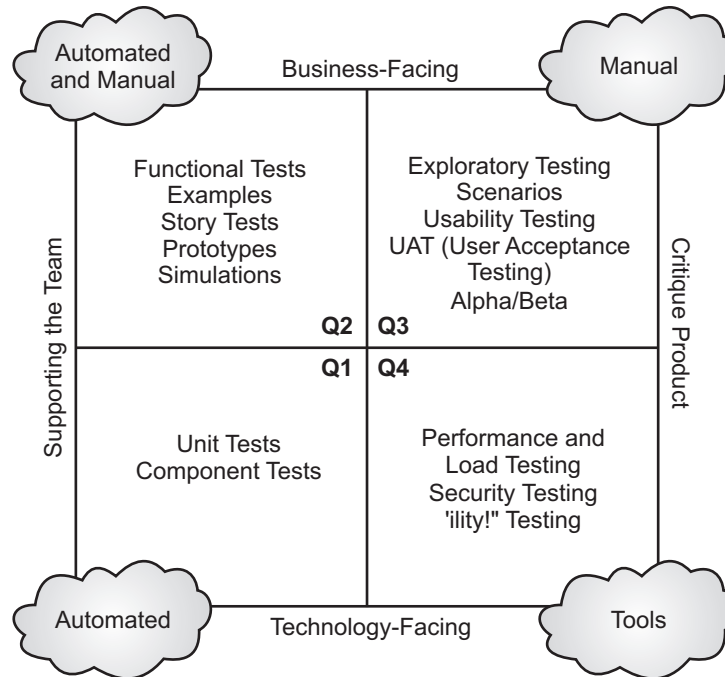


Fig. 5.4: Quadrants in Agile Testing

- Let us discuss the four Quadrants in the process of agile testing:

Quadrant 1 (Automated):

- In agile testing, the lower left quadrant represents test-driven development, which is a core agile development practice. The internal code quality is the main focus in Quadrant Q1.
- The Quadrant 1 i.e., Q1 is the automated quadrant contains tests that are designed to improve the code of the product being created.
- In the first quadrant Q1 of Agile testing, we will see mainly emphasis on the quality of the code. We can say internal code quality, which contains the test cases and test components that is executed by the test engineers.
- And these test cases in the Quadrant 1 (Q1) are technology-driven and used for automation testing in order to enhance the code and support the testing team to perform their tasks.
- All through the Quadrant 1 (Q1) of agile testing, we can execute the Unit Testing and Component Testing.
- Unit tests verify functionality of a small subset of the system, such as an object or method. Component tests verify the behavior of a larger part of the system, such as a group of classes that provide some service.
- Both, unit tests and component tests types of tests are usually automated with a member of the xUnit family of test automation tools.

- A major purpose of Quadrant 1 (Q1) tests is Test-Driven Development (TDD) or Test-Driven Design. The process of writing tests first helps programmers design their code well.
- These tests let the programmers/testers/developers confidently write code to deliver a story's features without worrying about making unintended changes to the system.
- They can verify that their design and architecture decisions are appropriate. Unit and component tests are automated and written in the same programming language as the software application or product.
- A business/organizational expert probably couldn't understand them by reading them directly, but these tests aren't intended for customer use.
- In fact, internal quality isn't negotiated with the customer; it's defined by the programmers.
- Software programmer tests are normally part of an automated process that runs with every code check-in, giving the team instant, continual feedback about their internal quality.

Quadrant 2 (Automated and Manual):

- In agile testing, the Quadrant 2 (Q2) contains test cases that are business driven and are implemented to support the team.
- In agile testing, the Quadrant Q2, we will see mainly emphasis on the customer requirements given to the team before and throughout the testing procedures, which expands the business results of the newly created software.
- The tests in Quadrant Q2 in agile testing are also support the work of the development team, but at a higher level.
- These business-facing agile tests also known as customer-facing tests and customer tests, define external quality and the features that the customers want.
- The test case involved in Quadrant Q2 of agile testing is business-driven, usually manual and automated functional tests, prototypes and examples of test scenarios performed by the testing team.
- In Quadrant 2 (Q2) of agile testing, we can execute the following tests:
 - Testing scenarios which may occur and workflow
 - Implementing the pair testing
 - Testing the user story and experiences like prototypes.

