BCA
Semester-6

Software Testing

Subject Code : CC - 307

Prof. Kamesh Raval
MCA [Master in Computer Application]
Som-Lalit Institute of Computer Application, A'bad.

Prof. Parimalkumar P. Patel
MCA [Master in Computer Application]
Diploma in Cyber Security
Khyati School of Computer Application-Abad

Shri Keyur Shah
MCA [Master in Computer Application], M.Phil.
System Analyst-GTU, Formerly Coordinator- CPICA.

Prof. Niku Brahmbhatt
MCA (Master in Computer Application)
JG-CCA, ASIA-Campus,
Opp. Drive-In-Cinema, Thaltej -A'bad

Reviewer: **Dr. Bhavna Bajpai**
Ph.D. (Computer Science)
Associate Professor, Dept of Computer,
Dr . C. V. Raman University, Khandwa (MP) - 01

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.

COMPUTER WORLD
43, 5th Floor, SANIDHYA Complex, Nr. M. J. Library,
Opp. Sanyas Ashram, Ashram Road, Ahmedabad-09.
Mobile : 9725019114, 9725022917, 9725020595, 9825020595
URL : www.computerworld.ind.in | e-Mail :info@computerworld.ind.in

Index

UNIT-1 Introduction of Software Testing 05 to 20

1.1 Definition of Software Testing
1.2 Terms: Failure, Error, Fault, Defect, Bug
1.3 Goals of Testing
1.4 Principles of Testing
1.5 Software Testing Life Cycle (STLC)
1.6 Verification and Validation Model (V-testing Life cycle)
1.7 Static and Dynamic Testing

UNIT-2 Type of Testing 21 to 84

2.1 Black-box Testing
2.2 Black box Testing Techniques (Methods)
2.2.1 Boundary Value Analysis (BVA)
2.2.2 Equivalence Class (EVC) testing method
2.2.3 Decision Table-Based Testing
2.3 White-box Testing
2.4 Classification of white box testing
2.4.1 Structural Testing
2.5 Static Testing

UNIT-3 Levels of Testing 85 to 103

3.1 Unit Testing: Overview
3.2 Integration Testing: Overview

Index

UNIT-1 Introduction of Software Testing 05 to 20

1.1 Definition of Software Testing

1.2 Terms: Failure, Error, Fault, Defect, Bug

1.3 Goals of Testing

1.4 Principles of Testing

1.5 Software Testing Life Cycle (STLC)

1.6 Verification and Validation Model (V-testing Life cycle)

1.7 Static and Dynamic Testing

UNIT-2 Type of Testing 21 to 84

2.1 Black-box Testing

2.2 Black box Testing Techniques (Methods)

2.2.1 Boundary Value Analysis (BVA)

2.2.2 Equivalence Class (EVC) testing method

2.2.3 Decision Table-Based Testing

2.3 White-box Testing

2.4 Classification of white box testing

2.4.1 Structural Testing

2.5 Static Testing

UNIT-3 Levels of Testing 85 to 103

3.1 Unit Testing: Overview

3.2 Integration Testing: Overview

UNIT-1

Introduction of Software Testing

- Definition of Software Testing and its Role
- Terms- Failure, Error, Fault, Defect, Bug
- Goals of Testing
- Principles of Testing
- Software Testing Life Cycle
- Verification and Validation : V-testing Life cycle
- Static and Dynamic Tests

3.3 Techniques: Graph based & Path based
3.4 Functional Testing
3.5 System Testing: Overview
3.6 Categories: Reliability Security Performance Recovery
3.7 Acceptance Testing: Overview, Types of Acceptance Testing

UNIT-4 Test Management 104 to 115

4.1 Test Planning
4.2 Test Management
4.2.1 Choice of Standards
4.2.2 Test Infrastructure Management
4.2.3 Test Process
4.2.4 Test Reporting

Paper 2021 116 to 118

UNIT-1

Introduction of Software Testing

- Definition of Software Testing and its Role
- Terms:- Failure, Error, Fault, Defect, Bug
- Goals of Testing
- Principles of Testing
- Software Testing Life Cycle
- Verification and Validation: - V-testing Life cycle
- Static and Dynamic Testin

1.2 Terms: Failure, Error, Fault, Defect, Bug

There are some important terms of Software testing which is required to understand here.

- **Failure:** It is the incompetence of a system or component to perform required function according to its specification. In under certain environment and situation, defects in the application get executed then the system will produce the incorrect results causing a failure. Failures can also be caused because of the other reasons also like:
  - ⇨ Because of the environmental conditions as well like a strong magnetic field, electronic field, a radiation burst or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.
  - ⇨ Failures may also possible because of human error in interacting with the software, perhaps a wrong input value being entered or an output being misinterpreted.
  - ⇨ Failures may also be caused by someone purposefully trying to cause a failure in the system.
- **Error:** Refers to difference between Actual Output and Expected output. Error is terminology of Developer. The mistakes made by programmer are known as an 'Error'. This could happen because of the following reasons:
  - ⇨ Because of some confusion in understanding the functionality of the software.
  - ⇨ Because of some miscalculation of the values.
  - ⇨ Because of misinterpretation of any value, etc.
- **Fault:** It is a condition that causes the software to fail to perform its required function.
- **Defect:** The bugs introduced by programmer inside the code are known as a defect. This can happen because of some programmatic mistakes.
- **Bug:** Bug is terminology of Tester. The Bug is the informal name of defects, which means that software or application is not working as per the requirement. In software testing, a software bug can also be issue, error, fault, or failure. The bug occurred when developers made any mistake or error while developing the product.

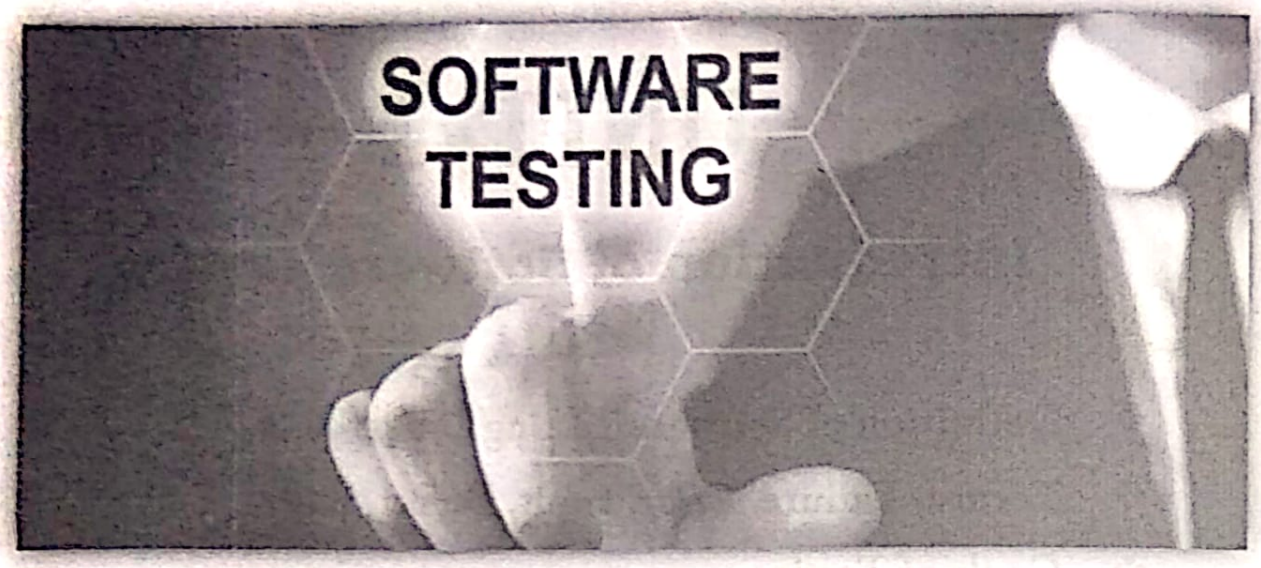
1.3 Goals of Testing:

The ultimate goal of software testing is to troubleshoot all the issues and bugs as well as control the quality of a resulted product. It can be defined as under:

- **Bug Prevention:** Quality Assurance engineers prevent defects in a system at the earliest stage of development. The bug-prevention objective is superior to others and

UNIT-1 Introduction of Software Testing

1.1 Definition of Software Testing:



The process or method of finding error/s in a software application or program so that the application functions according to the end user's requirement is called software testing.

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.

It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

- **Role of Software Testing:**

The role of testing in software development begins with improved reliability, quality and performance of the software. It assists a developer to check out whether the software is performing the right way and to assure that software is not performing what it is not supposed to do. Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

1.2 Terms: Failure, Error, Fault, Defect, Bug

There are some important terms of Software testing which is required to understand here.

- **Failure:** It is the incompetence of a system or component to perform required function according to its specification. In under certain environment and situation, defects in the application get executed then the system will produce the incorrect results causing a failure. Failures can also be caused because of the other reasons also like:
  - ⇨ Because of the environmental conditions as well like a strong magnetic field, electronic field, a radiation burst or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.
  - ⇨ Failures may also possible because of human error in interacting with the software, perhaps a wrong input value being entered or an output being misinterpreted.
  - ⇨ Failures may also be caused by someone purposefully trying to cause a failure in the system.
- **Error:** Refers to difference between Actual Output and Expected output. Error is terminology of Developer. The mistakes made by programmer are known as an 'Error'. This could happen because of the following reasons:
  - ⇨ Because of some confusion in understanding the functionality of the software.
  - ⇨ Because of some miscalculation of the values.
  - ⇨ Because of misinterpretation of any value, etc.
- **Fault:** It is a condition that causes the software to fail to perform its required function.
- **Defect:** The bugs introduced by programmer inside the code are known as a defect. This can happen because of some programmatic mistakes.
- **Bug:** Bug is terminology of Tester. The Bug is the informal name of defects, which means that software or application is not working as per the requirement. In software testing, a software bug can also be issue, error, fault, or failure. The bug occurred when developers made any mistake or error while developing the product.

1.3 Goals of Testing:

The ultimate goal of software testing is to troubleshoot all the issues and bugs as well as control the quality of a resulted product. It can be defined as under:

- **Bug Prevention:** Quality Assurance engineers prevent defects in a system at the earliest stage of development. The bug-prevention objective is superior to others and

implies not only expectation but also prevention of defects from recurring in the future. In the long run, bug prevention helps to reduce the product time to market, reduce the cost of software quality maintenance and increase the customer satisfaction and loyalty to your product.

- **Bug Detection:** Quality Assurance experts detect and root out bugs and malfunctions before customers find them. It is a short-term objective that requires a scrutinise approach which can be provided by manual software testing.
- **User Satisfaction:** Quality Assurance team make sure that the product satisfies the user requirements and works as desired. In the process of the software verification & validation, a tester usually writes a set of test cases which help to determine the software compliance with specific business and user requirements under positive and negative conditions. For instance, our team establishes the set of tests to ensure the main functions and features cover different scenarios, work properly in a range of countries and locations.
- **Software quality and reliability:** Keeping control of software quality means keeping bugs at a low level and making sure software is compatible. Software compatibility is the capability of software or an app to work well with other hardware, software or network, including web, desktop, mobile platform types, all types of operating systems and web browsers, etc.
- **Recommendations:** Quality Assurance team provides suggestions for software improvements. A good software testing provider should not only tell the difference between the actual and expected result but also make suggestions on how to improve this or that software element performance from the user perspective. In particular, the team might recommend how to make the app more spontaneous for customers as well as provide with simpler ways of software development.
- **Information for stakeholders:** Quality assurance team inform to stakeholders regarding Software testing and Quality Assurance activities which provide understandings about technical restrictions, risk factors, ambiguous requirements, etc. QA Team also share the testing reports with stakeholders which actually allows making changes when they are necessary.
- **Confidence in the product:** Due to software testing report, decision-makers get sufficient information about the product highlights and areas that require increased attention. Thus, product owners can gain confidence in the product before the launch or

decide to release its less stunning but more stable version. In both cases, they operate with facts.

❖ **Short-term or Immediate goals:**

These goals are immediate results after performing testing. These goals may be set in the individual phases of SDLC. Some of them are discussed below.

- **Bug discovery:** The immediate goal of testing is to find errors at any stage of software development. More the bugs discovered at an early stage, better will be the success rate of software testing.

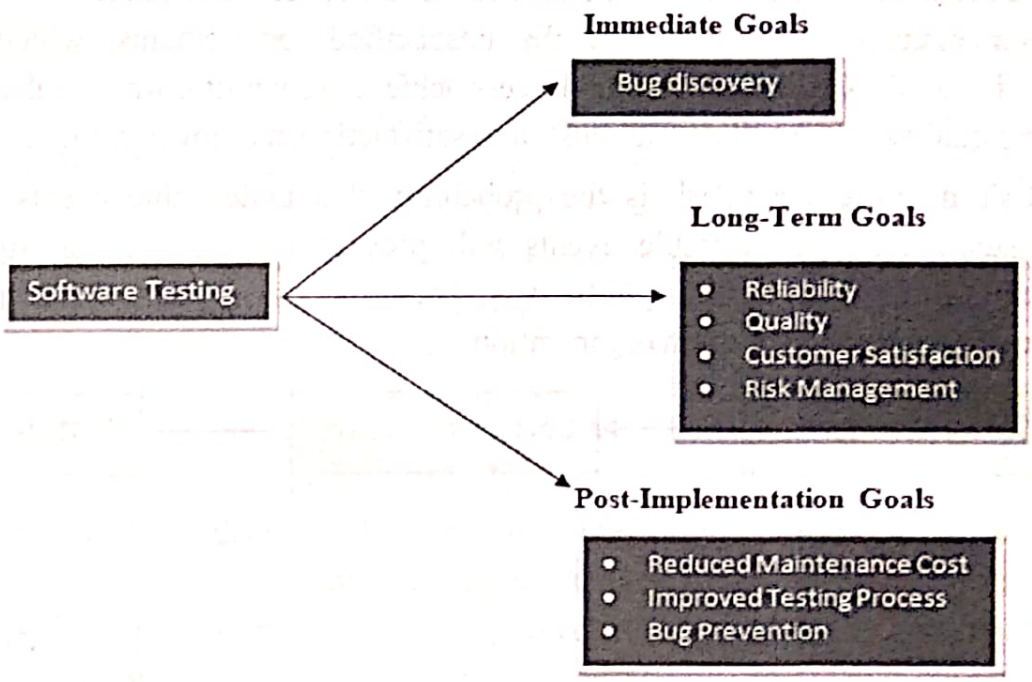


Figure 1.1 : Short-term or immediate goals

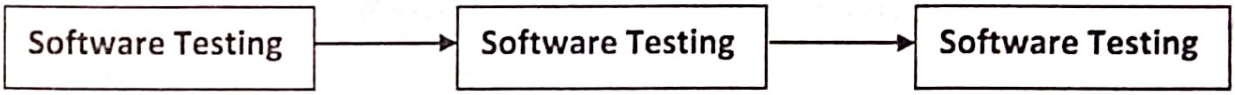


Figure 1.2 : Testing Produces Reliability and Quality

- **Long-term goals:**

These goals affect the product quality in the long run when one cycle of the SDLC is complete. Some of them are discussed here.

- **Quality:** Since the software is also a product, its quality is primary from the users' point of view. Thorough testing ensures superior quality. Therefore, the first goal of understanding and performing the testing process is to enhance the quality of the software product. Though quality depends on various factors, such as correctness,

integrity, efficiency, etc., reliability is the major factor to achieve quality. The software should be passed through a rigorous reliability analysis to attain high-quality standards. Reliability is a matter of confidence that the software will not fail, and this level of confidence increases with rigorous testing. The confidence in the reliability, in turn, increases the quality, as shown in Figure 2.

- **Customer satisfaction:** From the user's perspective, the prime concern of testing is customer satisfaction only. If we want the customer to be satisfied with the software product, then testing should be complete and thorough. Testing should be complete in the sense that it must satisfy the user for all the specified requirements mentioned in the user manual, as well as for the unspecified requirements, which are otherwise understood. A complete testing process achieves reliability, which enhances the quality, and quality, in turn, increases customer satisfaction, as shown in Figure 3.
- **Risk management:** Risk is the probability that undesirable events will occur in a system. These undesirable events will prevent the organization from successfully implementing its business initiatives. Thus, the risk is basically concerned with the business perspective of an organization.

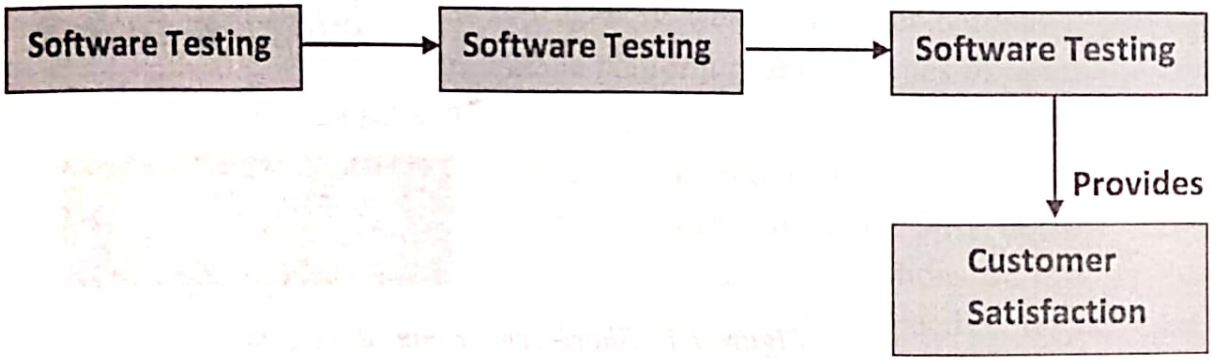


Figure 1.3 : Quality Leads to Customer Satisfaction

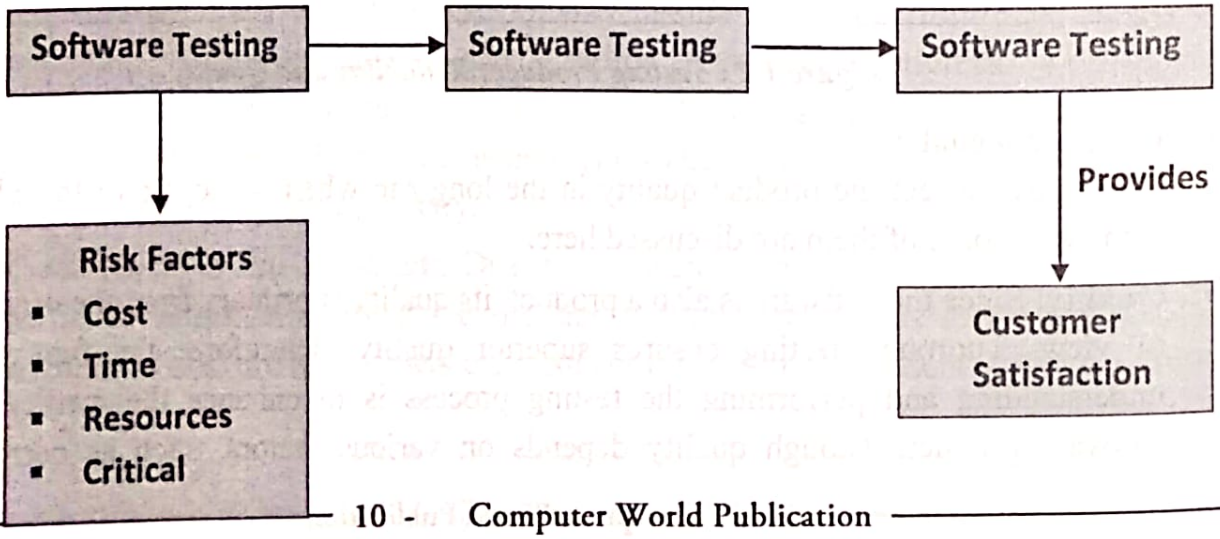


Figure 1.4 : Testing Controlled by Risk Factors

Risks must be controlled to manage them with ease. Software testing may act as a control, which can help in eliminating or minimizing risks (see Figure 4). Thus, managers depend on software testing to assist them in controlling their business goals.

- **Post-implementation goals:**
  These goals are important after the product is released. Some of them are discussed here.
- **Reduced maintenance cost:** The maintenance cost' of any software product is not its physical cost, as the software does not wear out. The only maintenance cost in a software product is its failure due to errors. Post-release errors are costlier to fix, as they are difficult to detect. Thus, if testing has been done rigorously and effectively, then the chances of failure are minimized and, in turn, the maintenance cost is reduced.
- **Improved software testing process:** A testing process for one project may not be successful and there may be scope for improvement. Therefore, the bug history and post-implementation results can be analyzed to find out snags in the present testing process, which can be rectified in future projects. Thus, the long-term post-implementation goal is to improve the testing process for future projects.
- **Bug prevention:** It is the consequent action of bug discovery. From the behavior and interpretation of bugs discovered, everyone in the software development team gets to learn how to code safely such that the bugs discovered are not repeated in later stages or future projects. Though errors cannot be prevented to zero, they can be minimized. In this sense, bug prevention is a superior goal of testing.

1.4 Principles of Testing:

Software testing is a process of executing a program with the aim of finding the error. To make our software perform perfect, it should be error and bug free. If testing is done successfully it will remove all the errors and bugs from the software.

There are seven principles in software testing:

1. Testing shows presence of defects
2. Exhaustive testing is not possible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of errors fallacy

- Testing shows presence of defects: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defects free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not removes all defects.
- Exhaustive testing is not possible: It is the process of testing the functionality of a software in all valid and invalid inputs and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test cases. It can test only some test cases and assume that software is correct and it will produce the correct output in every test cases. If the software will test every test cases then it will take more cost, effort, etc. and which is impractical.
- Early Testing: To find the defect in the software, early test activity shall be started. For better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase.
- Defect clustering: In a project, a small number of the module can contain most of the defects. *Pareto Principle* to software testing state that 80% of software defect comes from 20% of modules. The Pareto Principle, named after esteemed economist Vilfredo Pareto which is also known as the Pareto Rule or the 80/20 Rule.
- Pesticide paradox (inconsistency): Repeating the same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.
- Testing is context dependent: Testing approach depends on context of software developed. Different types of software need to perform different types of testing. For example, the testing of the e-commerce site is different from the testing of the mobile application.
- Absence of errors fallacy: If built software is 99% bug-free but it does not work as per the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it also mandatory to fulfil all the customer requirements.

1.5 Software Testing Life Cycle (STLC):

STLC is a sequence of different activities performed during the software testing process.

❖ Characteristics of STLC:

- STLC is a fundamental part of Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.
- STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.

- STLC produces a step-by-step process to ensure quality software.

  In the initial stages of STLC, while the software product or the application is being developed, the testing team analyses and defines the scope of testing, entry and exit criteria and also the test cases. It helps to reduce the test cycle time and also enhance the product quality.

  As soon as the development phase is over, testing team is ready with test cases and start the execution. This helps in finding bugs in the early phase.

Phases of STLC:

Depending on the difficulty and goal of a project, STLC may be informal or highly formal. Software Testing Life Cycle has seven key phases. But it's not essential to follow all of them while you are working on your project. The choice of STLC phases depends on many factors, including the type of product, time, and resources you have to complete.

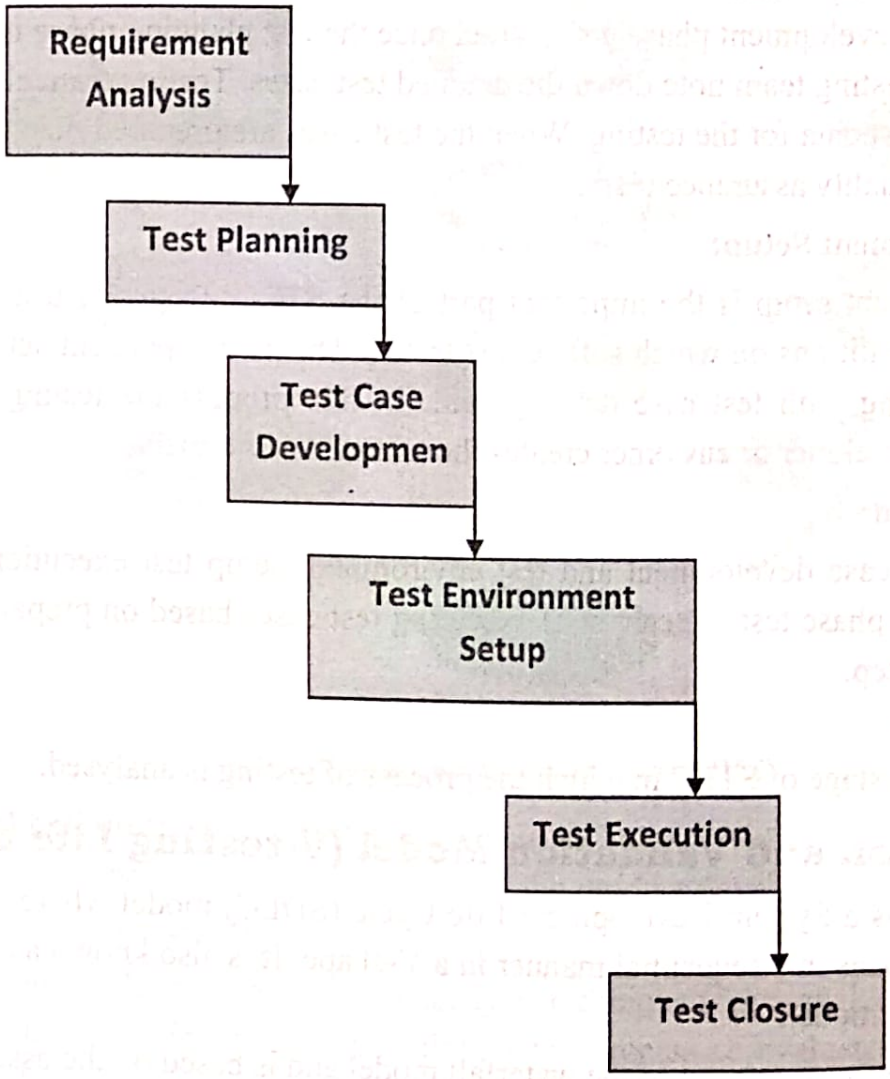


Figure 1.5 : Step of Software Testing Life Cycle (STLC)

1. **Requirement Analysis:**

   Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.

2. **Test Planning:**

   Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.

3. **Test Case Development:**

   The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.

4. **Test Environment Setup:**

   Test environment setup is the important part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involve and developer or customer creates the testing environment.

5. **Test Execution:**

   After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.

6. **Test Closure:**

   This is the last stage of STLC in which the process of testing is analysed.

1.6 Verification and Validation Model (V-testing Life cycle):

The V-model is a System Development Life Cycle (SDLC) model where execution of processes happens in a sequential manner in a 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 the association of a testing phase for each corresponding development stage. In this model, every phase of

SDLC is tested. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

❖ **V-Model – Design:**

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.

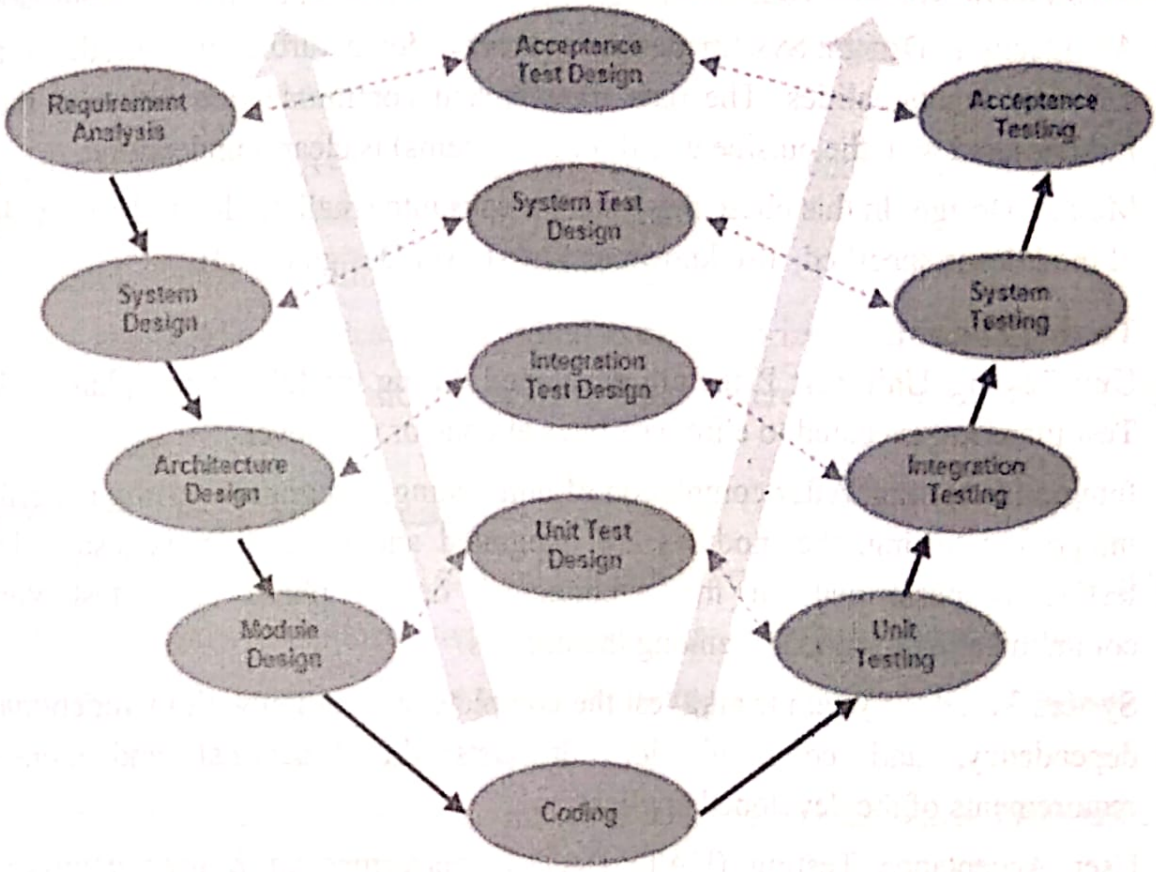


Figure 1.6 : V-Model of the SDLC

- **Verification:** It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.
- **Validation:** It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation phases are joined by coding phase in V-shape. Thus it is called V-Model.

- **Design Phase:**

⇨ Requirement Analysis: This phase contains detailed communication with the customer to understand their requirements and expectations. This stage is known as Requirement Gathering.

⇨ System Design: This phase contains the system design and the complete hardware and communication setup for developing product.

⇨ Architectural Design: System design is broken down further into modules taking up different functionalities. The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.

⇨ Module Design: In this phase the system breaks into small modules. The detailed design of modules is specified, also known as Low-Level Design (LLD).

- **Testing Phases:**

⇨ Unit Testing: Unit Test Plans are developed during module design phase. These Unit Test Plans are executed to eliminate bugs at code or unit level.

⇨ Integration testing: After completion of unit testing, Integration testing is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed on the Architecture design phase. This test verifies the communication of modules among themselves.

⇨ System Testing: System testing test the complete application with its functionality, inter dependency, and communication. It tests the functional and non-functional requirements of the developed application.

⇨ User Acceptance Testing (UAT): UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets user's requirement and system is ready for use in real world.

- **Principles of V-Model:**

⇨ Large to Small: In V-Model, testing is done in a hierarchical perspective, For example, requirements identified by the project team, create High-Level Design, and Detailed Design phases of the project. As each of these phases is completed the requirements, they are defining become more and more refined and detailed.

- ⇨ Data/Process Integrity: This principle states that the successful design of any project requires the incorporation and interrelation of both data and processes. Process elements must be identified at each and every requirements.
- ⇨ Scalability: This principle states that the V-Model concept has the flexibility to accommodate any IT project irrespective of its size, complexity or duration.
- ⇨ Cross Referencing: Direct correlation between requirements and corresponding testing activity is known as cross-referencing.
- ⇨ Tangible Documentation: This principle states that every project needs to create a document. This documentation is required and applied by both the project development team and the support team. Documentation is used to maintaining the application once it is available in a production environment.

Industrial Challenge: As the industry has evolved, the technologies have become more complex, increasingly faster, and forever changing.

- ⇨ Accurately define and refine user requirements.
- ⇨ Design and build an application according to the authorized user requirements.
- ⇨ Validate that the application they had built adhered to the authorized business requirements.

❖ V-Model - Pros and Cons:

The advantage of the V-Model method is that it is very easy to understand and apply.

- ✓ This is a highly-disciplined model and Phases are completed one at a time.
- ✓ Works well for smaller projects where requirements are very well understood.
- ✓ Simple and easy to understand and use.
- ✓ The simplicity of this model also makes it easier to manage. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows –

- ☒ High risk and uncertainty.
- ☒ Not a good model for complex and object-oriented projects.
- ☒ Poor model for long and ongoing projects.
- ☒ Not suitable for the projects where requirements are at a moderate to high risk of changing.
- ☒ Once an application is in the testing stage, it is difficult to go back and change functionality.

1.7 Static and Dynamic Testing:

❖ Static Testing:

Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application.

Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily.

❖ Dynamic Testing:

Dynamic Testing is a type of Software Testing which is performed to analyse the dynamic behaviour of the code. It includes the testing of the software for the input values and output values that are analysed.

Difference between Static Testing and Dynamic Testing:

Static Testing	Dynamic Testing
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
In static testing whole code is not executed.	In dynamic testing whole code is executed.
Static testing prevents the defects.	Dynamic testing finds and fixes the defects.
Static testing is performed before code deployment.	Dynamic testing is performed after code deployment.
Static testing is less costly.	Dynamic testing is highly costly.
Static Testing involves checklist for testing process.	Dynamic Testing involves test cases for testing process.

Static Testing	Dynamic Testing
It includes walkthroughs, code review, inspection etc.	It involves functional and nonfunctional testing.
It generally takes shorter time.	It usually takes longer time as it involves running several test cases.
It can discover variety of bugs.	It expose the bugs that are explorable through execution hence discover only limited type of bugs.
Static Testing may complete 100% statement coverage in comparably less time.	While dynamic testing only achieves less than 50% statement coverage.
Example: Verification	Example: Validation

Exercises

❖ Answer the following Questions in brief.

1. What is the role of Software Testing?
2. What are the principles of Software Testing?
3. Explain the different phases of STLC.
4. Explain the V-Model with pros and cons
5. What is the difference between static and dynamic testing
6. What are the principles of V-Model?
7. What are the characteristics of STLC?

❖ Fill in the Blanks:

1. Software Testing is a method to ensure that software product is _____ free.
2. _____ is a condition that causes the software to fail to perform its required function.
3. The bugs introduced by programmer inside the code are known as a _____.
4. Software testing is a process of executing a program with the aim of finding the _____.
5. STLC stands for _____.
6. STLC has _____ number of phases.
7. _____ is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application.
8. _____ is a type of Software Testing which is performed to analyse the dynamic behaviour of the code.
9. The _____ is a System Development Life Cycle (SDLC) model where execution of processes happens in a sequential manner in a V-shape.
10. _____ involves static analysis technique (review) done without executing code.

Answer:

1. Defect **2.** Fault **3.** Defect **4.** Error **5.** Software Testing Life Cycle
6. Six **7.** Static Testing **8.** Dynamic Testing **9.** V-model **10.** Verification

UNIT-2

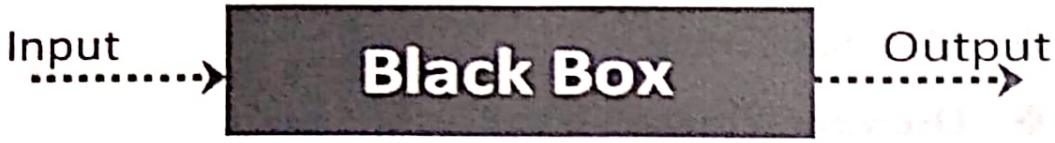
Type of Testing

- Black Box Testing
- Overview: What is & When?
- Techniques
  - Boundary Value Analysis
  - Equivalence class testing
  - Decision Table
- White Box Testing
- What is white box Testing
- Need of white box Testing
- Classification
- Structural : Coverage, Path
- Static : Inspection, Walkthrough , Technical Review

UNIT-2 Type of Testing

2. 1 Black-box Testing:

BLACK BOX TESTING is defined as a dynamic testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software.



It is also known as functional testing.

In this technique, based on function requirements, test cases are designed, input test data is given to the system and results are checked against expected outputs after executing the software.

It helps to find errors in the following categories:

- to test module independently
- to test the functional validity of the software so that incorrect or missing functions can be recognized.
- to look for interface errors
- to test the system behavior and check its performance
- to test the maximum load or stress on the system
- to test the software such that the user/customer accepts the system within defined acceptable limits.

2.2 Blackbox Testing Techniques (Methods):

2.2.1 Boundary Value Analysis (BVA):

The objective of any test case is to have maximum coverage and capability to discover more and more errors. It has been observed that test cases designed with boundary input values have a high chance to find more errors.

Boundary value testing is focused on the values at boundaries. This technique

determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges. E.g.systems having requirements of minimum or maximum temperature, pressure or speed. It is not useful for Boolean variables.

What are boundary values?

The different variables of module under testing having different range. The upper and lower value of that range is known as boundary value.

The boundary values can be of three different types as shown in figure.

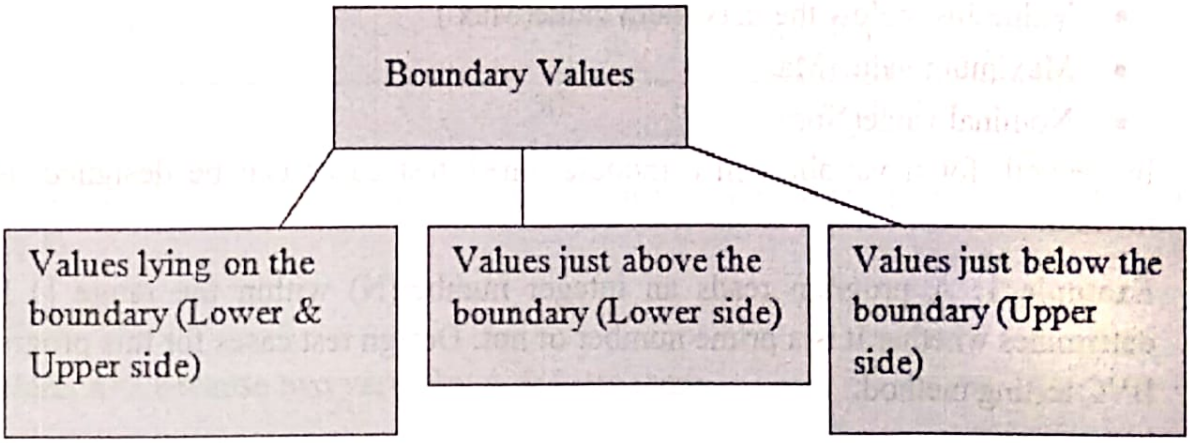


Figure 2.1 : different types of boundary values

Let us consider that there is a variable A which lies between 1 to 10. In this case, the boundary values are as given below:

- Values lying on the boundary are: lower boundary=1 and upper boundary=10
- Value just above the boundary (Lower side) = 2
- Values just below the boundary (Upper side) = 9

During test case designing, we consider single fault assumption.

Single fault assumption

Holding the values of all but one variable at their nominal value and letting that variable assume extreme value. In other word, if there are n variables, then at a time only one variable can take its extreme (boundary) value and others at nominal. Refer example 2.

This technique offers several methods to design test cases as given below:

(1) Boundary Value Checking (BVC):

In this method, the test cases are designed by holding one variable as its extreme value and other variables at their nominal values in the input domain.

These extreme ends like Start- End, Lower- Upper, Maximum - Minimum".

The basic idea in boundary value testing is to select input variable values at their extreme as:

- Minimum value(Min)
- Value Just above the minimum value (Min^+)
- Value Just below the maximum value(Max^-)
- Maximum value(Max)
- Nominal value(Nom)

In general, for n variables in a module, 4n+1 test cases can be designed using this method.

Example 1: A program reads an integer number(N) within the range [1,100] and determines whether it is a prime number or not. Design test cases for this program using BVC testing method.

Solution:

Instead of checking, one value for each input, we will check the values at the boundary values like:

Min value	1
Min^+ value	2
Max value	100
Max^- value	99
Nom value	50 (Mid value)

Here, total number of variables (n)=1

Total test cases designed will be 4*n+1 = 4*1+1 = 5.

If we consider all the above combinations with nominal values, then test cases can be designed: A_{min}, A_{min+}, A_{max}, A_{max-},A_{nom}

Using boundary values, test cases can be designed as shown below:

Test Case ID	Integer Variable	Expected Output
1	1	Not a prime number
2	2	Prime number
3	100	Not a prime number
4	99	Not a prime number
5	50	Prime number

Example 2: A Program computes A^b where A lies in the range[1,10] and b within [1,5].

Solution:

Here, n=2 because two variables A & b are there.