Quadrant 3 (Manual):

- In agile testing, Quadrant 3 (Q3) provides feedback to quadrants one (Q1) and two (Q2). The Quadrant 3 (Q3) of agile testing primarily emphasizes the response for the previous two phases (Quadrant 1 and Quadrant 2).
- The execution of agile testing involves number of iterations and in Quadrant 3 (Q3), these reviews and responses of the particular iterations are sustained that helps to strengthen the code.

- To test the user experience and determine business results allows the testing team to learn as the test develops.
- The agile team, business owners and even customers realistically use the product. In the third quadrant, the test cases have been designed to implement automation testing, which helps us develop certainty in the particular product.
- In Quadrant 3 (Q3) of agile testing, below types of testing can be executed:
 - **Usability Testing:** Usability testing can also include navigation from page to page or even something as simple as the tabbing order. Knowledge of how people use systems is an advantage when testing usability.
 - **Exploratory Testing:** Exploratory testing is central to this quadrant. During exploratory testing sessions, the tester simultaneously designs and performs tests, using critical thinking to analyze the results. This offers a much better opportunity to learn about the application than scripted tests.
 - **User Acceptance Testing (UAT):** The UAT gives customers a chance to give new features a good workout and see what changes they may want in the future, and it's a good way to gather new story ideas. If the team is delivering software on a contract basis to a client, UAT might be a required step in approving the finished stories.

Quadrant 4 (Tools):

- The Quadrant 4 (Q4) of agile testing primarily emphasizes the product's non-functional requirements, including compatibility, performance, security, and constancy.
- With the help of this Quadrant 4, the application is made to deliver the non-functional qualities and expected value.
- In other words, we can say that the Quadrant 4 (Q4) in agile testing ensures that the code fulfils all the non-functional requirements.
- The types of tests that fall into the Quadrant 4 (Q4) in agile testing are just as critical to agile development as to any type of software development.
- Like other Quadrants, various types of testing are performed in Quadrant 4 (Q4) in agile testing to deliver the non-functional qualities and the expected value.
- The Quadrant 4 (Q4) in agile testing consists of non-functional testing such as stress testing, performance testing, Scalability testing, Security testing and load testing, etc.

5.5 AUTOMATED TESTS

- Today, organizations/firms round the globe are embracing agile methodology of software development. Agile methodology with its core principle of frequent deploys to production has set stringent testing targets.
- Manual testing team with its limited capabilities can't alone meet the testing expectations/requirements set by agile methodology.

- In today's world, test team relies heavily on Test Automation to match agile methodology testing expectations.
- Traditional automation approaches with long cycles of test development fails to keep pace with these new expectations.
- This puts in place a requirement to develop a new test automation development methodology that best meets the agile software development expectations. This test automation methodology is called AgileAutomation.
- Test automation is a core agile practice and agile projects depend on automation. Automation testing in Agile allows creating test cases that will run automatically every time new code is pushed to the code repository for a specific application.
- Agile development and testing is growing in popularity and smart testing teams keep pace with current development trends.
- In agile methodology, software is released in small iterations and each iteration goes through planning, estimation, development, integration, testing and release.
- Because of frequent releases, test automation becomes ever so important as developers need to get quick feedback on the status of the application.
- Automation testing works on agile projects by running a large number of tests repeatedly to make sure an application doesn't break whenever new changes are introduced - at the Unit-, API- and GUI-level.
- For number of agile development teams, these automated tests are executed as part of a Continuous Integration (CI) build process, where developers check code into a shared repository several times a day.
- Each check-in is then verified by an automated build, allowing teams to detect errors and conflicts as soon as possible.
- CI tools such as Jenkins, Bamboo, and Selenium are also used to build, test and deploy applications automatically when requirements change in order to speed up the release process.
- Test automation allows agile teams to execute more tests in less time, increasing coverage and freeing human testers to do more high-level, exploratory testing.
- Since, automation test scripts are re-usable, they can be used to do more comprehensive testing by testing repetitive steps with different data sets, such as those for cross-browser or cross-device compatibility.
- Among the risks of automation are those related to the need for version control and maintainability of test scripts and test results.
- Choosing the right automation testing tool is critically important since we want to avoid ones that are incompatible with other software testing tools in the test environment.
- Agile automation testing in software development is an approach of using test automation in agile methodologies.