Total test cases will be 4*n+1= 4*2+1= 9

boundary values for variable A	
Min value	1
Min^+ value	2
Max value	10
Max^- value	9
Nom value	5

boundary values for variable b	
Min value	1
Min^+ value	2
Max value	5
Max^- value	4
Nom value	3

If we consider all the above boundary values with nominal value, then possible combinations for test cases will be as shown below:

A_{min}, b_{nom}	A_{min+}, b_{nom}	A_{max}, b_{nom}	A_{max-}, b_{nom},	
A_{nom}, b_{min}	A_{nom}, b_{min+}	Anom, b_{max}	A_{nom}, b_{max-}	A_{nom}, b_{nom}

and test cases will be designed as:

Test Case ID	Integer Variable		Expected Output
	A	b	
1	1	3	1
2	2	3	8
3	10	3	1000
4	9	3	729
5	5	1	5
6	5	2	25
7	5	5	3125
8	5	4	625
9	5	3	125

(2) Robust Testing Method

It is an extension of BVC method. It also checks the behavior of a module for invalid input value (moving beyond the valid input range) by adding two additional boundary values like:

- A value just above the maximum value(Max+)
- A value just below the minimum value(Min-)

For n input variables module, 6n+1 test cases can be designed using this method.

This method has the desirable property that it forces attention on exception handling.

Example 1: A program reads an integer number(N) within the range [1,100] and determines whether it is a prime number or not. Design test cases for this program using Robust testing method.

Solution:

Since there is only one variable N, total test cases will be 6*1+1=7

boundary values for variable n	
Min value	1
Min^{+} value	2
Min^{-} value	0
Max value	100
Max^{-} value	99
Max^{+} value	101
Nom value	50

If we consider all the above boundary values with nominal value, then possible combinations for test cases will be as shown below:

N_{min-}	N_{min}	N_{min+}	N_{max}	N_{max-}
N_{max+}	N_{nom}			

and test cases will be designed as :

Test Case ID	Integer Variable N	Expected Output
1	0	Invalid input
2	1	Not a prime number

3	2	Prime number
4	100	Not a prime number
5	99	Not a prime number
6	101	Invalid input
7	50	Prime number

Example 2: A Program computes A^b where A lies in the range[1,10] and b within [1,5].

Solution:

Here, n=2

Total test cases will be 6*2+1=13

boundary values for variable A		boundary values for variable b	
Min value	1	Min value	1
Min^- value	2	Min^- value	2
Min^- value	0	Min^- value	0
Max value	10	Max value	5
Max^- value	9	Max^- value	4
Max^+ value	11	Max^+ value	6
Nom value	5	Nom value	3

If we consider all the above boundary values with nominal value, then possible combinations for test cases will be as shown below:

A_{min-}, b_{nom}	A_{min}, b_{nom}	A_{min+}, b_{nom}	A_{max}, b_{nom}	A_{max+}, b_{nom},
A_{max-},	A_{nom},	A_{nom},	A_{nom}	Anom,

b_{nom}	b_{min-}	b_{min}	, b_{min+}	b_{max-}
A_{nom},b_{max} x	A_{nom},b_{max} +	A_{nom}, b_{nom}		

and test cases will be designed as :

Test Case ID	Integer Variable		Expected Output
	A	b	
1	0	3	Invalid input A
2	1	3	1
3	2	3	8
4	10	3	1000
5	11	3	Invalid input A
6	9	3	729
7	5	0	Invalid input b
8	5	1	5
9	5	2	25
10	5	4	625
11	5	5	3125
12	5	6	Invalid input b
13	5	3	125

(3) Worst-Case Testing Method:

BVC uses the critical fault assumption and therefore only tests for a single variable at a time assuming its extreme values. By disregarding this assumption we are able to test the outcome if more than one variable were to assume its extreme value (Rejecting

b_{min}	b_{min+}	b_{nom}	b_{max-}	A_{nom}, b_{max}
A_{nom}, b_{min}	A_{nom}, b_{min+}	A_{nom}, b_{nom}	A_{nom}, b_{max-}	
A_{max-}, b_{min}	A_{max-}, b_{min+}	A_{max-}, b_{nom}	A_{max-}, b_{max-}	A_{max-}, b_{max}
A_{max}, b_{min}	A_{max}, b_{min+}	A_{max}, b_{nom}	A_{max}, b_{max-}	A_{max}, b_{max}

and by considering only A at its extreme values, test cases will be designed as :

Test Case ID	Integer Variable		Expected Output
	A	b	
1	1	1	1
2	1	2	1
3	1	3	1
4	1	4	1
5	1	5	1
6	2	1	2
7	2	2	4
8	2	3	8
9	2	4	16
10	2	5	32
11	5	1	5
12	5	2	
13	5	3	

single fault assumption). In other word this method is an extension of BVC by assuming more than one variable on the boundary.

This method is more thorough form of testing and it requires more efforts as it generates more test cases.

For n input variables in a module, 5^n test cases can be designed using this method.

Example 1: A program reads an integer number(N) within the range [1,100] and determines whether it is a prime number or not. Design test cases for this program using worst case testing method.

Since there is one variable, the total number of test cases will be 5^n. Therefore, the number of test cases will be same as BVC. Refer BVC method.

Example 2: A Program computes A^b where A lies in the range[1,10] and b within [1,5]. Design test cases for this program using worst case testing method.

Solution:

Here, n=2

Total test cases will be $5^n=5^2=25$

boundary values for variable A	
Min value	1
Min^+ value	2
Max value	10
Max- value	9
Nom value	5

boundary values for variable b	
Min value	1
Min^+ value	2
Max value	5
Max- value	4
Nom value	3

If we consider all the above boundary values with nominal value, then possible combinations for test cases will be as shown below:

A_{min}, b_{min}	A_{min}, b_{min+}	A_{min}, b_{nom}	A_{min}, b_{max-}	A_{min}, b_{max}
A_{min+},	A_{min+},	A_{min+},	A_{min+},	A_{min+},

b_{min}	b_{min+}	b_{nom}	b_{max-}	b_{max}
A_{nom}, b_{min}	A_{nom}, b_{min+}	A_{nom}, b_{nom}	A_{nom}, b_{max-}	A_{nom}, b_{max}
A_{max-}, b_{min}	A_{max-}, b_{min+}	A_{max-}, b_{nom}	A_{max-}, b_{max-}	A_{max-}, b_{max}
A_{max}, b_{min}	A_{max}, b_{min+}	A_{max}, b_{nom}	A_{max}, b_{max-}	A_{max}, b_{max}

and by considering only A at its extreme values, test cases will be designed as :

Test Case ID	Integer Variable		Expected Output
	A	b	
1	1	1	1
2	1	2	1
3	1	3	1
4	1	4	1
5	1	5	1
6	2	1	2
7	2	2	4
8	2	3	8
9	2	4	16
10	2	5	32
11	5	1	5
12	5	2	25
13	5	3	125

14	5	4	625
15	5	5	3125
16	9	1	9
17	9	2	81
18	9	3	729
19	9	4	6561
20	9	5	59049
21	10	1	10
22	10	2	100
23	10	3	1000
24	10	4	10000
25	10	5	100000

(4) Robust Worst-Case Testing Method

The worst-case method is extended by considering extreme values for every possible input variable & it is known as Robust Worst-Case testing method.

It can be generalized that for n input variables in a module, 7^n test cases can be designed using this method.

Example 1: A Program computes A^b where A lies in the range[1,10] and b within [1,5]. Design test cases for this program using robust worst case testing method.

Solution:

Here, n=2

Total test cases will be $7^n=7^2=49$

boundary values for variable A	
Min value	1
Min^+ value	2

boundary values for variable b	
Min value	1
Min^+ value	2

Min⁻ value	0
Max value	10
Max+ value	11
Max⁻ value	9
Nom value	5

Min⁻ value	0
Max value	5
Max⁺ value	6
Max⁻ value	4
Nom value	3

If we consider all the above boundary values for both A & b with nominal value, then possible combinations for test cases will be as shown below:

A_{min}, b_{min}	A_{min}, b_{min+}	A_{min}, b_{min-}	A_{min}, b_{max}	A_{min}, b_{max+}
A_{min}, b_{max-}	A_{min}, b_{nom}	A_{min+}, b_{min}	A_{min+}, b_{min+}	A_{min+}, b_{min-}
A_{min+}, b_{max}	A_{min+}, b_{max+}	A_{min+}, b_{max-}	A_{min+}, b_{nom}	A_{min-}, b_{min}
A_{min-}, b_{min+}	A_{min-}, b_{min-}	A_{min-}, b_{max}	A_{min-}, b_{max+}	A_{min-}, b_{max-}
A_{min-}, b_{nom}	A_{max}, b_{min}	A_{max}, b_{min+}	A_{max}, b_{min-}	A_{max}, b_{max}
A_{max}, b_{max+}	A_{max}, b_{max-}	A_{max}, b_{nom}	A_{max+}, b_{min}	A_{max+}, b_{min+}
A_{max+}, b_{min-}	A_{max+}, b_{max}	A_{max+}, b_{max+}	A_{max+}, b_{max-}	A_{max}, b_{nom}
A_{max-}, b_{min}	A_{max-}, b_{min+}	A_{max-}, b_{min-}	A_{max-}, b_{max}	A_{max-}, b_{max+}
A_{max-},	A_{max-},	A_{nom}	A_{nom},	A_{nom},

b_{max-}	b_{nom}	b_{min}	b_{min+}	b_{min-}
A_{nom}, b_{max}	A_{nom}, b_{max+}	A_{nom} b_{max-}	A_{nom} b_{nom}	

and by considering only A at its extreme values, test cases will be designed as :

Test Case ID	Integer Variable		Expected Output
	A	b	
1	1	1	1
2	1	2	1
3	1	0	Input b is invalid
4	1	5	1
5	1	6	Input b is invalid
6	1	4	1
7	1	3	1
8	2	1	2
9	2	2	4
10	2	0	Input b is invalid
11	2	5	32
12	2	6	Input b is invalid
13	2	4	16
14	2	3	8
15	0	1	Input A is invalid
16	0	2	Input A is invalid
17	0	0	Both inputs A and b are invalid

18	0	5	Input A is invalid
19	0	6	Both inputs A and b are invalid
20	0	4	Input A is invalid
21	0	3	Input A is invalid
22	10	1	10
23	10	2	100
24	10	0	Input b is invalid
25	10	5	100000
26	10	6	Input b is invalid
27	10	4	10000
28	10	3	1000
29	11	1	Input As is invalid
30	11	2	Input As is invalid
31	11	0	Both inputs A and b are invalid
32	11	5	Input As is invalid
33	11	6	Both inputs A and b are invalid
34	11	4	Input As is invalid
35	11	3	Input As is invalid
36	9	1	9
37	9	2	81
38	9	0	Input b is invalid
39	9	5	59049
40	9	6	Input b is invalid

41	9	4	6561
42	9	3	729
43	5	1	5
44	5	2	25
45	5	0	Input b is invalid
46	5	5	3125
47	5	6	Input b is invalid
48	5	4	625
49	5	3	125

2.2.2 Equivalence Class (EVC) testing method:

As we know that input domain for testing is too large, so we can divide or partition it based on common feature or a class of data.

In this method, the input and the output domain is partitioned into mutually exclusive parts called **equivalence classes**.

These classes are identified such that each member of the class causes the same kind of processing and output to occur. Any one sample from a class is representative of entire class. Thus instead of testing every input, only one test case from each class can be executed and will be sufficient to find error.

Following are the advantages of this method:

- Without executing all test cases, it test complete testing domain.
- It reduces redundant test cases.
- It reduced total number of test cases.
- It reduces testing cost.
- It reduces time for testing.
- It requires less resource.

This method is used by performing following two steps:

1. **Identifying equivalence classes**

By assumption that is if the specification require exactly the same behavior for each

element in a class of values, then the program is likely to be considered such that it either correct or fail for each element in that class.

By considering each input and output conditions and partitioning it, two types of classes can always be identified as:

i. Valid equivalence class

The class considers all valid inputs

ii. Invalid equivalence class

The class considers all invalid inputs

Guidelines for forming equivalence classes:

- If the entire range of an input won't be treated in the same manner then the range should be split into 2 or more classes. e.g. a[1,10] then invalid class={a<0},{a<1}{a>10} and valid class={1<=a<=10}
- If program handles each valid input differently then define one valid class per input. e.g {1<=a<=10} {1<=b<=10} {1<=c<=10}{a>b, a>c,b#c}
- Boundary value analysis can help in identifying the classes.

  e.g. 0<=a<=10 then, one valid class(0<a<10) & two invalid classes can be design(a<0 and a>10)
- If input variable can identify more than one category then for each category, we can make equivalence class. e.g. if input is character then we can make three valid classes as it can be number, alphabet or special characters.
- For range design one valid class where input must be between the range, one invalid class where few invalid inputs, one invalid class where more invalid inputs. E.g. requirements is maximum for 4 items order can be placed, classes are: one valid class(1<=item<=4), one invalid class(item>4) and one invalid class(item<1)
- If input condition specifies 'must be' situation(first character must be capital or letter), identify a valid class(it is capital/letter) and an invalid class(not capital/letter)
- Classes can be of the output desired in the program.
- Look for membership of an input condition in a set and identify valid and invalid class. E.g. (valid code is HR,AP or WB then one valid class for each code and one invalid class for none of these)

If input match a set of values and each case will deal with differently, identify valid class for each element and only one invalid class for values outside the set. E.g. discount code for preferred customer is P, R for reduced rate or N for none then valid class code=P, valid class code=R, valid class code=N and invalid class code is none of P,R nor N.

- If an element of a class will be handled differently than the others, divide the class to create class with only these elements and class with none of these elements. E.g. Bank account balance may be from Rs. 0 to Rs 10lakh and balance of Rs 1000 or more, no service charges then valid class: 0<=balance<1000), valid class:1000<=balance<=10lakh, invalid class: balance>10lakh

2. Design test cases using equivalence classes

Guidelines for designing test cases are:

- Assign a unique identification number to each equivalence class
- Write a new test case covering as many of the uncovered valid equivalence classes until all valid classes have been covered.
- Write a test case that covers, one and only one of the uncovered equivalence classes, until all invalid classes have been covered.

Example 1: A program reads an integer number(N) within the range [1,100] and determines whether it is a prime number or not.

Solution:

Valid and Invalid classes:

I1={<N>:1<=N<=10} I2={<N>:N<1} I3={<N>:N>10}

Test Case Id	A	Expected output	Classes Covered
1	1	Not a Prime Number	I1
2	10	Not a Prime Number	I1
3	2	Prime Number	I1
3	0	Invalid input	I2
4	11	Invalid input	I3

We also can define classes as:

I1={<N>:1<=N<=10} I2={<N>:N==0} I3={<N>:N<0} I4={<N>:N>10}

Example 2: A program reads three numbers A,B and C with a range[1,50] and prints the largest number. Design test cases using equivalence class testing technique.

Solution:

Divide domain of input as valid input values and invalid values as:

I1={<A,B,C>:1<=A<=50} I2 ={<A,B,C>:1<=B<=50}

I3={<A,B,C>:1<=C<=50} I4={<A,B,C>:A<1}

I5={<A,B,C>:A>50} I6={<A,B,C>:B<1}

I7={<A,B,C>:AB>50} I8={<A,B,C>:C<1} and I9={<A,B,C>:C>50}

Test cases can be designed as below:

Test Case Id	A	B	C	Expected output	Classes Covered
1	13	25	36	C is largest	I1,I2,I3
2	0	13	45	Invalid input	I4
3	51	34	17	Invalid input	I5
4	29	0	18	Invalid input	I6
5	36	53	32	Invalid input	I7
6	27	42	0	Invalid input	I8
7	33	21	51	Invalid input	I9

Another set is also possible for valid values:

I1={<A,B,C>:A>B,A>C} I2={<A,B,C>:B>A,B>C} I3={<A,B,C>:C>B,C>A}

I4={<A,B,A=B,A#C} I5={<A,B,C>:A=C ,C#B} I6={<A,B,C>:B=C,A#B}

I7={<A,B,C>:A=B=C}

Test cases can be designed as below:

Test Case Id	A	B	C	Expected output	Classes Covered
1	25	13	13	A is largest	I1,I5
2	25	40	25	B is largest	I2,I4
3	24	24	37	C is largest	I3,I6
4	25	25	25	All are equal	I7

2.2.3 Decision Table-Based Testing:

Boundary value analysis and equivalence class partitioning do not consider combinations of input conditions, but consider each input separately.

A decision table is a good way to test the system behavior for different combination of inputs .It is useful to determine the test scenarios for complex business logic. It is a systematic approach where the different input combinations and their corresponding system behavior are captured in a tabular form.

This table helps us to deal with different combination of inputs with their associated outputs. Also, it is known as the cause-effect table.

To use this method, following steps are done:

1. Formation of Decision Table:

A decision table is formed with the following four components:

i. **Condition stub** is a list of input conditions for which the complex combination is made.

ii. **Action stub** is a list of resulting actions, which will be performed if a combination of input condition is satisfied.

iii. **Condition entry** is a specific entry in the table corresponding to input conditions mentioned in the condition stub.

When the condition entry takes only two values –TRUE or FALSE, then a table is called **Limited Entry Decision Table**. In Limited Entry Decision Table, condition entry, which has no effect whether it is TRUE or FALSE, is called

Don't Care state or immaterial state (Represented by |) and it doesn't affect the resulting action.

When the condition entry takes several values, it is called **Extended Entry Decision Table.**

When we enter TRUE of FALSE for all input conditions for a particular combination, then it is called a **Rule** and it defines which combination of conditions produces the resulting action.

iv. **Action entry** it the entry in the table for resulting action to be performed when one rule is satisfied. Action entry is denoted by 'X' in the table.

How to develop a decision table

The guidelines to develop a decision table for a problem are discussed below:

- List all actions that can be associated with a specific procedure(module).
- List all conditions (or decision made) during execution of the procedure.
- Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.
- Define rules by indicating what action occurs for a set of conditions.

2. Test Case Design Using Decision Table:

For designing test cases from a decision table, following interpretations should be done:

Interpret condition stubs as the input for the test case.

Interpret action stubs as the expected output for the test case.

Rule, which is the combination of input conditions, becomes the test case itself.

If there are K rules over n binary conditions, there are at least K test cases and the most 2^n test cases.

- **Example 1:** A program calculates the total salary of an employee with the conditions that is the working hours are less than or equal to 8, then give normal salary. The hours over 8 on normal working days are calculated at the rate of 1.35 of the salary. However, on holiday or Sundays, the hours are calculated at the rate of 2.00 times of the salary. Design test cases using Decision Table testing.

Solution:

Step 1: Design a decision table.

List all conditions

C1: Is working hours >8? Or <=8

C2: is Holidays or Sundays? Or is normal days

List all actions

A1: Normal Salary

A2:1.25 of the salary

A3:2.00 of the salary

Possible combinations as Rules:

R1: If Working hours <= 8 and no holidays or Sundays then normal salary

R2: Else if working hours >8 and no holidays or Sundays then 1.25 of salary

R3: Else if holidays or Sundays irrespective of hours, 2.00 of salary

Decision Table is:

Entry

		R1	R2	R3
Condition Stubs	C1:Working hours>8	F	T	|
	C2:Holidays or Sundays	F	F	T
Action Stubs	A1:Nomral salary	X		
	A2:1.25 of salary		X	
	A3:2.00 of salary			X

Step 2: Test Cases designed from the decision table:

Test Case Id	Working hours	Day	Expected Result
1.	8	Monday	Normal Salary
2.	10	Wednesday	1.25 of salary
3.	10	Sunday	2.00 of salary

- **Example 2:** A wholesaler has three commodities to sell and has three types of

customers. Discount is given as per the following procedure:

(a) For DGS & D orders, 10 % discount is given irrespective of the value of order.

(b) For orders of more than ₹50,000, agent gets a discount of 15% and the retailer gets a discount of 10%.

(c) For orders of ₹20,000 or more and upto ₹50,000, agent gets 12% and retailer gets 8% discount.

(d) For orders of less than ₹20,000, agent gets 8% and the retailer gets 5% discount.

(e) The aforementioned rules don't apply to the furniture items wherein a flat rate of 10% discount is admissible to all customers irrespective of the value of the order.

Design test cases for this system using decision table testing.

Solution:

Step 1: Design a decision table.

List all conditions

C1: is Customer type=DGS & D ?

C2: is Customer type=agent?

C3: is Customer type=retailer?

C4: Order value>₹50,000

C5:Order value >=20,000 and <=50,000

C6:Orcder value<20,000

C7:Is items=furniture?

List all actions

A1: Discount of 5%

A2:Discount of 8%

A3:Discount of 10%

A4:Discount of 12 %

A5:discount of 15%

Possible combinations as Rules:

R1: If customer type=DGS & D, discount is 10% irrespective of order value and item should not be furniture

R2: Else If customer type=agent and order value >50,000, discount is 15% and item should not be furniture

R3: Else If customer type=retailer and order value >50,000, discount is 10% and item should not be furniture

R4:Else If customer type=agent and order value >=20,000 and <=50,000, discount is 12% and item should not be furniture

R5: Else If customer type=retailer and order value >=20,000 and <=50,000, discount is 8% and item should not be furniture

R6:Else If customer type=agent and order value <20,000, discount is 8% and item should not be furniture

R7: Else If customer type=retailer and order value <20,000 ,discount is 5% and item should not be furniture

R8: Else If item is furniture then 10% discount will be given to all customer irrespective of the value of the order.

Decision Table is:

Entry

		R1	R2	R3	R4	R5	R6	R7	R8
Condition Stubs	C1: DGS &D	T	F	F	F	F	F	F	|
	C2:Agent	F	T	F	T	F	T	F	|
	C3:Retailer	F	F	T	F	T	F	T	|
	C4:Order value>50,000	|	T	T	F	F	F	F	|
	C5:Order value>=20,000 and <=50,000	|	F		T	T	F	F	|
	C6:Order value <20,000	|	F	F	F	F	T	T	|
	C7:Furniture	F	F	F	F	F	F	F	T
Action	A1:5% discount							X	

Stubs	A2:8% discount					X	X		
	A3:10% discount	X		X					X
	A4:12 % discount				X				
	A5:15% discount		X						

Step 2: Test Cases designed from the decision table:

Test Case Id	Type of Customer	Order value(₹)	Item Furniture?	Expected Result
1.	DGS & D	51000	N	10% discount
2.	Agent	53000	N	15% discount
3.	Retailer	67000	N	10% discount
4.	Agent	25000	N	12% discount
5.	Retailer	30000	N	8% discount
6.	Agent	18000	N	8% discount
7.	Retailer	15000	N	5% discount
8.	Agent	34000	Y	10% discount

Note: in case 8, customer type & order value doesn't matter.

2.2.3.1 Expanding Immaterial Cases in Decision Table:

Immaterial cases(I), which have been shown in the table are don't care conditions. These conditions mean that the value of a particular condition in the specific rule doesn't make a difference whether it is TRUE or FALSE. However we should always test both the values of a don't care condition. So the rules in the table can be expanded. Consider the decision table of previous example 1 as given below.

		Entry		
		R1	R2	R3
Condition Stubs	C1:Working hours>8	F	T	|
	C2:Holidays or Sundays	F	F	T
Action Stubs	A1:Nomral salary	X		
	A2:1.25 of salary		X	
	A3:2.00 of salary			X

The immaterial test case in Rule 3 of the above table can be expanded by taking both T and F values of C1 and expanded decision table is shown below:

		Entry			
		R1	R2	R3-1	R3-2
Condition Stubs	C1:Working hours>8	F	T	F	T
	C2:Holidays or Sundays	F	F	T	T
Action Stubs	A1:Nomral salary	X			
	A2:1.25 of salary		X		
	A3:2.00 of salary			X	X

The test cases derived from the above expanded decision table are given below:

Test Case Id	Working hours	Day	Expected Result
1.	8	Monday	Normal Salary
2.	10	Wednesday	1.25 of salary
3.	10	Sunday	2.00 of salary
4.	5	Sunday	2.0 f salary

2.3 White-box Testing:

White-box testing is a software testing method in which the internal structure/ design/implementation of the software being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Often developers use this technique to test their own design and code.

It is a dynamic testing technique. It is also known as glass-box, open box, logic driven, clear box, structural or development testing. It is a complementary to black-box testing.

It ensures that the internal parts of the software are adequately tested.
As it is based on the inner workings of an application and revolves around internal testing, the term "White-Box" was used because of the see-through box concept.

- **Need of White-box testing.**

It is needed for the following reasons:

⇨ It is used for testing the module for initial stage testing.

⇨ There may be portions in the code, which are not checked when executing functional

test cases, but these will be executed and tested by white-box testing.

- ⇨ Errors that come from the design phase will also be reflected in the code and therefore, we must execute white-box test cases for code verification (Unit verification).
- ⇨ We often believe that a logical path is not likely to be executed but, in fact, it may be executed on a regular basis. White-box testing explores these paths too.
- ⇨ Some typographical errors are not observed and go undetected and are not covered by black box testing techniques. White-box testing techniques help to detect these errors.

- **Advantages of White Box Testing:**
  - ⇨ Code optimization by finding hidden errors.
  - ⇨ White box tests cases can be easily automated.
  - ⇨ Testing is more thorough as all code paths are usually covered.
  - ⇨ Testing can start early in SDLC even if GUI is not available.

- **Disadvantages of White Box Testing:**
  - ⇨ Expensive as one has to spend both time and money to perform white box testing.
  - ⇨ Every possibility that few lines of code are missed accidentally.
  - ⇨ In-depth knowledge about the programming language is necessary to perform white box testing.

2.4 Classification of white box testing:

White-box testing can be classified as structural and static testing.

2.4.1 Structural Testing:

Structural testing considers the program code, and test cases are designed based on the logic of the program such that every element of the logic is covered. This type of testing requires knowledge of the code, so, it is mostly done by the developers. It is more concerned with how system does it rather than the functionality of the system. It provides more coverage to the testing. It can be used on different levels such as unit testing, component testing, integration testing, functional testing etc. The following are different structural testing techniques or coverage which is also known as white-box

(1) Logic Coverage Criteria:

The basic forms of logic coverage are:

- **Statement Coverage:**

The aim of this form is, all the statements of the module are executed or tested at least once to notify every bug. Using this technique we can check what the source

code is expected to do and what it should not. It can also be used to check the quality of the code and the flow of different paths in the program. The main drawback of this technique is that we cannot test the false condition in it. (Statement coverage = No of statements Executed/Total no of statements in the source code * 100)
Consider the below code 1:

```
scanf("%d",&x);
scanf("%d",&y);
while(x!=y)
{
if(x>y)
x=x-y;
else
y=y-x;
}
printf("x=",x);
printf("y=",y);
```

code 1

If we want to cover every statement in this, then the following test cases must be designed:

Test case 1: x=y=n, where n is any number e.g. x=y=5
In this case, while loop is skipped and hence all statements of while loop are not executed. Statement coverage= 4/11*100=36.36%

Test case 2: x=n, y=n', where n and n' are different numbers. e.g. x=5,y=6
In this case, while loop is executed, but not all statements of a loop.
Statement coverage=10/11*100=90.90%
So two more test cases are designed as:

Test case 3: x>y, e.g. x=6,y=5, statement coverage=9/11*100=81.81%

Test case 4: x<y, e.g. x=5,y=6, statement coverage=10/11*100=91.91%

If we executes only Test case 3 and 4, it is sufficient to execute all statements of the code, and Test case 1 will never be tested and error will go undetected.
Hence this form is not a sufficient criteria for logic coverage.

- **Branch Coverage:**
It says that each decision takes on all possible outcomes(True or False) at least once.

i.e. each branch or edge direction must be covered at least once. It doesn't provide a good coverage from different conditions that lead from one node to another node as it covers that branch only once. If we consider the previous **code 1**, while and if statements have two outcomes: True and False. So, test cases must be designed such that both outcomes for while and if statements are tested. The test cases are designed:

Test case 1: x=y

Test case 2: x!=y

Test case 3: x>y

Test case 4: x<y

Consider following code 2:

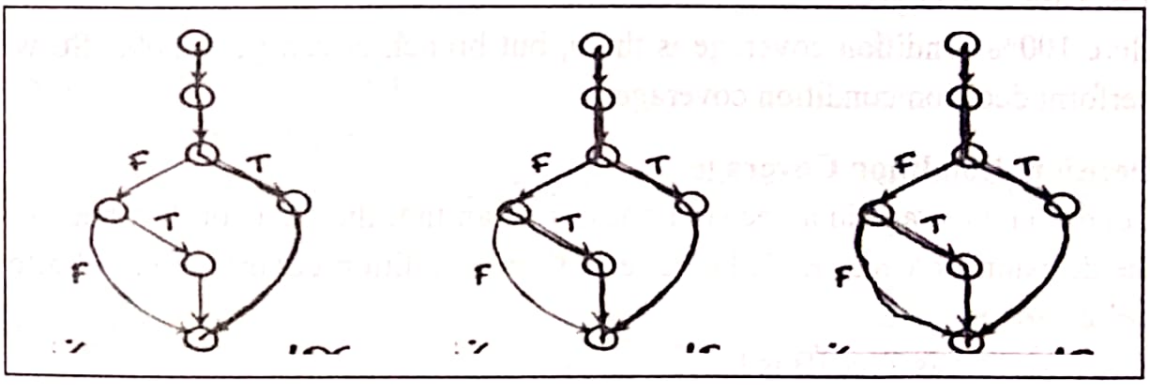
```
scanf("%d",&x); scanf("%d",&y);
int r=x+y;
if(r>0) printf("Result:",pass);
else if(r<0) printf("Result:",fail);
```

code 2

Test case 1: x=5,y=3, it covers 25%

Test case 2: x=-3,y=-2, it covers 75%

Test case 3:x=3,y=-3, it covers 100%



With all three above test cases it covers all branches as shown in above figure.

- **Condition Coverage:**

This form says that each condition in a decision takes on all possible outcomes at least once. It aims to test individual conditions with possible different combination of Boolean input for the expression. It is modification of Decision coverage but it provides better coverage. It will reveal how the variables or subexpressions in the

conditional statement are evaluated. In this coverage expressions with logical operands are only considered.

e.g. while ((I<=5 && J<count))
in this loop statement, two conditions are there. So, test cases should be designed such that both the conditions are tested for True and False outcomes. The following test cases are designed:
Test case 1: I<=5,J<count
Test case 2: I>5,J>count
Consider following code 3:

```
scanf("%d",&x); scanf("%d",&y);
if((x==0) || (y>0))
  y=y*x;
else
  x=y+2;
printf("%d",x);
printf("%d",y);
```

code 3

Test case 1: x=0,y=-5
Test case 2: x=5,y=5
Here 100% condition coverage is there, but branch coverage is 50%. So we need to perform decision/condition coverage.

- **Decision/ Condition Coverage:**
Condition coverage in a decision doesn't mean that the decision has been covered. If the decision if(A && B) is being tested, the condition coverage would allow one to write two test cases:
Test case 1: A is True, B is False
Test case 2: A is False, B is True

However, these test cases would not execute THEN clause. The obvious way out of this dilemma is a criterion called decision/condition coverage. It requires sufficient test cases such that each condition in a decision takes on all possible outcomes at least once, each decision takes on all possible outcomes at least once, and each point of entry is invoked at least once.

For code 3, we need one more test case as: Test case 3: x=5,y=-5

Consider following code 4:

```
scanf("%d",&x);
scanf("%d",&y);
if(x==0 || y==0)
printf("Zero");
```

code 4

Here, two conditions are there: x==0 and y==0, then test cases designed are:
Test case 1: x==0,y=5
Test case 2: x=5,y=0

- **Multiple Condition Coverage:**

In this form, all possible combinations of the possible outcomes of conditions are tested at least once. Objective of this form, is to test important combination of conditions with less testing cost.
Each condition should affect the decision outcome independently.
For n conditions, 2n test cases can be designed.
e.g. for two conditions A & B, the test cases can be designed as:
Test case 1: A =True, B=True
Test case 2: A=True, B=False
Test case 3: A=False, B=True
Test case 4: A=False, B=False
Consider following table:
Take condition a, b and c as independently & find where it affects on outcome.

a && b && c

Test Case	a	b	c	OutCome
1	True	True	True	Ture
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
6	False	False	False	False

From the above table, we can say that for the condition a, test case 1 & 5 affects on outcome. Similarly for the condition b and c, test cases 1, 3 and 1, 2 affects on

outcomes. So finally, test cases 1, 2, 3 & 5 required to be executed.

For code 4, test case can be designed as:

Test case 1: x=0,y=0

Test case 2: x=0,y=5

Test case 3: x=15,y=0

Test case 4: x=15,y=5

(2) Basis Path Testing:

This method is the oldest structural testing technique. It is based on the control structure of the program. Based on the control structure, a flow graph is prepared and all the possible paths can be covered and executed during testing. This method is useful for detecting more errors. However, if a program contains loops may have an infinite number of possible paths and it is not practical to test all the paths. Hence some criteria should be devised such that selected paths are executed for maximum coverage of logic. The effectiveness of path testing gets reduced with the increase in size of software under test.

Steps for effectiveness of path testing:

1. **Draw the flow graph based on program structure.**

Flow graph can be prepared as directed graph using the following notations:

Node: It represents one or more procedural statements. It is denoted by a circle with label or number.

Edges or links: They represent the flow of control in a program. It is denoted by arrow and it must terminate at the node.

Decision node: A node with more than one arrow leaving it is called a decision node.

Junction node: A node with more than one arrow entering it is called a junction node.

Regions: area bounded by edges and nodes are called regions. When counting the regions, the area outside the graph is also considered a region.

Path A path is a sequence of instructions or statements that starts at an entry, junction or decision node and ends at another or possibly the same, junction, decision or exit. It may go through several junctions, processes or decisions, one or more times.

Segment: A single process that lies between two nodes(junction-process-junction, junction-process-decision,decision-process-junction,decision-process-decision).

The smallest segment is a link.

Path segment: A path segment is a succession of consecutive links that belongs to some path.

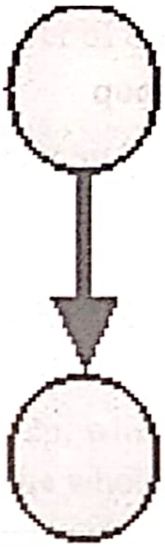
Length of path: It is measured by number of links in a path and not by the number of instructions or statements executed along the path. Or number of nodes traversed in a path is known as length of path.

Independent path: It is any path through the graph that introduces at least one new set of processing statement or new condition. It must move along at least one edge that has not been traversed before the path is defined.

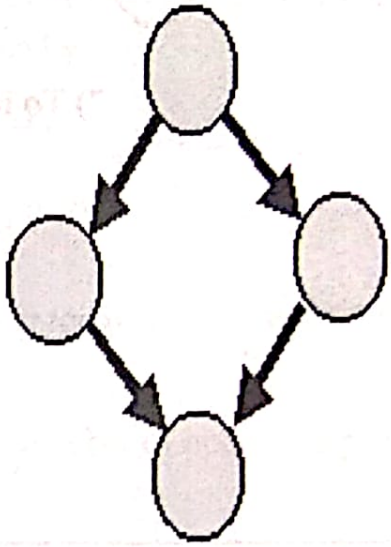
Sequential statements having no conditions or loops can be merged in a single node, hence the flow graph is also known as **decision-to-decision**-graph or **DD** graph.

Some fundamental graphical notations are shown as below:

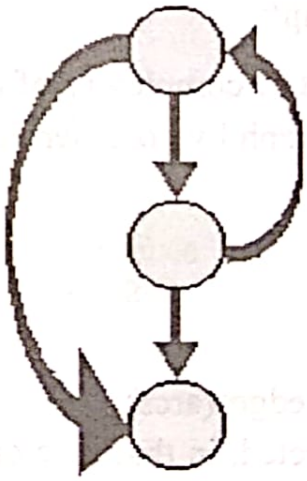
1) To indicate a Sequence



2) To indicate IF THEN ELSE

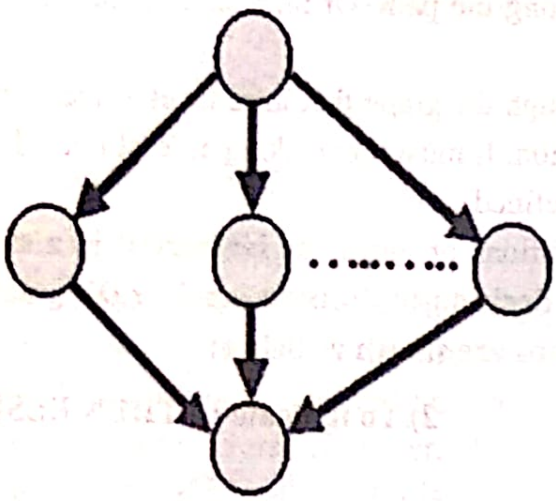


3) To indicate WHILE Loop

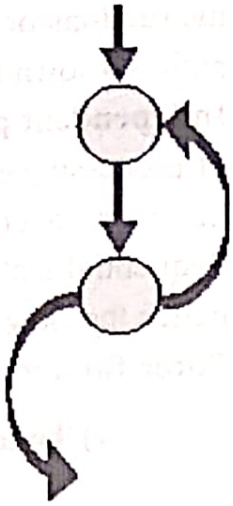


4) To indicate Do While (Repeat Until) Loop

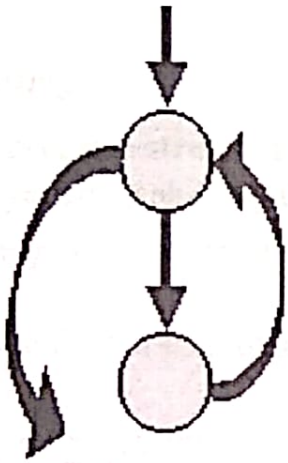
5) To indicate SWITCH Statement



6) To indicate a Post Test FOR Loop



7) To indicate Pre Test FOR Loop



2. Determine the cyclomatic complexity of the flow graph.

McCabe has given a measure for the logical(cyclomatic) complexity of a program by considering the number of paths in the control flow graph by considering independent paths only.

The number of independent paths is given by

$$V(G) = e - n + 1,$$

Where n is the number of nodes and e is the number of edges(arcs).
It may be possible that the graph is not strongly connected, in that case one more edge is added from the last node to the first node of the graph, therefore, the number of

independent paths is given by

$$V(G) = e - n + 2$$

This is called the cyclomatic number of a program.
We can calculate the cyclomatic number only by knowing the number of choice points (decision nodes) **d** in the program. It is given by

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

This is also kwon as Miller's theorem. We assume that K-way decision point contributes for k-1 choice points.

The program may contain several procedures also and represented as separate flow graphs. If p is the number of graphs and e and n are referred to as the whole graph, the cyclomatic number of the whole graph is then given by

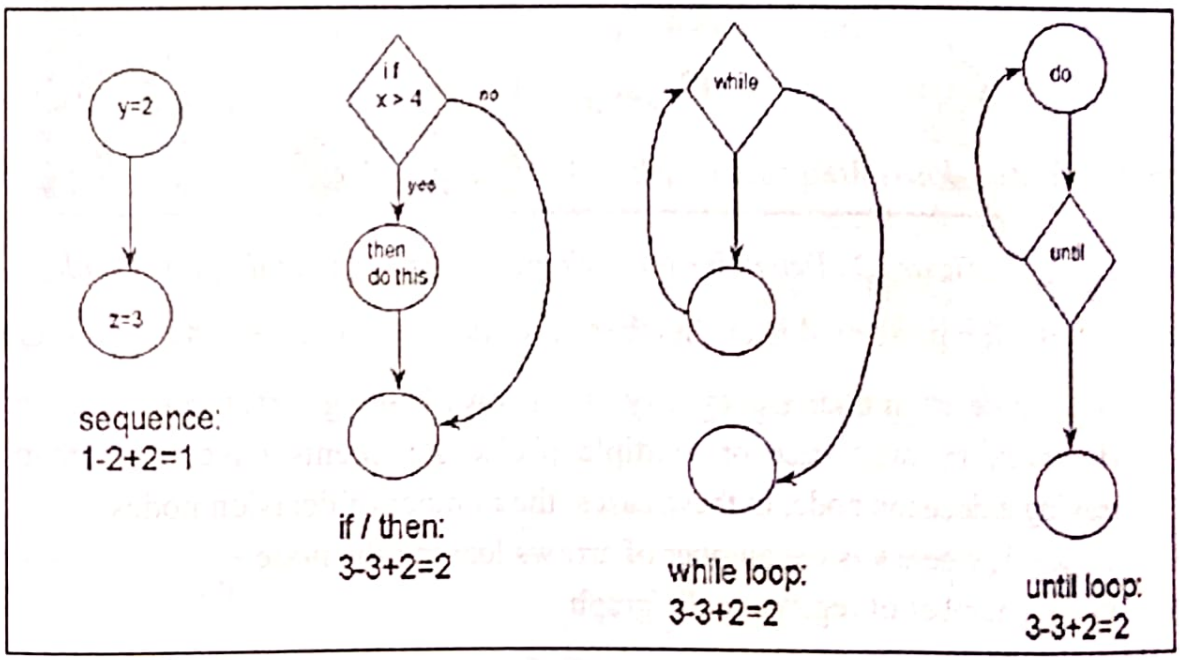
$$V(G) = e - n + 2P,$$

And Miller's theorem becomes

$$V(G) = d + p$$

Cyclomatic complexity number can be derived through any of the following three formulae:

i. V(G) = e – n + 2p, where e is number edges, n is number of nodes and p is number of components in the whole graph.