- The purpose of agile automation testing is to make the software development process more effective and efficient while maintaining the quality and time as well as resource consumption.
- There are following reasons for automation to be successful using agile:
 1. Manual testing takes too long and time consuming.
 2. Manual processes are error prone.
 3. Manual testing does not give feedback early and often.
 4. Tests and examples that drive coding can do more.
 5. Manual tests do not provide documentation.
 6. Manual testing is not repeatable and rewriting is tedious.
- Fig. 5.5 shows why we adopt test automation in agile testing.

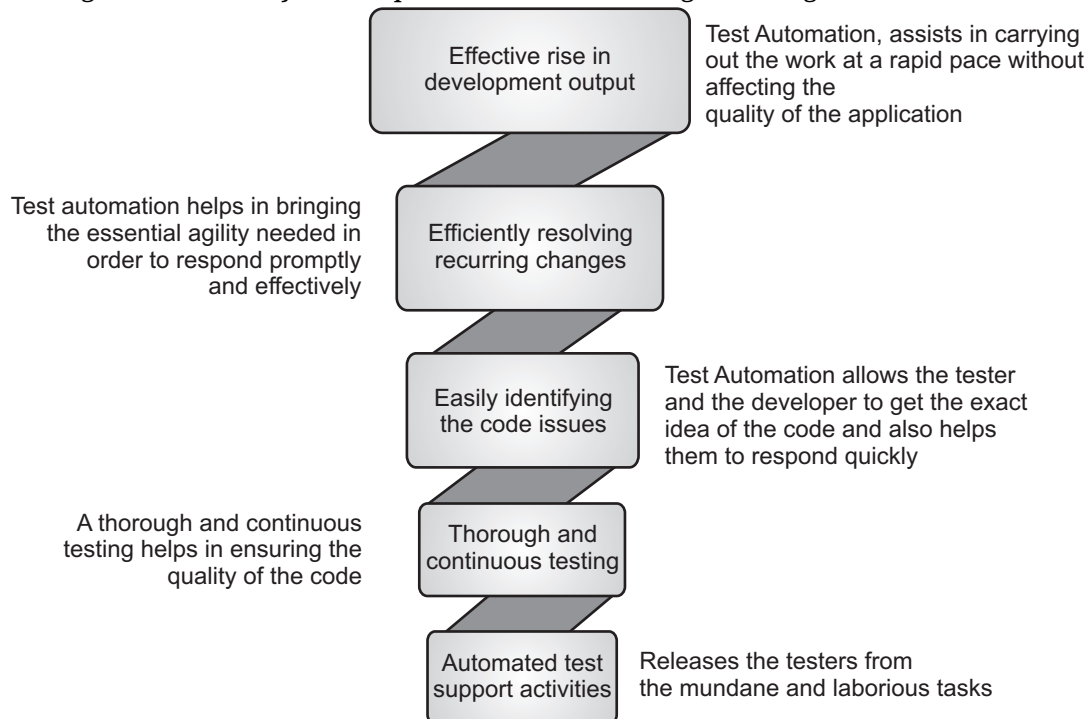


Fig. 5.5: Reasons for Test Automation in Agile Testing

- Fig. 5.6 shows the test automation pyramid. We like the version that Mike Cohn introduced, which shows the foundation layer made up of technology-facing unit and component tests.
- We recognize that number of test teams will struggle with this idea, because it seems the opposite of what many teams currently have.

- Number of test teams has been taught the “V” model of testing, where activities such as component, system, and release testing are done in sequence after coding activities.
- Other test teams have an inverted pyramid, with the majority of the tests in the functional or presentation layer.
- The agile test automation pyramid shows following three different layers of automated tests:
 1. **Lowest Tier:** Lowest tier is the foundation that supports all of the rest. The lowest tier is mainly made up of robust unit tests and component tests, the technology-facing tests that support the team. This layer represents the bulk of the automated tests. They are generally written in the same language as the system under test, using the xUnit family of tools. After a team has mastered the art of TDD, these tests are by far the quickest and least expensive to write. They provide the quickest feedback, too, making them highly valuable. They have the biggest ROI by far of any type of test. In agile development, we try to push as many tests as possible to this layer. While business-facing tests tend to go in one of the higher levels, we implement them at the unit level when it makes sense. If they’re tests the customers don’t have to be able to read, and they can be coded much more quickly as unit tests, it’s a good option. Other types of technology-facing tests such as performance tests may also be possible at the unit level.
 2. **Middle Tier:** The middle tier in the pyramid is the layer that includes most of the automated business-facing tests written to support the team. These are the functional tests that verify that we are “building the right thing.” The tests in middle layer may include “story” tests, “acceptance” tests, and tests that cover larger sets of functionality than the unit test layer. These tests operate at the API level or “behind the GUI,” testing the functionality directly without going through the GUI. We write test cases that set up inputs and fixtures that feed the inputs into the production code, accept the outputs, and compare them to expected results. Because these tests bypass the presentation layer, they are less expensive to write and maintain than tests that use the interface. Fit (a tool for agile collaboration on software requirements) and FitNesse (an open source testing tool) are examples of tools used for the middle layer of the pyramid. Home-grown test harnesses that use spreadsheets or other business friendly means for defining test cases are also common.
 3. **Top Tier:** It represents what should be the smallest automation effort, because the tests generally provide the lowest ROI. These tests are the ones done through the

GUI, the ones that actually operate and manipulate the presentation layer. They are written after the code is completed, and so are usually written to critique the product and go directly to the regression suite. These tests are traditionally more expensive to write, although there are new tools that help reduce the investment needed. Because components of the user interface tend to be changed often, these tests are much more brittle than tests that work at a functional or unit level. For example, just renaming HTML elements could cause a test script to fail. Operating through the user interface also slows these tests down, compared to tests in the lower levels of the pyramid that operate directly on production code. The tests in the top layer do provide important feedback, but a suite of GUI tests may take hours to run rather than the few minutes required for unit-level test suites. We want to minimize the number of tests at this layer, so they should only form the tip of the pyramid.

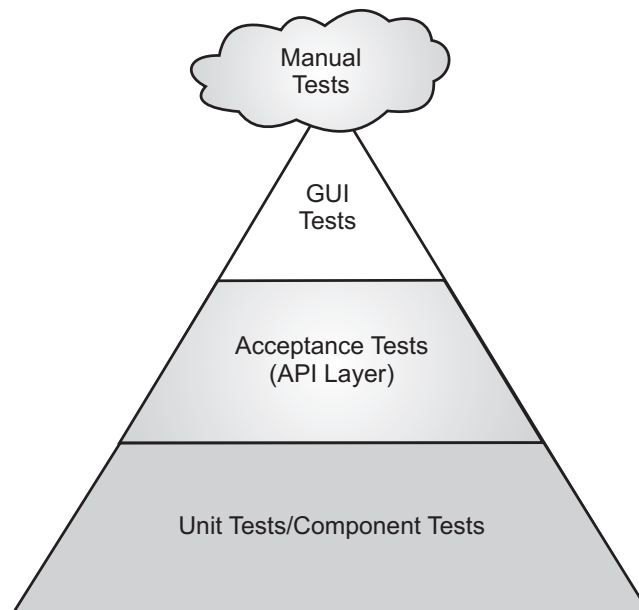


Fig. 5.6: Pyramid for Test Automation

PRACTICE QUESTIONS

Q. I Multiple Choice Questions:

1. A software testing practice that follows the principles of agile software development is called as,
 - (a) Agile Testing
 - (b) Web Testing
 - (c) System Testing
 - (d) Performance Testing