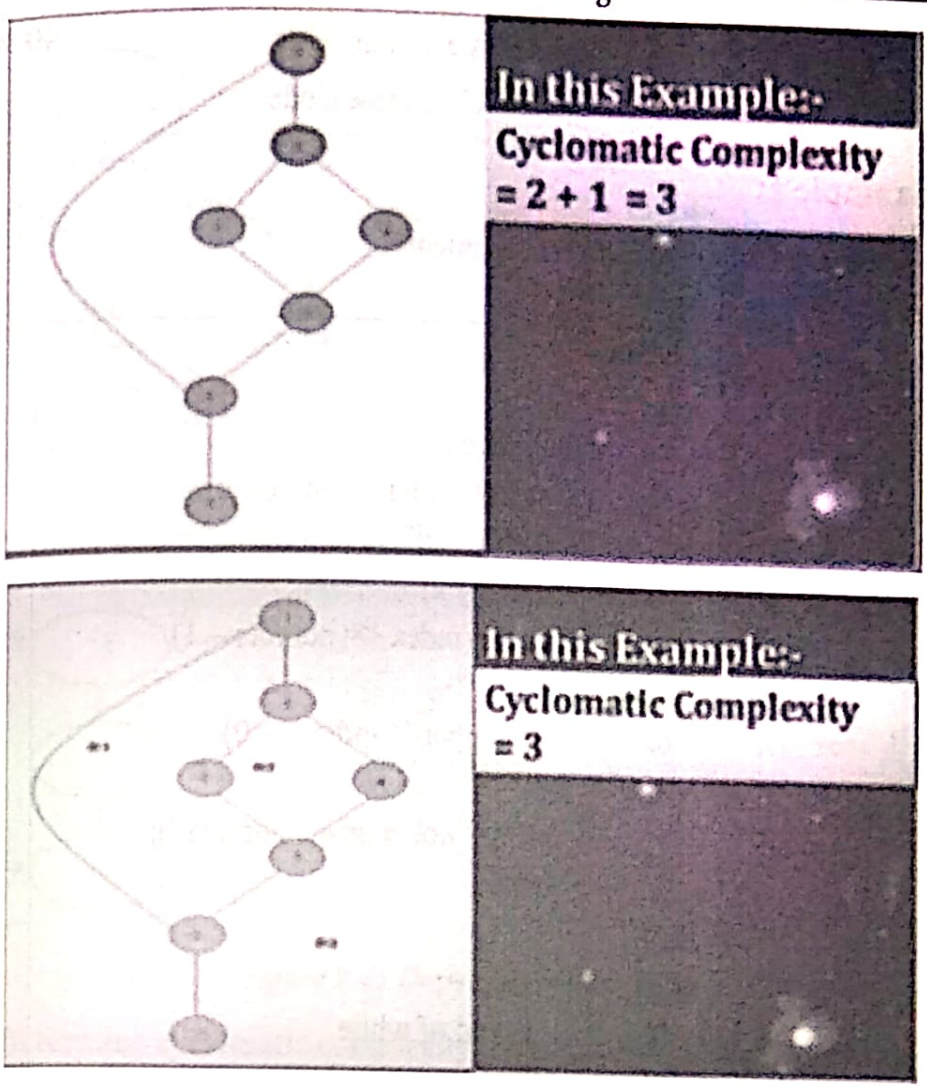


Figure 2.3 : cyclomatic complexity Example

3. **Determine the basis set of independent paths.**

4. **Design test cases corresponding to each independent path.(each path is executed).** Means number of test cases will be equivalent to cyclomatic complexity.

Cyclomatic Complexity	Risk Evaluation	Probability Of Bad fix
1-10	Low risk, Testable Code	5 %
11-20	Moderate Risk	10 %
21-50	High Risk	30 %

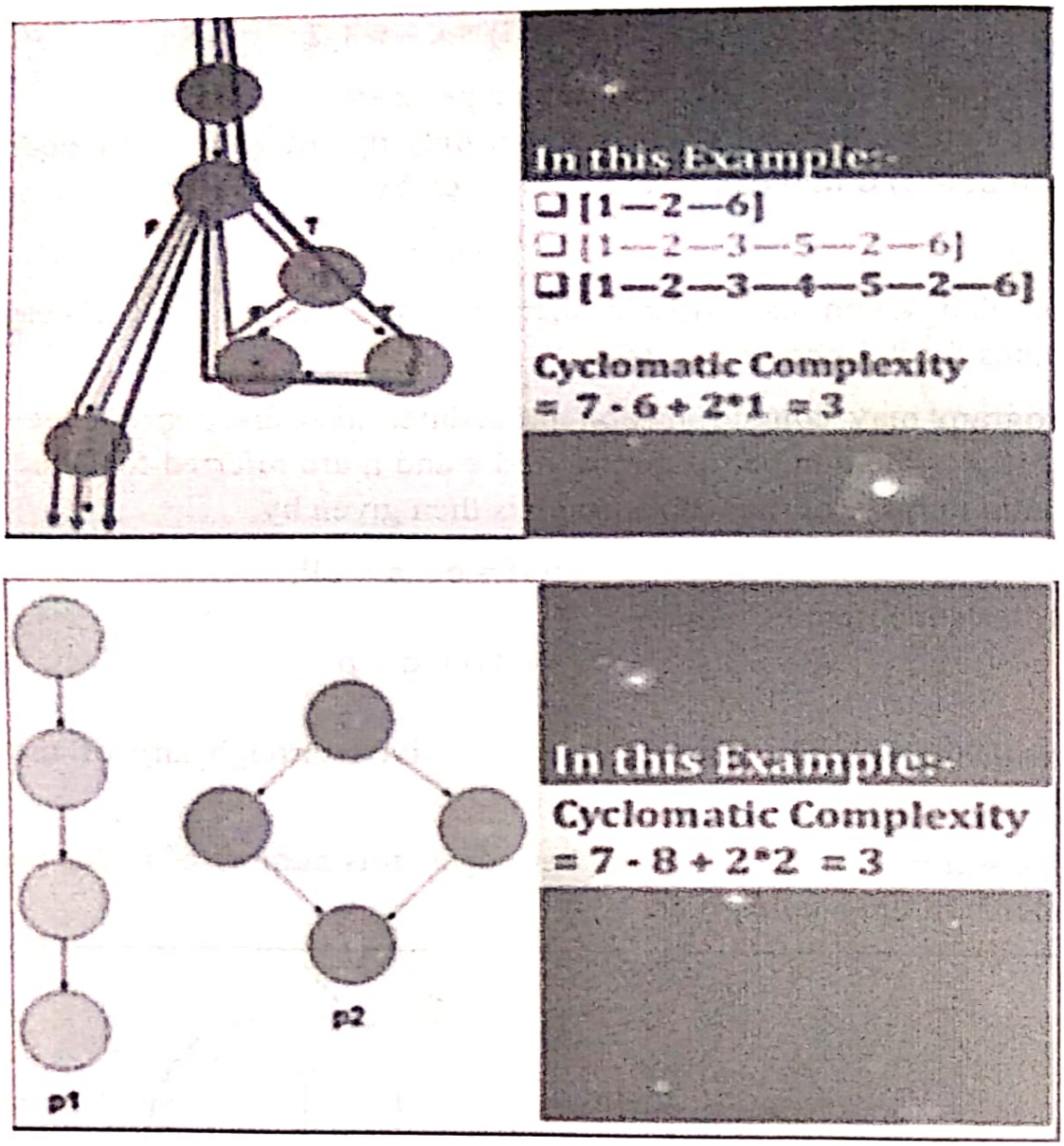


Figure 2.2 : Determine the cyclomatic complexity of the flow graph

i. V(G) = d + p, where d is the number of decision(predicate) nodes in the g

When a decision node has exactly two arrows leaving it, we count it as However, in swith case or multiple if-else statements have more tha leaving a decision node, in these cases, the number of decision nodes d = k – 1, where k is the number of arrows leaving the node.

ii. V(G) = number of regions in the graph

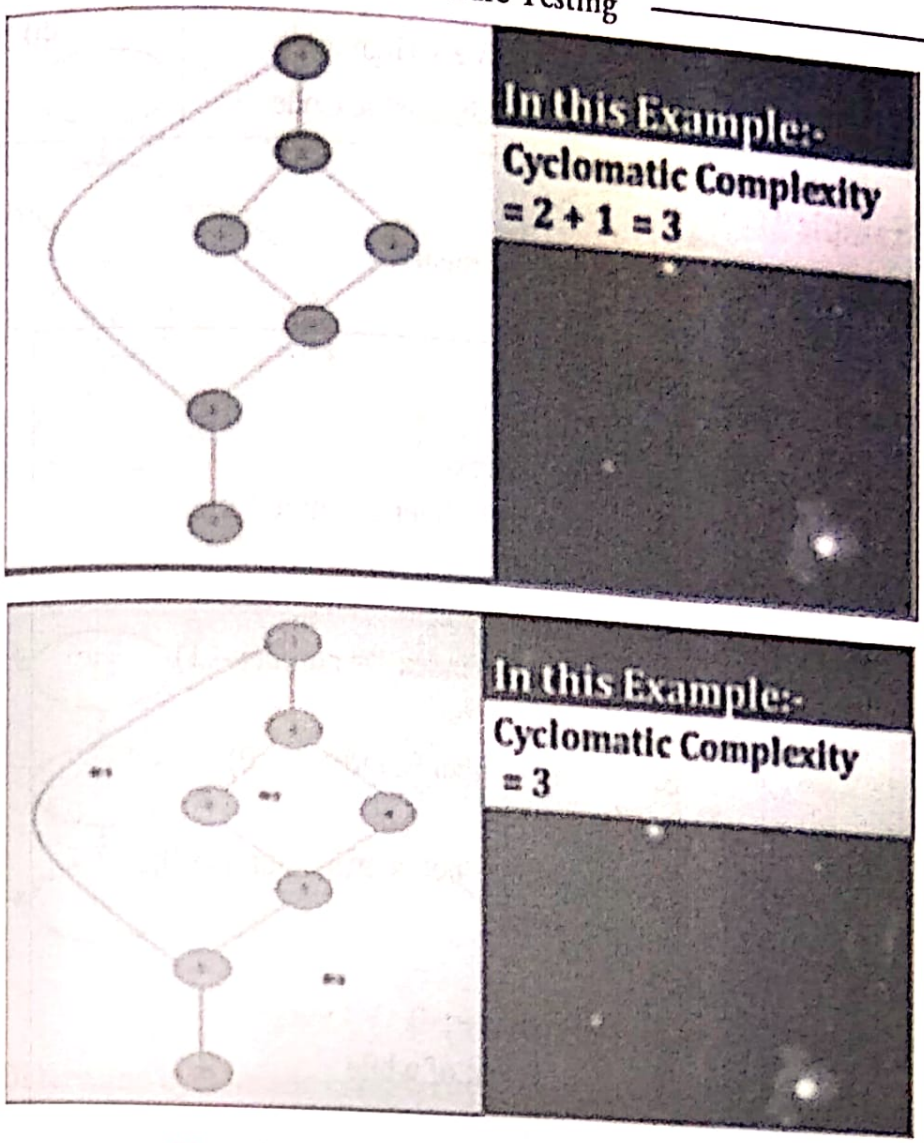


Figure 2.3 : cyclomatic complexity Example

3. **Determine the basis set of independent paths.**

4. **Design test cases corresponding to each independent path.(each path is executed).** Means number of test cases will be equivalent to cyclomatic complexity.

Cyclomatic Complexity	Risk Evaluation	Probability Of Bad fix
1-10	Low risk, Testable Code	5 %
11-20	Moderate Risk	10 %
21-50	High Risk	30 %

> 50	Very High Risk, Untestable Code	40 %

➢ Example 1:

Consider the following program segment:

```
main()
{
int number, index;
1.      printf("Enter a number");
2.      scanf("%d", & number);
3.      index = 2;
4.      while (index <= number – 1)
5.      {
6.      if(number % index == 0)
7.      {
8.      printf("not  a prime number");
9.      break;
10.     }
11.     index++;
12.     } // end of while
13.     if(index == number)
14.     printf("Prime number");
15.     }  //end of main
```

Solution:

Step 1: Draw control flow graph:

- Put line number on the execution statements of the program, after declaring variables, if no variables have been initialized. Otherwise, start from the statements where a variable has been initialized. See in above code.
- Put the sequential statements in one node. e.g. Statements 1, 2 and 3 have been put inside one node.
- Put the edges between the nodes according to their flow of execution.
- Put alphabetical or numerical numbering on each node like A, B, C or 1, 2, 3 etc.

For the above code the control flow graph without and with region is given below:

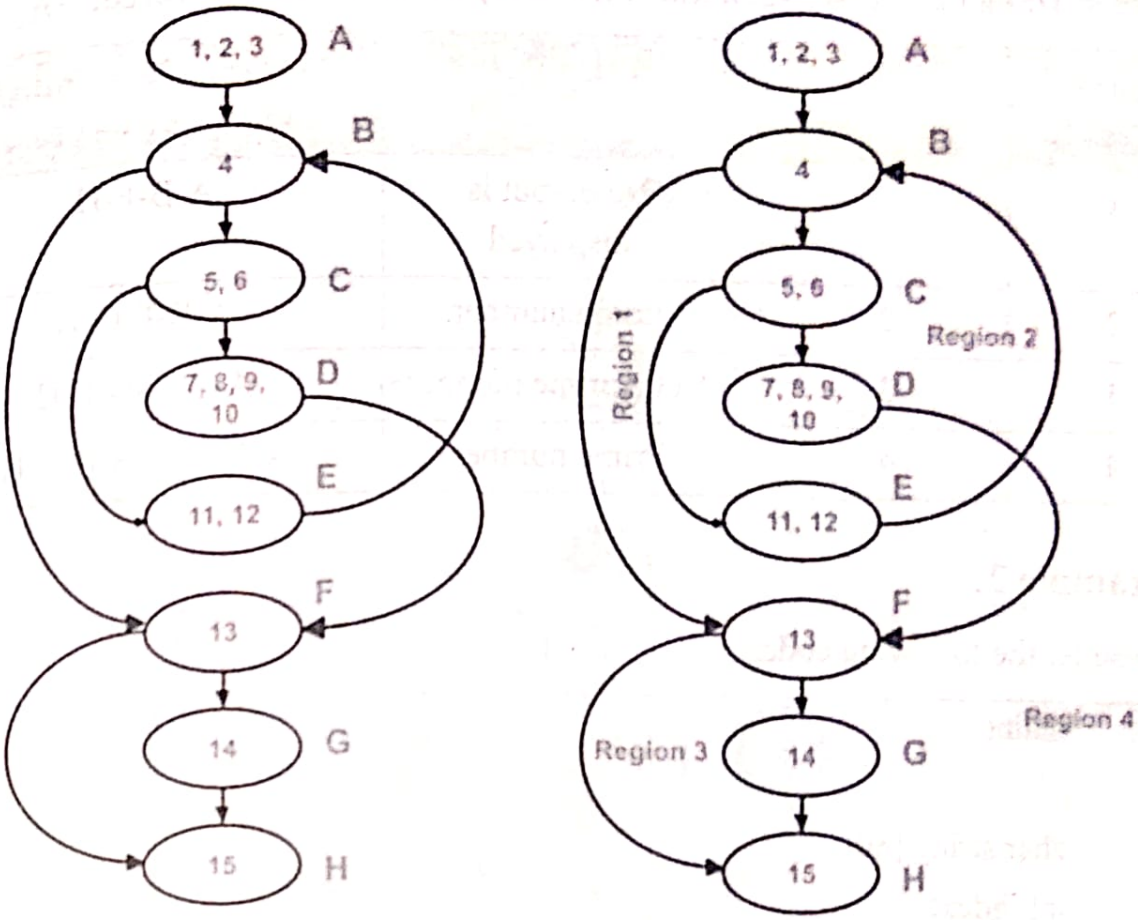


Figure 2.4 : Draw control flow graph

Step 2: Determine cyclomatic complexity using any one of the following ways.

(i) $V(G) = e - n + 2P = 10 - 8 + 2(1) = 4$

(ii) V(G) = number of decision(predicate) nodes + 1 , here node B, C and F are decision nodes

$V(G) = 3 + 1 = 4$

(iii) V(G) = number of regions= 4

Step 3: Determine basis independent paths

Number of independent paths= cyclomatic complexity =4

Independent paths are:

(i) A-B-F-H

(ii) A-B-F-G-H

(iii) A-B-C-E-B-F-G-H

(iv) A-B-C-D-F-H

Step 4: Design test cases such that each independent path is covered.

Test Case ID	Input Number	Expected Result	Independent paths covered by test case
1	1	No output is displayed	A-B-F-H
2	2	Prime number	A-B-F-G-H
3	3	Not a prime number	A-B-C-D-F-H
4	4	Prime number	A-B-C-E-B-F-G-H

➢ Example 2:

Consider the following code:

```
main()
{
char string [80];
int index;
1.  printf("Enter the string for checking its characters");
2.  scanf("%s", string);
3.  for(index = 0; string[index] != '\0'; ++index)
4.    {
5.     if(string[index] >= 'A' && string[index] <='9')
6.     printf("%c is a digit", string[index]);
7.     else  if(string[index] >= 'A' && string[index] <'Z') ||
8.           (string[index] >= 'a' && string[index] <'z')
9.     printf("%c is an alphabet", string[index]);
10.    else
11.    printf("%c is a special character", string[index]);
12.    }
13. }
```

Solution:

Step 1: Control flow graph of the above code is shown below:

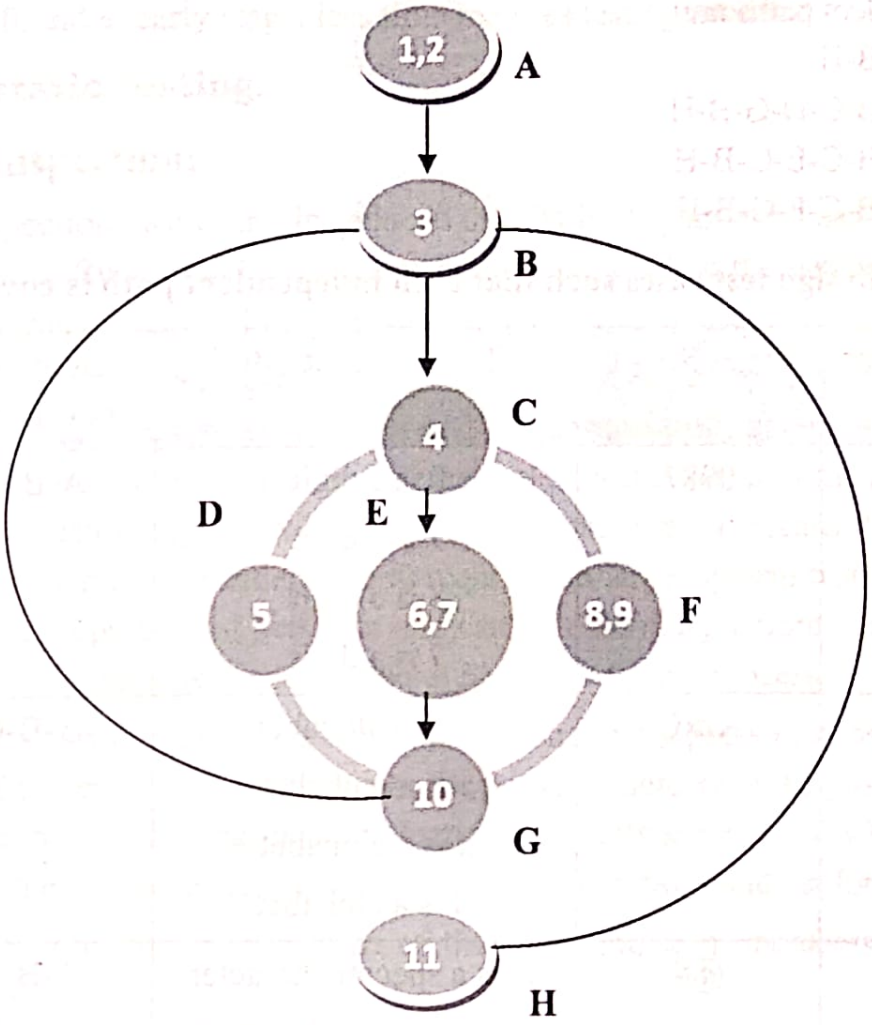


Figure 2.5 : Draw control flow graph

Step 2: Determine cyclomatic complexity using any one of the following ways.

(i) V(G) = e – n + 2P = 10 – 8 + 2(1) = 4

(ii) V(G) = number of decision(predicate) nodes + 1 , here node B and C are decision nodes.

Here, Node C is a multiple if-then-else; so for finding out the number of predicate nodes for this case, follow the formula:

Number of predicated nodes=number of links out of main node -1=3-1=2(For node C)

V(G) = 3 + 1=4

(iii)V(G) = number of regions= 4

Step 3: Determine basis independent paths

Number of independent paths= Cyclomatic complexity =4

Independent paths are:

i. A-B-H
ii. A-B-C-D-G-B-H
iii. A-B-C-E-G-B-H
iv. A-B-C-F-G-B-H

Step 4: Design test cases such that each independent path is covered.

Test Case ID	Input String	Expected Result	Independent paths covered by test case
1	0987	0 is a digit 9 is a digit 8 is a digit 7 is a digit	A-B-C-D-G-B-H A-B-H
2	AzxG	A is a alphabet z is a alphabet x is a alphabet G is a alphabet	A-B-C-E-G-B-H A-B-H
3	@#	@ is a special character # is a special character	A-B-C-F-G-B-H A-B-H

2.5 Static Testing:

Static testing techniques don't execute the software and don't require the bulk of test cases. This type of testing is also known as non-computer based testing or human testing. It reveals errors which are not shown by dynamic testing. It can be applied for most of the verification activities.

It is done to avoid errors at an early stage of development as it is easier to find sources of failures then failures themselves. It checks the software product at each SDLC stage for conformance with the required specifications or standards.

Some of the advantages of it are:

It detects bugs with the exact location of a bug and fixed, hence the quality of the

product increases.
More technically correct base is available for each new phase of development.
Overall software life cycle cost is lower.
As bugs are found at early stage, less time requires testing product.

➢ Types of static testing:

(1) Software Inspection:

Software inspections were first introduced at IBM by Fagan in the early 1970. It can be used to tackle software quality problem because they allow the detection and removal of defects after each phase of the software development process.

It may be applied to any product or partial product of the software development process, including requirements, design and code, project management plan, SQA plan, software configuration plan, risk management plan, test cases, user manual etc.
This process doesn't require executable code or test cases. It means it doesn't execute the code. So it is machine-independent, requires no target system resources or changes to the program's operational behavior and can be used much before the target hardware is available for dynamic testing purpose. It is a more formal process.

This process is carried out by a group of peers. The group peers first inspect the product at the individual level, then they discuss the potential defects of the product observed in a formal meeting. i.e. Formal peer evaluation of a software element whose objective is to verify that the software element satisfies its specifications and conforms to standards.

For the inspection process minimum of the following team members are required as **Inspection Team.**

- **Author/Owner/Producer:**
  A programmer or designer responsible for producing the program or document. He /She is also responsible for fixing defects discovered during the inspection process.
- **Inspector:**
  A peer member of the team, that is, He/ She is not a manager or supervisor nor directly related to the product under inspection. He /She finds errors, omissions and inconsistencies in program and documents.
- **Moderator:**
  A team member who is a key person or responsible for successful planning, execution and who manages the whole inspection process. Who is also responsible for the following activities:
  Scheduling meeting venue & time, leads & controls the inspection process.

Checking for the product availability for inspection.
Selecting inspection team & assign their role
Distributing inspection material as **to be tested**

- **Recorder:**
  A member who records all results of the meeting.

➢ **Inspection Process stages/steps:**

1) **Planning :** It includes the following activities:
   - The product to be inspected is identified.
   - A moderator is assigned.
   - Objectives are defined like to detect defect or else, what kind of defects to be detected(requirements/ design/code) etc.

   During this phase moderator performs the following activities:
   - Assures that the product is ready for inspection
   - Selects the inspection team and assigns their roles
   - Schedules the meeting venue and time
   - Distributes the inspection material such as the item to be inspected and checklists.

2) **Overview:**
   Define or explain the objectives/purpose of the meeting to team members with the background information. The author presents the rational for the product, its relationship to the rest of the products being developed, its functions and intended use, and the approach used to develop it.

3) **Individual preparation:**
   After the overview, the reviewers individually prepare themselves for the process of inspection by studying the given documents.
   They find defects with the help of checklist & record them in a log form & finally submit that log to moderator.

   Moderator checks whether the reviewers have done work adequately or not. If yes, then moderator checks for defects that may need extra attention during inspection and compiles different logs files & submit to author. If not done then, moderator reschedules the meeting.

4) **Inspection meeting:**
   In this stage author starts meeting by discussing every issue reported in compiled log file. Main goal of it is to uncover any bug & not to fix.

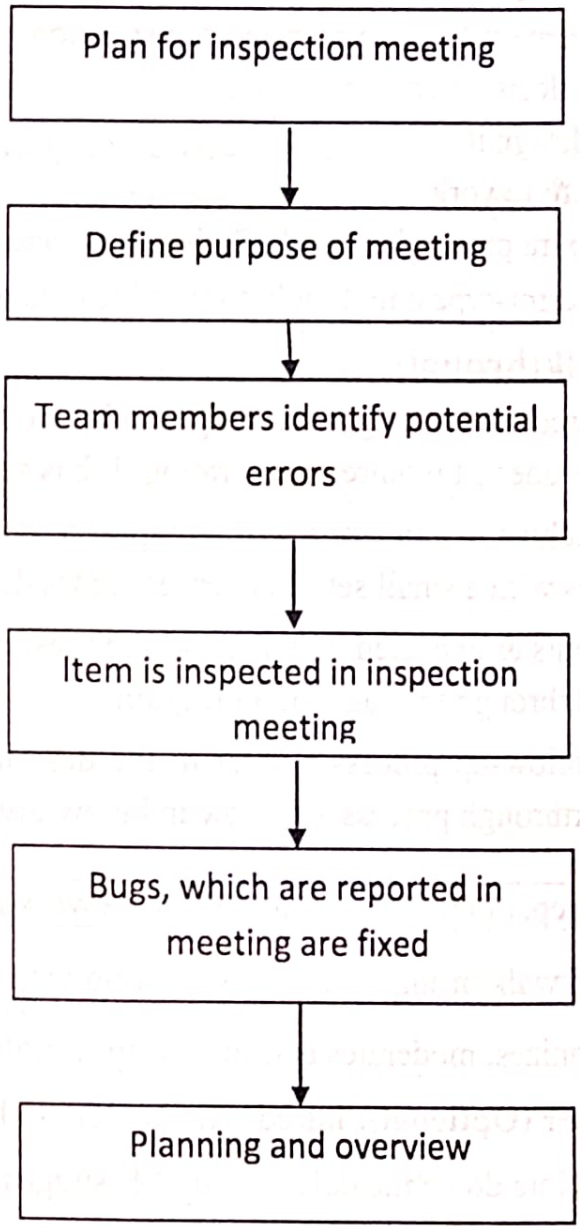
Every activity of the meeting should be constructive so that more & more defects can be discovered. At the end of meeting, moderator prepares a summary report which includes bugs/defect found.

5) Rework:

The summary list of the bugs that arise during the inception meeting needs to be reworked by the author & all bugs should be fixed & reported back to moderator.

6) Follow-up:

It is the responsibility of the moderator to check that the bugs detected in last meeting have been addressed & fixed. Then document is approved for release. If not then another inspection meeting is schedule.



Inspection Process Stages

➢ **Advantages of Inspection Process:**

- Bug reduction and prevention from previous experience of inspection.
- As it is done early, reduced cost of finding bugs & increases productivity.
- It gives real-time feedback to developers.
- Resource requirement is reduced as bugs are detected & fixed at their correct place, hence dynamic testing efforts is reduced.
- It improves quality
- It is useful for monitoring project progress.
- Coupling & Cohesion can be checked easily
- Team members learn some point like various types of defects, why they occur etc through the meeting & discussion.
- Process improvement by information of inspection process. i.e if any module has more defects & decision can be taken as:
  - to redesign it,
  - check & rework
  - take extra precautions and efficient test cases to test that module.
- Distribution of error-types: ie. It help to provide data according to error type.

(2) Structured Walkthrough:

The idea of structured walkthrough was proposed by Yourdon. It is less formal meeting & less rigorous. i.e. doesn't required preparation. It has fewer steps.

It doesn't use a checklist to guide or a written report to document the team's work. In this, tester comes with a small set of paper test cases during the meeting.

Test cases have inputs & expected outputs, and each test case is mentally executed. i.e. test data is walkthrough the logic of a program.

It should have a follow-up process similar to the described in the inspection process. The steps of a walkthrough process are show in below given figure.

Organization → Preparation → walkthrough → Rework and follow-up

A typical structured walkthrough team consists following members with their role:

Coordinator: Organizes, moderates and follow up walkthrough activities

Presenter/Developer (Optional): Introduces the item to be inspected.

Scribe/Recorder: Note down the defects found & suggestions proposed by members

Reviewer/Tester: Finds the defects in the item.

Maintenance Oracle: Focuses on long term implication and future maintenance of the project.

Standard Bearer: Assesses adherence to standards.

User Representative/Accreditation Agent: Reflects the needs and concerns of the user.

(3) Technical Review:

Technical review is intended to evaluate the software in the light of development standards, guidelines and specifications & to show that the development process is being carried out according to the stated objectives.

A review is similar to walkthrough and inspection, except that the review team also includes peers & technical expert with management; hence it is high level technique compare to walkthrough & inspection.

A review team is generally comprised of management-level & project management. The purpose is to evaluate the system relative to specifications and standards and recoding defects and deficiencies.

In this process, a moderator should gather and distribute the documentation to all team members for examination before the review. He/ She should also prepare a set of indicators to measure:

- Appropriateness of the problem definition & requirements.
- Adequacy of all underlying assumption
- Adherence to standards
- Consistency
- Completeness
- Documentation

Moderator may also prepare a checklist to help the team focus only on the key points. At the end a report should be prepared which includes deficiencies identifies & reviewer's recommendations.

Exercises

❖ Answer the following Questions in brief.

1. What is black box testing? List the method offers by BVA to design test cases.
2. Explain the Boundary Value Analysis techniques of testing.
3. Explain Equivalence Class Partitioning testing technique with example.
4. Explain Decision Tables Based testing.
5. What is White Box testing? Why we need it?
6. What is logic coverage criteria technique? List different forms of it and explain all.
7. Explain basis path testing technique in details.
8. What is inspection process? Who can be a member of the inspection team? Explain their role.
9. List and explain different stages of inspection process.
10. Explain structured walkthroughs technique in details.
11. Write a short note on: Technical Review method.

Exercise:

1. A program reads three numbers A,B and C within the range [1,50] and prints the largest number. Design test cases for this program using BVC, robust testing, worst-case testing and robust worst-case testing methods.
2. A program determines the next date in the calendar. Its input is entered in the form of <ddmmyyyy> with the following range:
   1<=m<=12
   1<=dd<=31
   1900<=yyyy<=2025
   Its output will be the next date or it will display "invalid date". Design test cases for this program using BVC, robust testing and worst-case testing methods.
3. For the above program, design test cases using Equivalence Class Partitioning(EVC) method.
4. A program takes an angle as input within the range [0,360] and determines in

which quadrant the angle lies. Design test cases using EVC method.

5. A university is admitting students in a professional course subject to the following conditions:
   a. Marks in Java >=70
   b. Marks in C++ >= 60
   c. Marks in OOAD >= 60
   d. Total in all three subjects >= 220 or Total in Java and C++ >=150

   If the aggregate mark of an eligible candidate is >240, he/she will be eligible for the scholarship course, otherwise he/she will be eligible for a normal course. The program reads the marks in the three subjects and generates the following outputs:
   i. Not eligible
   ii. Eligible for scholarship course
   iii. Eligible for normal course

   Design test cases for this program using decision table testing method.

❖ Multipal choice Question - MCQS:

1. Black-box testing is a _____.
   a. Static testing
   b. Dynamic testing
   c. Logical testing
   d. None of the above

2. It has been observed that test cases, which are designed with boundary input values, have a _____ chance of finding errors.
   a. High
   b. Low
   c. Medium
   d. Zero

3. How many test cases are there in BVC if there are 5 variables in a module?
   a. 23
   b. 13
   c. 10
   d. 21

4. How many test cases are there in robustness testing if there are 5 variables in a module?
   a. 23
   b. 31
   c. 10
   d. 21

5. How many test cases are there in worst-case testing if there are 4 variables in a module?
   a. 623
   b. 513
   c. 625
   d. 521

6. What are the components of a decision table? _____
   a. Condition stub
   b. Condition entry
   c. Action stub
   d. All

7. If there are k rules over n binary conditions, there are at least _____ test cases and at the most _____ test cases.
   a. k+2, 2n+2
   b. k+3, 2n+3
   c. k, 2n
   d. None of the above

8. Boundary value analysis and equivalence class partitioning methods do not consider _____.
   a. Combinations of input conditions
   b. Inputs
   c. Outputs
   d. None

9. White-box testing is _____ to black-box testing.
   a. mutually exclusive
   b. complementary
   c. not related
   d. related

10. The effectiveness of path testing gets_____with the increase in size of software under test.

a. reduced
b. increased
c. does not change
d. none of the above

11. A node with more than one arrow leaving it is called a _____.
a. decision node
b. junction node
c. region
d. all of the above

12. A node with more than one arrow entering it is called a _____.
a. decision node
b. junction node
c. region
d. all of the above

13. Areas bounded by edges and nodes are called _____.
a. decision node
b. junction node
c. region
d. all of the above

14. The length of a path is measured by the number of _____.
a. Instructions
b. junction nodes
c. decision nodes
d. links

15. 7. An independent path is any path through the graph that introduces at least _____ new set of processing statements or new conditions.
a. 4
b. 3
c. 1
d. 2

16. The number of independent paths is given by _____.
a. $V(G) = e - n + 1$
b. $V(G) = 2e - n + 1$
c. $V(G) = e - n + 2$

d. none of the above

17. According to Mill's Theorem, _____ .

a. V(G) = d + 2P
b. V(G) = d + P
c. V(G) = 2d + P
d. None of the above

18. In static testing, a bug is found at its _____ location.

a. Exact
b. Nearby
c. None of the above

19. Static testing can be applied for most of the _____ .

a. Validation activities
b. Verification activities
c. SDLC activities
d. None of the above

20. Formal peer evaluation of a software element whose objective is to verify that the software element satisfies its specifications and conforms to standards, is called _____ .

a. Walkthrough
b. Inspections
c. Reviews
d. None of the above

21. The programmer or designer responsible for producing the program or document is known as _____.

a. Author
b. Owner
c. Producer
d. All

22. The person who finds errors, omissions, and inconsistencies in programs and documents during an inspection is known as _____ .

a. Inspector
b. Moderator
c. Author
d. Producer

23. The key person with the responsibility of planning and successful execution of inspection is

known as _____.

a. Inspector
b. Moderator
c. Author
d. Producer

24. The inspection team points out any potential errors or problems found and records them in _____.

a. SDD
b. SRS
c. STD
d. Log Form

25. 'How much evaluation of an item has been done by the team' is called _____.

a. Rate of errors
b. Rate of inspection
c. Rate of failures
d. None of the above

26. _____ is a more formal process.

a. Walkthroughs
b. Inspection
c. Reviews
d. None of the above

27. A review is similar to an inspection or walkthrough, except that the review team also includes _____.

a. Customer
b. Developer
c. Tester
d. Management

Answer:

1. b. Dynamic Testing 2. a. High 3. d. 21 4. b. 3 5. c. 625
6. d. all 7. c. k,2n 8. a. combinations of input conditions
9. b. complementary 10. b. reduced 11. a. decision node 12. b. junction node
13. c. region 14. d. links 15. c.1 16. c. V(G)=e-n+2 17. b. V(G)=d+p
18. a. Exact 19. b. verification activities 20. b. inspections 21. d.all
22. a. Inspector 23. b. moderator 24. d. Log form
25. d. none of the above 26. b. Inspection 27. d. management

❖ Practical Exercise :

Practical 1 :

a. Test case using BVC:

Since there are three variables A,B and C. (n=3), the total number of test cases will be 4n+1=4*3+1=13

The set of boundary values is shown:

Min value	1
Min+ value	2
Max value	50
Max- value	49
Nominal value	25-30

Using these values, test cases can be designed as shown below:

Test Case ID	A	B	C	Expected Output
1.	1	25	27	C is largest
2.	2	25	28	C is largest
3.	49	25	25	A is largest
4.	50	25	29	A is largest
5.	25	1	30	C is largest
6.	25	2	26	C is largest
7.	25	49	27	B is largest
8.	25	50	28	B is largest
9.	25	28	1	B is largest
10.	25	27	2	B is largest
11.	25	26	49	C is largest
12.	25	26	50	C is largest
13.	25	25	25	All three are same

b. Test case using robust testing:

Here total number of test cases will be 6n+1=6*3+1=19

The set of boundary values is shown below:

Min value	1
Min- value	0
Min+ value	2
Max value	50

Max+ value	51
Max- value	49
Nominal value	25-30

Using these values, test cases will be designed as shown below:

Test Case ID	A	B	C	Expected Output
1.	0	25	27	Input A is invalid
2.	1	25	27	C is largest
3.	2	25	28	C is largest
4.	49	25	25	A is largest
5.	50	25	29	A is largest
6.	51	25	25	Input A is invalid
7.	25	0	30	Input B is invalid
8.	25	1	30	C is largest
9.	25	2	26	C is largest
10.	25	49	27	B is largest
11.	25	50	28	B is largest
12.	25	51	26	Input B is invalid
13.	25	25	0	Input C is invalid
14.	25	28	1	B is largest
15.	25	27	2	B is largest
16.	25	26	49	C is largest
17.	25	26	50	C is largest
18.	25	25	25	All three are same
19.	25	29	51	Input C is invalid

c. Test case using worst-case testing:

Total number of test cases will be $5^n=5^3=125$

The set of boundary values is shown below:

Min value	1
Min+ value	2
Max value	50
Max- value	49
Nominal value	25-30

Using these values, test cases will be designed as shown below:

Test Case ID	A	B	C	Expected Output
1.	1	1	1	All three are same
2.	1	1	2	C is largest
3.	1	1	25	C is largest
4.	1	1	49	C is largest

5.	1	1	50	C is largest
6.	1	2	1	B is largest
7.	1	2	2	B and C are largest
8.	1	2	35	C is largest
9.	1	2	49	C is largest
10.	1	2	50	C is largest
11.	1	25	1	B is largest
12.	1	25	2	B is largest
13.	1	26	25	B is largest
14.	1	25	49	C is largest
15.	1	25	50	C is largest
16.	1	49	1	B is largest
17.	1	49	2	B is largest
18.	1	49	25	B is largest
19.	1	49	49	B and C are largest
20.	1	49	50	C is largest
21.	1	50	1	B is largest
22.	1	50	2	B is largest
23.	1	50	25	B is largest
24.	1	50	49	B is largest
25.	1	50	50	B is largest
26.	2	1	1	A is largest
27.	2	1	2	A and C are largest
28.	2	1	25	C is largest
29.	2	1	49	C is largest
30.	2	1	50	C is largest
31.	2	2	1	A and B are largest
32.	2	2	2	All are same
33.	2	2	25	C is largest
34.	2	2	49	C is largest
35.	2	2	50	C is largest
36.	2	25	1	B is largest
37.	2	27	2	B is largest
38.	2	28	25	B is largest
39.	2	26	49	C is largest
40.	2	28	50	C is largest
41.	2	49	1	B is largest
42.	2	49	2	B is largest
43.	2	49	25	B is largest
44.	2	49	49	B and C are largest
45.	2	49	50	C is largest

46.	2	50	1	B is largest
47.	2	50	2	B is largest
48.	2	50	25	B is largest
49.	2	50	49	B is largest
50.	2	50	50	B and C are largest
51.	25	1	1	A is largest
52.	25	1	2	A is largest
53.	25	1	25	A and C are largest
54.	25	1	49	C is largest
55.	25	1	50	C is largest
56.	25	2	1	A is largest
57.	25	2	2	A is largest
58.	25	2	25	A and C are largest
59.	25	2	49	C is largest
60.	25	2	50	C is largest
61.	25	27	1	B is largest
62.	25	26	2	B is largest
63.	25	25	25	All are same
64.	25	28	49	C is largest
65.	25	29	50	C is largest
66.	25	49	1	B is largest
67.	25	49	2	B is largest
68.	25	49	25	B is largest
69.	25	49	49	B and C are largest
70.	25	49	50	C is largest
71.	25	50	1	B is largest
72.	25	50	2	B is largest
73.	25	50	25	B is largest
74.	25	50	49	B is largest
75.	25	50	50	B and C are largest
76.	49	1	1	A is largest
77.	49	1	2	A is largest
78.	49	1	25	A is largest
79.	49	1	49	A and C are largest
80.	49	1	50	C is largest
81.	49	2	1	A is largest
82.	49	2	2	A is largest
83.	49	2	25	A is largest
84.	49	2	49	A and C are largest
85.	49	2	50	C is largest
86.	49	25	1	A is largest