2. Principles of Agile testing includes,
 - (a) Shortening Feedback Response Time
 - (b) Testing is NOT a Phase and Reduces Test Documentation
 - (c) Test Driven and Reduces Test Documentation
 - (d) All of the mentioned
3. Which testing in agile testing helps to discover the unknown risks from the software that a simple testing approach could not have noticed?
 - (a) Session-based
 - (b) Exploratory
 - (c) Both (a) and (b)
 - (d) None of the mentioned
4. Agile testing _____ may be understood as a traditional process or strategies to perform the end-to-end agile testing of a software application.
 - (a) methodology
 - (b) Manifesto
 - (c) quadrants
 - (d) None of the mentioned
5. Advantages of Agile testing includes,
 - (a) Offers efficient risk management
 - (b) Ensures rapid product delivery
 - (c) Promotes feature driven development
 - (d) All of the mentioned
6. Agile is,
 - (a) iterative
 - (b) incremental
 - (c) Both (a) and (b)
 - (d) None of the mentioned
7. Agile _____ of software development is a specific approach of software development that is carried out to develop and deliver a software product in minimal time with the least possibility of defects.
 - (a) methodology
 - (b) Manifesto
 - (c) quadrants
 - (d) None of the mentioned
8. Agile testing methodologies includes,
 - (a) Feature Driven Development (FDD)
 - (b) Dynamic Software Development Method (DSDM)
 - (c) Extreme Programming (XP) and Crystal
 - (d) All of the mentioned
9. Which is a core agile practice?
 - (a) Test methodology
 - (b) Test Manifesto
 - (c) Test automation
 - (d) None of the mentioned

Answers

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

Q. II Fill in the Blanks:

1. _____ testing applies the principles of agile development to the practice of testing.
2. Agile is an _____ development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.
3. Agile testing quadrants may be considered as a tool or a manual outlined by the _____.
4. _____ testing works on agile projects by running a large number of tests repeatedly to make sure an application doesn't break whenever new changes are introduced at the Unit or the API.
5. The Agile testing _____ provide a helpful taxonomy to help teams identify, plan and execute the testing needed.
6. In Agile _____ a software is developed in small iterative and incremental process.
7. Agile testing is a type of software testing that follows the _____ of agile software development to test the software application.
8. The Agile testing Quadrants is a matrix which provides a schematic classification to help teams identify and _____ the testing needed.
9. Automation testing in Agile allows creating _____ that will run automatically every time new code is pushed to the code repository for a specific application.
10. Agile testing quadrants may be seen as an established procedure or _____ or steps to carry out the end-to-end agile testing of a software application.
11. In the second quadrant of Agile testing, we will see mainly emphasis on the customer _____ given to the team before and throughout the testing procedures.

Answers

1. Agile	2. iterative	3. Brain Marick	4. Automation
5. Quadrants	6. methodology	7. principles	8. plan
9. test cases	10. guidelines	11. requirements	

Q. III State True or False:

1. Agile testing refers to a software testing practice that follows different principles of agile software development.
2. Agile is an iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.

3. Agile testing means testing software for defects quickly or within the context of agile and give quick feedback for better and faster development of the project.
4. The Agile software testing practice is an iterative and incremental approach. The entire testing team collaborates to find defects in the software while validating its quality, performance, and effectiveness.
5. In agile testing methodology, both the development and testing tasks are performed collaboratively while ensuring an exclusive tester for testing purposes.
6. Agile testing quadrants are beneficial to define the scope and coverage of automation tests in the planning.
7. In the first quadrant of Agile testing, we will see mainly emphasis on the quality of the code.
8. Agile projects depend on automation.
9. The fourth Quadrant of agile testing primarily emphasizes the product's functional requirements, including compatibility, performance, security, and constancy.

Answers

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

Q. IV Answer the following Questions:

(A) Short Answer Questions:

1. Define Agile.
2. Define Agile testing.
3. List any two Agile principles.
4. Define Agile testing quadrants.
5. What is Agile test automation?
6. Define Agile values.
7. What is Agile methodology?

(B) Long Answer Questions:

1. What is Agile testing? List its features.
2. What are the methods used in Agile testing? Explain two of them in detail.
3. With the help of diagram describe Agile testing strategies.
4. Explain Agile testing life cycle diagrammatically.
5. What are the advantages and disadvantages of Agile testing?
6. Differentiate between traditional testing and Agile testing.
7. List Agile principles in detail.

8. Describe Agile values in detail.
9. What is Agile quadrant? What are its types? Explain with diagram.
10. Write a short note on: Automated tests.
11. With the help of diagram lists reasons for test automation in Agile testing.
12. What is Agile Manifesto? Describe in detail.

■■■