87.	49	29	2	A is largest
88.	49	25	25	A is largest
89.	49	27	49	A and C are largest
90.	49	28	50	C is largest
91.	49	49	1	A and B are largest
92.	49	49	2	A and B are largest
93.	49	49	25	A and B are largest
94.	49	49	49	All are same
95.	49	49	50	C is largest
96.	49	50	1	B is largest
97.	49	50	2	B is largest
98.	49	50	25	B is largest
99.	49	50	49	B is largest
100.	49	50	50	B and C are largest
101.	50	1	1	A is largest
102.	50	1	2	A is largest
103.	50	1	25	A is largest
104.	50	1	49	A is largest
105.	50	1	50	A and C are largest
106.	50	2	1	A is largest
107.	50	2	2	A is largest
108.	50	2	25	A is largest
109.	50	2	49	A is largest
110.	50	2	50	A and C are largest
111.	50	26	1	A is largest
112.	50	25	2	A is largest
113.	50	27	25	A is largest
114.	50	29	49	A is largest
115.	50	30	50	A and C are largest
116.	50	49	1	A is largest
117.	50	49	2	A is largest
118.	50	49	25	A is largest
119.	50	49	49	A is largest
120.	50	49	50	A and C are largest
121.	50	50	1	A and B are largest
122.	50	50	2	A and B are largest
123.	50	50	25	A and B are largest
124.	50	50	49	A and B are largest
125.	50	50	50	All are same

d. Test case using robust worst-case testing:

Total number of test cases will be $7^n=7^3=343$

The set of boundary values is shown below:

Min- value	0
Min value	1
Min+ value	2
Nominal value	25-30
Max- value	49
Max value	50
Max+ value	51

Practical 2 :

a. **Test cases using BVC:**

Since there are three variables month, day and year, the total number of test cases will be 4n+1=4*3+1=13

The set of boundary values is shown:

	Month	Day	Year
Min value	1	1	1900
Min+ value	2	2	1901
Max value	12	31	2025
Max- value	11	30	2024
Nominal value	6	15	1962

Using these values, test cases can be designed as shown below:

Test Case ID	Month	Day	Year	Expected Output
1.	1	15	1962	16-1-1962
2.	2	15	1962	16-2-1962
3.	11	15	1962	16-11-1962
4.	12	15	1962	16-12-1962
5.	6	1	1962	2-6-1962
6.	6	2	1962	3-6-1962
7.	6	30	1962	1-7-1962
8.	6	31	1962	Invalid input(Day)
9.	6	15	1900	16-6-1900
10.	6	15	1901	16-6-1901
11.	6	15	2024	16-6-2024
12.	6	15	2025	16-6-2025
13.	6	15	1962	16-6-1962

b. Test cases using robust testing:

Here total number of test cases will be 6n+1=6*3+1=19

The set of boundary values is shown below:

	Month	Day	Year
Min value	1	1	1900
Min- value	0	0	1899
Min+ value	2	2	1901
Max value	12	31	2025
Max+ value	13	32	2026
Max- value	11	30	2024
Nominal value	6	15	1962

Using these values, test cases will be designed as shown below:

Test Case ID	Month	Day	Year	Expected Output
1.	0	15	1962	Invalid date
2.	1	15	1962	16-1-1962
3.	2	15	1962	16-2-1962
4.	11	15	1962	16-11-1962
5.	12	15	1962	16-12-1962
6.	13	15	1962	Invalid date
7.	6	0	1962	Invalid date
8.	6	1	1962	2-6-1962
9.	6	2	1962	3-6-1962
10.	6	30	1962	1-7-1962
11.	6	31	1962	Invalid date
12.	6	32	1962	Invalid date
13.	6	15	1899	Invalid date
14.	6	15	1900	16-6-1900
15.	6	15	1901	16-6-1901
16.	6	15	2024	16-6-2024
17.	6	15	2025	16-6-2025
18.	6	15	2026	Invalid date
19.	6	15	1962	16-6-1962

c. Test cases using worst-case testing:

Total number of test cases will be $5^n=5^3=125$

The set of boundary values is shown below:

	Month	Day	Year
Min value	1	1	1900

Min+ value	2	2	1901
Max value	12	31	2025
Max- value	11	30	2024
Nominal value	6	15	1962

Using these values, test cases will be designed as shown below:

Test Case ID	Month	Day	Year	Expected Output
1.	1	1	1900	2-1-1900
2.	1	1	1901	2-1-1901
3.	1	1	1962	2-1-1962
4.	1	1	2024	2-1-2024
5.	1	1	2025	2-1-2025
6.	1	2	1900	3-1-1900
7.	1	2	1901	3-1-1901
8.	1	2	1962	3-1-1962
9.	1	2	2024	3-1-2024
10.	1	2	2025	3-1-2025
11.	1	15	1900	16-1-1900
12.	1	15	1901	16-1-1901
13.	1	15	1962	16-1-1962
14.	1	15	2024	16-1-2024
15.	1	15	2025	16-1-2025
16.	1	30	1900	30-1-1900
17.	1	30	1901	30-1-1901
18.	1	30	1962	30-1-1962
19.	1	30	2024	30-1-2024
20.	1	30	2025	30-1-2025
21.	1	31	1900	1-2-1900
22.	1	31	1901	1-2-1901
23.	1	31	1962	1-2-1962
24.	1	31	2024	1-2-2024
25.	1	31	2025	1-2-2025
26.	2	1	1900	2-2-1900
27.	2	1	1901	2-2-1901
28.	2	1	1962	2-2-1962
29.	2	1	2024	2-2-2024
30.	2	1	2025	2-2-2025
31.	2	2	1900	3-2-1900
32.	2	2	1901	3-2-1901
33.	2	2	1962	3-2-1962

34.	2	2	2024	3-2-2024
35.	2	2	2025	3-2-2025
36.	2	15	1900	16-2-1900
37.	2	15	1901	16-2-1901
38.	2	15	1962	16-2-1962
39.	2	15	2024	16-2-2024
40.	2	15	2025	16-2-2025
41.	2	30	1900	Invalid date
42.	2	30	1901	Invalid date
43.	2	30	1962	Invalid date
44.	2	30	2024	Invalid date
45.	2	30	2025	Invalid date
46.	2	31	1900	Invalid date
47.	2	31	1901	Invalid date
48.	2	31	1962	Invalid date
49.	2	31	2024	Invalid date
50.	2	31	2025	Invalid date
51.	6	1	1900	2-6-1900
52.	6	1	1901	2-6-1901
53.	6	1	1962	2-6-1962
54.	6	1	2024	2-6-2024
55.	6	1	2025	2-6-2025
56.	6	2	1900	3-6-1900
57.	6	2	1901	3-6-1901
58.	6	2	1962	3-6-1962
59.	6	2	2024	3-6-2024
60.	6	2	2025	3-6-2025
61.	6	15	1900	16-6-1900
62.	6	15	1901	16-6-1901
63.	6	15	1962	16-6-1962
64.	6	15	2024	16-6-2024
65.	6	15	2025	16-6-2025
66.	6	30	1900	1-7-1900
67.	6	30	1901	1-7-1901
68.	6	30	1962	1-7-1962
69.	6	30	2024	1-7-2024
70.	6	30	2025	1-7-2025
71.	6	31	1900	Invalid date
72.	6	31	1901	Invalid date
73.	6	31	1962	Invalid date
74.	6	31	2024	Invalid date

75.	6	31	2025	Invalid date
76.	11	1	1900	2-11-1900
77.	11	1	1901	2-11-1901
78.	11	1	1962	2-11-1962
79.	11	1	2024	2-11-2024
80.	11	1	2025	2-11-2025
81.	11	2	1900	3-11-1900
82.	11	2	1901	3-11-1901
83.	11	2	1962	3-11-1962
84.	11	2	2024	3-11-2024
85.	11	2	2025	3-11-2025
86.	11	15	1900	16-11-1900
87.	11	15	1901	16-11-1901
88.	11	15	1962	16-11-1962
89.	11	15	2024	16-11-2024
90.	11	15	2025	16-11-2025
91.	11	30	1900	1-12-1900
92.	11	30	1901	1-12-1901
93.	11	30	1962	1-12-1962
94.	11	30	2024	1-12-2024
95.	11	30	2025	1-12-2025
96.	11	31	1900	Invalid date
97.	11	31	1901	Invalid date
98.	11	31	1962	Invalid date
99.	11	31	2024	Invalid date
100.	11	31	2025	Invalid date
101.	12	1	1900	2-12-1900
102.	12	1	1901	2-12-1901
103.	12	1	1962	2-12-1962
104.	12	1	2024	2-12-2024
105.	12	1	2025	2-12-2025
106.	12	2	1900	3-12-1900
107.	12	2	1901	3-12-1901
108.	12	2	1962	3-12-1962
109.	12	2	2024	3-12-2024
110.	12	2	2025	3-12-2025
111.	12	15	1900	16-12-1900
112.	12	15	1901	16-12-1901
113.	12	15	1962	16-12-1962
114.	12	15	2024	16-12-2024
115.	12	15	2025	16-12-2025

116.	12	30	1900	31-12-1900
117.	12	30	1901	31-12-1901
118.	12	30	1962	31-12-1962
119.	12	30	2024	31-12-2024
120.	12	30	2025	31-12-2025
121.	12	31	1900	1-1-1900
122.	12	31	1901	1-1-1901
123.	12	31	1962	1-1-1962
124.	12	31	2024	1-1-2024
125.	12	31	2025	1-1-2025

UNIT-3

Levels of Testing

- Unit Testing : Overview
- Integration Testing : Overview
- Techniques: Graph based & Path based
- Functional Testing
- System Testing :Overview
- Categories: Reliability Security Performance Recovery
- Acceptance Testing : Overview, Types of Acceptance Testing

UNIT-3 Levels of Testing

❖ LEVELS OF TESTING:

- The system development phases involve many activities where chances for occurrence of human errors are huge.
- Logical error, carelessness, improper communication, the need to hurry through the whole process of software development due to time constraint, cost constraint etc.
- The system must be tested thoroughly so that such errors are detected and corrected as soon as possible.
- A successful test is one that uncovers every possible error.
- The systems are not designed as entire system and they are not tested as a single system.
- The analyst must perform both unit and system testing.
- Tests are frequently grouped by where they are added in the system development process, or by the level of specificity of the test.
- The levels of testing are as follows:

3.1 Unit Testing: Overview:

- In unit testing, the analyst tests the programs, which makes up a system.
- It is also known as program testing or module testing.
- Unit testing focuses on the modules to find the errors.
- This enables the tester to detect errors in coding and logic that is contained within that module alone.
- **For Example**, A hotel Information System have modules such as to handle reservation, guest check-in and check-out, restaurant, room service, and account billing and so on.
- Unit testing can be performed bottom up, that is starting with the smallest and lowest level modules and proceeding one at a time.

- **Advantages of Unit Testing:**
  - ✓ More reusable code and easier debugging.
  - ✓ Reduced cost for testing and bug-fixing.
  - ✓ Increased efficiency of code improvement and maintenance.
  - ✓ Simplified interaction due to separate and complex modules testing.
  - ✓ Errors can be detected at the early stages of the SDLC.

- **Disadvantages of Unit Testing:**

☒ It's difficult to write quality unit tests and the whole process can be time-consuming.
☒ Human factor. A developer can make a mistake that will impact the whole system.
☒ Not all errors can be detected, since every module it tested separately and later different integration bugs may appear.

3.2 Integration Testing: Overview:

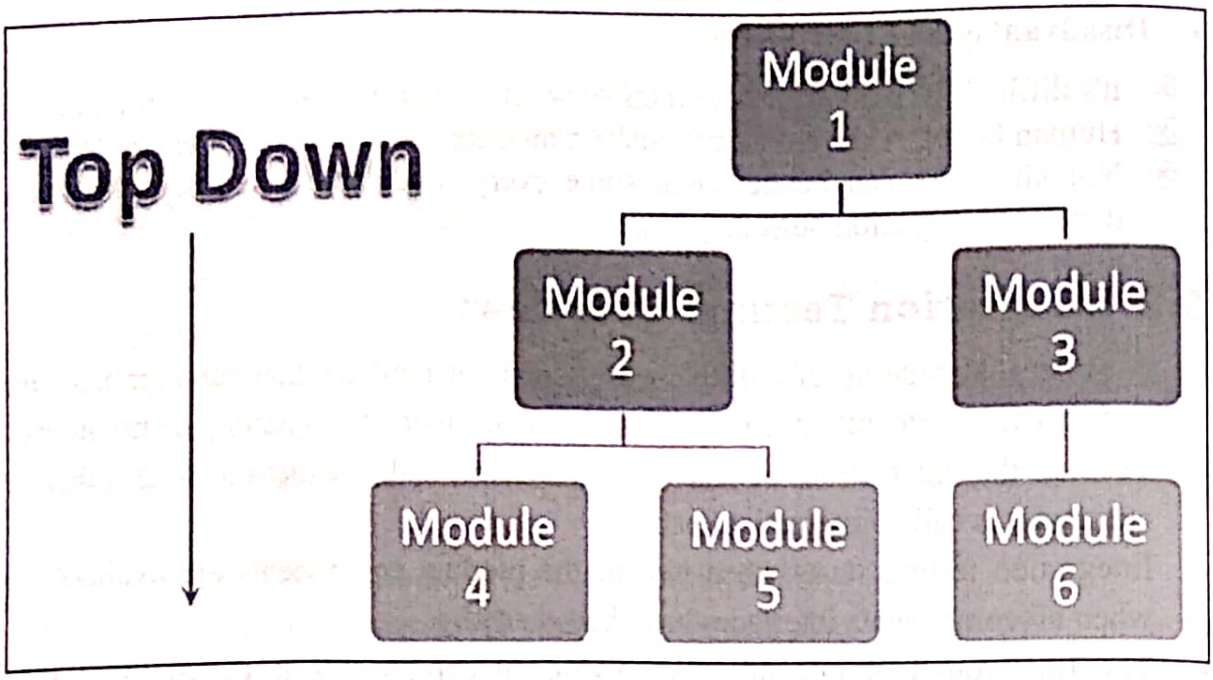
- A system is made up of multiple components or modules that can comprise hardware and software. Integration is defined as the set of interactions among components.
- Testing the interaction between the modules and interaction with other systems externally is called integration testing.
- Integration testing starts when two of the product components are available and end when all components interfaces have been tested.
- The final round of integration involving all components is called Final Integration Testing (FIT), or system integration.

- **Integration Testing as A Type of Testing:**

- Integration testing means testing of interfaces.
- When we talk about interfaces, there are two types of interfaces that have to be kept in mind for proper integration testing. They are *internal interfaces* and *exported* or *external interfaces*.
- Internal interfaces are those that provide communication across two modules within a project or product, internal to the product, and not exposed to the customer or external developers.
- Exported interfaces are those that are visible outside the product to third party developers and solution providers.
- There are several method of integration testing which are as follows:

1. **Top-Down Integration:**

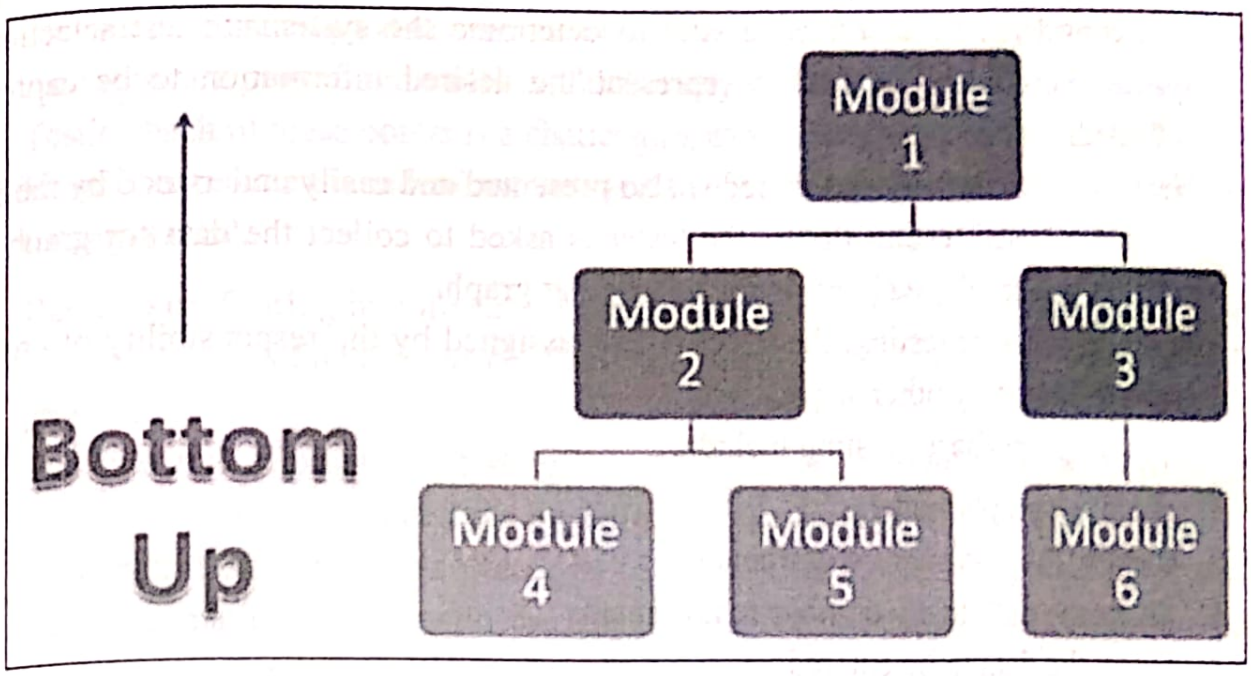
- Integration testing involves testing the topmost component interface with other components in same order as you navigate from top to bottom, till you cover all the components.
- In Top to down approach, testing takes place from top to down following the control flow of the software system.



- **Advantages of Top-Down Integration Testing:**
  - ✓ Easily faults or errors can be detected in this approach.
  - ✓ Crucial modules are tested thoroughly and prior to other modules.
  - ✓ Software system integration testing can be done in less time as when compared with other approaches.
- **Disadvantages of Top-Down Integration Testing:**
  - ✓ Bottom level modules may not be tested to the expected level or may not be tested to the requirements.
  - ✓ Stubs are needed and are required for the testing process to progress further.

2. Bottom-Up Integration:

- Bottom-up integration is just opposite of top-down integration, where the components for a new product development become available in reverse order, starting from the bottom.
- In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing.



- **Advantages of Bottom-Up Integration Testing:**
  - ✓ Development of individual modules can be done while the integration testing bottom-up approach is being used, as the coupling and integration testing is done after the bottom level modules are tested first.
  - ✓ If some error exists/arises, it can be fixed at the same time and at the same level. The error identification and correction are much easier than other approaches.
  - ✓ Time required for error identification and error correction is much less as compared with other approaches.
  - ✓ Errors can be solved at the same instance bottom level or at the top level.

- **Disadvantages of Bottom-Up Integration Testing:**
  - ✓ Time taken for the whole process is more, the process of testing does not finish till all the modules of both, the top and the bottom level are included and tested.
  - ✓ Drivers are necessary to call the high-level modules
  - ✓ If the software system contains, more and more small, but complex modules, it may take more time for completion of the software testing process.

3.3 Techniques: Graph based & Path based:

- **Graph Based:**

- Graph is one of the most widely used structures. It is a well-defined structure which makes the testing or case studies easier.

- It is considered as an effective way to determine the systematic test selection of a system. This model is used to represent the desired information to be captured or collected.
- Next, the information to be shared is also presented and easily understood by the testers.
- For graph-based testing firstly, the tester is asked to collect the data for graph model and then cover all the elements for a particular graph.
- In this process of testing, the tester is first assigned by the responsibility of creating a graph followed by other steps.
- Steps for graph-based testing include:

1. Build the graph model
2. Identify the test/major requirements
3. Select the path to cover those requirements
4. Select the data to be entered

- This graph is helpful in certain circumstances:
  - ⇨ To determine the present problem so that decision can be taken fast.
  - ⇨ To correlate the factors affecting the system.
  - ⇨ To recognize the main cause of a problem.

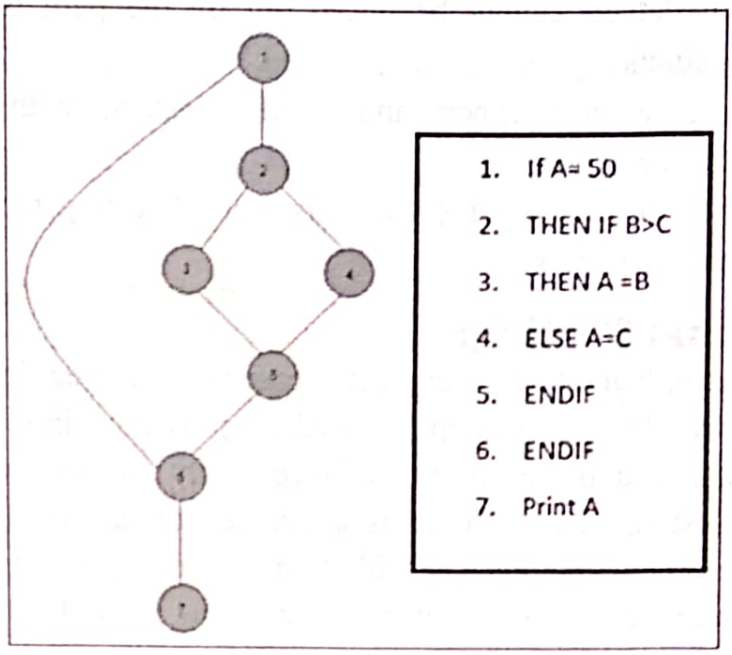
- **Advantages of using cause effect graph are as follows:**
  - ✓ It highlights the area from which the data is taken and can be taken for additional studies.
  - ✓ It helps in motivating the team.
  - ✓ The data is arranged in such manner that even a non-technical person can also read it.
  - ✓ It helps in detecting the reason of differences occurred in a process.
  - ✓ It helps the team to decide the root reason of a problem.
  - ✓ It is very appropriate for large systems.

- **Disadvantages of using cause effect graph are as follows:**
  - ☒ Difficult to design.
  - ☒ It is difficult to choose the important input in limited time.
  - ☒ There can be chances where path of drawing the graphs is not clear.
  - ☒ There are chances of repetition of data already entered in the graph.

- **Path Based:**
  - Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path.
  - It helps to determine all faults lying within a piece of code.

- This method is designed to execute all or selected path through a computer program.
- Any software program includes, multiple entry and exit points.
- Testing each of these points is a challenging as well as time-consuming.
- In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

- **Basis Path Testing** in software engineering is a White Box Testing method in which test cases are defined based on flows or logical paths that can be taken through the program.
- The objective of basis path testing is to define the number of independent paths, so the number of test cases needed can be defined explicitly to maximize test coverage.
- Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid method of branch testing and path testing methods.
- Here we will take a simple example, to get a better idea what is basis path testing include.



- In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,
  - ⇨ **Path 1**: 1,2,3,5,6, 7
  - ⇨ **Path 2**: 1,2,4,5,6, 7
  - ⇨ **Path 3**: 1, 6, 7

5	The comparison of actual and expected results based on the executed test cases.

- An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

- **Advantages of Functional Testing:**
  - ✓ Functional testing focuses on the requirements of the guidelines for end users to test the various models of the applications.
  - ✓ It accepts realistic facts and figures to make testing reports.
  - ✓ This type of testing is conducted when the project is very close to customers like operating system browsers database Excel.
  - ✓ It is a testing technique which focuses on the business and also helps to reduce the gap between the business organization and their uses.

- **Disadvantages of Functional Testing:**
  - ☒ Functional testing is a process in which various logical mistakes in the software are not detected in the testing process.
  - ☒ where is also a possibility of redundant testing which me in hands the testing cost and efforts of the.
  - ☒ It doesn't care how the developer implements the actual source code because it only focuses on the results of the source code.

3.5 System Testing: Overview:

- System testing is defined as a testing phase conducted on the complete integrated system, to evaluate the system compliance with its specified requirements.
- It is done after unit, component, and integration testing phases.
- A system is complete set of integrated components that together deliver product functionality and features.
- A system can also be defined as a set of hardware, software and other parts that together provide product features and solutions.
- System testing is the only phase of testing which tests the both *functional* and *non-functional* aspects of the product.
- On the functional side, system testing focus on real-life customer usage pf the product and solutions.
- On the non-functional side, system brings in different testing types also called quality factors, some of which are as follows:

5	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

- **Advantages of Functional Testing:**
  - ✓ Functional testing focuses on the requirements of the guidelines for end users to test the various models of the applications.
  - ✓ It accepts realistic facts and figures to make testing reports.
  - ✓ This type of testing is conducted when the project is very close to customers like operating system browsers database Excel.
  - ✓ It is a testing technique which focuses on the business and also helps to reduce the gap between the business organization and their uses.

- **Disadvantages of Functional Testing:**
  - ☒ Functional testing is a process in which various logical mistakes in the software are not detected in the testing process.
  - ☒ where is also a possibility of redundant testing which me in hands the testing cost and efforts of the.
  - ☒ It doesn't care how the developer implements the actual source code because it only focuses on the results of the source code.

3.5 System Testing: Overview:

- System testing is defined as a testing phase conducted on the complete integrated system, to evaluate the system compliance with its specified requirements.
- It is done after unit, component, and integration testing phases.
- A system is complete set of integrated components that together deliver product functionality and features.
- A system can also be defined as a set of hardware, software and other parts that together provide product features and solutions.
- System testing is the only phase of testing which tests the both *functional* and *non-functional* aspects of the product.
- On the functional side, system testing focus on real-life customer usage pf the product and solutions.
- On the non-functional side, system brings in different testing types also called quality factors, some of which are as follows:

- **Steps for Basis Path testing:**

- The basic steps involved in basis path testing include-
  - ⇨ Draw a control graph (to determine different program paths)
  - ⇨ Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
  - ⇨ Find a basis set of paths
  - ⇨ Generate test cases to exercise each path

- **Advantages of Basic Path Testing:**
  - ✓ It helps to reduce the redundant tests
  - ✓ It focuses attention on program logic
  - ✓ It helps facilitates analytical versus arbitrary case design
  - ✓ Test cases which exercise basis set will execute every statement in a program at least once

- **Disadvantages of Basic Path Testing:**
  - ☒ Presence of defects cannot be traced out by the path testing due to an error in the specifications.
  - ☒ Path testing requires expert and skillful testers, with in-depth knowledge of programming and code.
  - ☒ It is difficult to test all paths with this type of testing technique when the product becomes more complex.

3.4 Functional Testing:

- This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for.
- Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
- There are five steps that are involved while testing an application for functionality.

Steps	Description
1	The determination of the functionality that the intended application is meant to perform.
2	The creation of test data based on the specifications of the application.
3	The output based on the test data and the specifications of the application.
4	The writing of test scenarios and the execution of test cases.

5	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

- **Advantages of Functional Testing:**
  - ✓ Functional testing focuses on the requirements of the guidelines for end users to test the various models of the applications.
  - ✓ It accepts realistic facts and figures to make testing reports.
  - ✓ This type of testing is conducted when the project is very close to customers like operating system browsers database Excel.
  - ✓ It is a testing technique which focuses on the business and also helps to reduce the gap between the business organization and their uses.

- **Disadvantages of Functional Testing:**
  - ☒ Functional testing is a process in which various logical mistakes in the software are not detected in the testing process.
  - ☒ where is also a possibility of redundant testing which me in hands the testing cost and efforts of the.
  - ☒ It doesn't care how the developer implements the actual source code because it only focuses on the results of the source code.

3.5 System Testing: Overview:

- System testing is defined as a testing phase conducted on the complete integrated system, to evaluate the system compliance with its specified requirements.
- It is done after unit, component, and integration testing phases.
- A system is complete set of integrated components that together deliver product functionality and features.
- A system can also be defined as a set of hardware, software and other parts that together provide product features and solutions.
- System testing is the only phase of testing which tests the both *functional* and *non-functional* aspects of the product.
- On the functional side, system testing focus on real-life customer usage pf the product and solutions.
- On the non-functional side, system brings in different testing types also called quality factors, some of which are as follows:

1. **Performance/Load testing**

- To evaluate the time taken or response time of the system to perform its required functions in comparison with different versions of same product or a different competitive product is called performance testing.

2. **Scalability testing**

- A testing that requires huge amount of resource to find out the maximum capability of the system parameters is called scalability testing.

3. **Reliability testing**

- To evaluate the ability of the system to perform its function repeatedly for a specified period of time is called reliability testing.

4. **Stress testing**

- Evaluating a system beyond the limits of the specified requirement to ensure the system does not break down unexpectedly is called stress testing.

5. **Interoperability testing**

- This testing is done to ensure that two or more products can exchange information, use the information, and work closely.

6. **Localization testing**

- Testing conducted to verify that the localized product works in different languages is called localization testing.

- **Why is System Testing Done?**

- An independent test team normally does system testing.
- The behaviour of the complete product is verified during system testing.
- Tests that refer to multiple modules, programs, and functionality are included in system testing.
- System testing helps in identifying as many defects as possible before the customer finds them in deployment.
- System testing is conducted with an objective to find product level defects and in building confidence before the product is released to the customer.
- To summarize, system testing is done for the following reasons.
  - ⇨ Provide independent viewpoint in testing.
  - ⇨ Bring in customer perspective in testing.
  - ⇨ Provide a "fresh pair of eyes" to discover defects not found earlier by testing.
  - ⇨ Test product behaviour in a complete and realistic environment.
  - ⇨ Test both functional and non-functional aspects of the product.
  - ⇨ Build confidence in the product.

⇨ Analyse and reduce the risk of releasing the product.

⇨ Ensure all requirements are met and ready the product for acceptance testing.

- **Advantages of System Testing**

✓ It covers a complete end to end software testing.

✓ The business requirements and system software architecture are both tested in system testing.

✓ Appropriate system testing help in relieving after production goes live issues and bugs.

✓ System testing is run in a situation like a production condition or some of the time it is finished with production parallel test condition where the same data input is feed to the exiting framework and new framework to look at the differences in functionalities removed and added. This causes the client to understand the new framework better and feel great with new functionalities included or existing functionalities revised or removed.

- **Disadvantages of System Testing**

☒ As it needs to test the entire framework it requires a lot of time

☒ Relying upon the business necessities and application architecture cost may also increase.

3.6 Categories: Reliability Security Performance Recovery:

- **Reliability:**

- Reliability Testing is a software testing process that checks whether the software can perform a failure-free operation for a specified time period in a particular environment.
- The purpose of Reliability testing is to assure that the software product is bug free and reliable enough for its expected purpose.
- Reliability means "yielding the same," in other terms, the word "reliable" means something is dependable and that it will give the same outcome every time. The same is true for Reliability testing.

- **Factors Influencing Software Reliability:**

1. The number of faults presents in the software
2. The way users operate the system

- Reliability Testing is one of the keys to better software quality. This testing helps discover many problems in the software design and functionality.
- The main purpose of reliability testing is to check whether the software meets the requirement of customer's reliability.
- Reliability testing will be performed at several levels. Complex systems will be tested at unit, assembly, subsystem and system levels.

- **Why to do Reliability Testing:**

- Reliability testing is done to test the software performance under the given conditions.
- The objective behind performing reliability testing is,
  1. To find the structure of repeating failures.
  2. To find the number of failures occurring is the specified amount of time.
  3. To discover the main cause of failure
  4. To conduct Performance Testing of various modules of software application after fixing defect.
- After the release of the product too, we can minimize the possibility of occurrence of defects and thereby improve the software reliability.
- Some of the tools useful for this are - Trend Analysis, Orthogonal Defect Classification and formal methods, etc.

- **Security:**

- Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure –
  - ⇨ Confidentiality
  - ⇨ Integrity
  - ⇨ Authentication
  - ⇨ Availability
  - ⇨ Authorization
  - ⇨ Non-repudiation
  - ⇨ Software is secure against known and unknown vulnerabilities
  - ⇨ Software data is secure
  - ⇨ Software is according to all security regulations
  - ⇨ Input checking and validation
  - ⇨ SQL insertion attacks
  - ⇨ Injection flaws
  - ⇨ Session management issues
  - ⇨ Cross-site scripting attacks
  - ⇨ Buffer overflows vulnerabilities
  - ⇨ Directory traversal attacks

- **Performance:** It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in software. There are different causes that contribute in lowering the performance of a software –

- ⇨ Network delay
- ⇨ Client-side processing
- ⇨ Database transaction processing
- ⇨ Load balancing between servers
- ⇨ Data rendering

- Performance testing is considered as one of the important and mandatory testing types in terms of the following aspects –
  - ⇨ Speed (i.e., Response Time, data rendering and accessing)
  - ⇨ Capacity
  - ⇨ Stability
  - ⇨ Scalability
- Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

- **Load Testing:**
- It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.
- Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.
- Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

- **Stress Testing:**
- Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.
- The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as –
  - ⇨ Shutdown or restart of network ports randomly
  - ⇨ Turning the database on or off
  - ⇨ Running different processes that consume resources such as CPU, memory, server, etc.

- The strategy can be unique to each organization based on the criticality of the systems they are handling.
- The possible strategy for critical systems can be visualized as follows:
  - ⇨ To have a single backup or more than one
  - ⇨ To have multiple back-ups at one place or different places
  - ⇨ To have an online backup or offline backup
  - ⇨ Can the backup be done automatically based on a policy or to have it manually?
  - ⇨ To have an independent restoration team or development team itself can be utilized for the work

- **Advantages of Recovery Testing:**
  - ✓ **Improves the quality of the system** by eliminating the potential flaws in the system so that the system works as expected.
  - ✓ Recovery testing is also referred to as **Disaster Recovery Testing**. A lot of companies have disaster recovery centers to make sure that if any of the systems is damaged or fails due to some reason, then there is back up to recover from the failure.
  - ✓ **Risk elimination** is possible as the potential flaws are detected and removed from the system.
  - ✓ **Improved performance** as faults is removed and the system becomes more reliable and performs better in case a failure occurs.

- **Disadvantages of Recovery testing:**
  - ☒ Recovery testing is **a time-consuming process** as it involves multiple steps and preparations before and during the process.
  - ☒ The **recovery personnel must be trained** as the process of recovery testing takes place under his supervision. So, the tester needs to be trained to ensure that recovery testing is performed in the proper way. For performing recovery testing, he should have enough data and back up files to perform recovery testing.
  - ☒ The **potential flaws or issues are unpredictable in a few cases**. It is difficult to point out the exact reason for the same, however, since the quality of the software must be maintained, so random test cases are created and executed to ensure such potential flaws are removed.

- **Recovery:**
  - Recovery Testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures etc.
  - The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss.
  - Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point.
  - For Example, when an application is receiving data from the network, unplug the connecting cable.
    - ✓ After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection was broken.
    - ✓ Restart the system while a browser has a definite number of sessions open and check whether the browser is able to recover all of them or not.
  - A system or software should be recovery tested for failures like:
    - ⇨ Power supply failure
    - ⇨ The external server is unreachable
    - ⇨ Wireless network signal loss
    - ⇨ Physical conditions
    - ⇨ The external device not responding
    - ⇨ The external device is not responding as expected, etc.
  - The time taken to recover depends upon:
    - ⇨ The number of restart points
    - ⇨ A volume of the applications
    - ⇨ Training and skills of people conducting recovery activities and tools available for recovery.
  - When there are a number of failures then instead of taking care of all failures, the recovery testing should be done in a structured fashion which means recovery testing should be carried out for one segment and then another.
  - It is done by professional testers. Before recovery testing, adequate backup data is kept in secure locations. This is done to ensure that the operation can be continued even after a disaster.

- **Restoration Strategy:**
  - The recovery team should have their unique strategy for retrieving the important code and data to bring the operation of the agency back to normalcy.

- The strategy can be unique to each organization based on the criticality of the systems they are handling.
- The possible strategy for critical systems can be visualized as follows:
  - ⇨ To have a single backup or more than one
  - ⇨ To have multiple back-ups at one place or different places
  - ⇨ To have an online backup or offline backup
  - ⇨ Can the backup be done automatically based on a policy or to have it manually?
  - ⇨ To have an independent restoration team or development team itself can be utilized for the work

- **Advantages of Recovery Testing:**
  - ✓ **Improves the quality of the system** by eliminating the potential flaws in the system so that the system works as expected.
  - ✓ Recovery testing is also referred to as **Disaster Recovery Testing**. A lot of companies have disaster recovery centers to make sure that if any of the systems is damaged or fails due to some reason, then there is back up to recover from the failure.
  - ✓ **Risk elimination** is possible as the potential flaws are detected and removed from the system.
  - ✓ **Improved performance** as faults is removed and the system becomes more reliable and performs better in case a failure occurs.

- **Disadvantages of Recovery testing:**
  - ☒ Recovery testing is a **time-consuming process** as it involves multiple steps and preparations before and during the process.
  - ☒ The **recovery personnel must be trained** as the process of recovery testing takes place under his supervision. So, the tester needs to be trained to ensure that recovery testing is performed in the proper way. For performing recovery testing, he should have enough data and back up files to perform recovery testing.
  - ☒ The **potential flaws or issues are unpredictable in a few cases**. It is difficult to point out the exact reason for the same, however, since the quality of the software must be maintained, so random test cases are created and executed to ensure such potential flaws are removed.

3.7 Acceptance Testing: Overview, Types of Acceptance Testing:

- Acceptance testing is a phase after system testing that is normally done by the customers.
- A customer defines a set of test cases that will be executed to qualify and accept the product.
- These test cases are executed by customers themselves to quickly judge the quality of product before deciding to buy the product.
- Acceptance test cases failing in a customer site may cause the product to be rejected and may mean financial loss or may mean rework or product involving effort and time.

• **Acceptance Criteria:**

- There are basically three acceptance criteria which are as follows:

1. Product Acceptance:

- During the requirements phase, each requirement is associated with acceptance criteria. It is possible that one or more requirements may be mapped to form acceptance criteria.
- Whenever there are changes to requirements, the acceptance criteria are accordingly modified and maintained.
- Testing for faithfulness to any specific legal or contractual terms is included in the acceptance criteria.

2. Procedure Acceptance:

- Acceptance criteria can be defined on the procedures followed for delivery. An example of procedure acceptance could be documentation and release media.
- Some examples of acceptance criteria of this nature are as follows:
  - ⇨ User, administration and troubleshooting documentation should be part of the release.
  - ⇨ Along with binary code, the source code of the product with build script to be delivered in a CD.
  - ⇨ A minimum of 20 employees are trained on the product usage prior to deployment.
- These procedural acceptance criteria are verified/tested as part of acceptance testing.

3. Service Level Agreements:

- Service level agreements can become part of acceptance criteria and are generally part of a contract signed by the customer and product organization.
- The important contract items are taken and verified as part of acceptance testing.

- For example, time limits to resolve those defects can be mentioned part of SLA such as-
  - ⇨ All major defects that come up during first three months of deployment need to be fixed free of cost;
  - ⇨ Downtime of the implemented system should be less than 0.1%;
  - ⇨ All major defects are to be fixed within 48 hours of reporting.

- **Advantages of Acceptance Testing:**
  - ✓ It increases the satisfaction of clients as they test application itself.
  - ✓ The quality criteria of the software are defined in an early phase so that the tester has already decided the testing points. It gives a clear view to testing strategy.
  - ✓ The information gathered through acceptance testing used by stakeholders to better understand the requirements of the targeted audience.
  - ✓ It improves requirement definition as client tests requirement definition according to his needs.

- **Disadvantaged of Acceptance Testing:**
  - ☒ Customers are not willing to do that; it defeats the whole point of acceptance testing.
  - ☒ If test cases are written by someone else, the customer does not understand them, so tester has to perform the inspections by themselves only.

7. _____ Testing is a software testing process that checks whether the software can perform a failure-free operation for a specified time period in a particular environment.
   A. Reliability B. Security
   C. Performance D. Recovery

8. _____ testing is a phase after system testing that is normally done by the customers.
   A. System B. Unit
   C. Integration D. Acceptance

9. FIT stands for _____.
   A. Final Integration Testing B. First Integration Testing
   C. Full Integration Testing D. Finish Integration Testing

10. _____ is considering as an acceptance standard.
    A. Product Acceptance B. Procedure Acceptance
    C. Service Level Agreement D. All of Above

Answer:

(1) Unit (2) Integeration (3) Internal Interfaces (4) Exported Interfaces
(5) Functional (6) System (7) Reliability (8) Acceptance
(9) Final Integration Testing (10) All of Above

❖ True/False:

1. A successful test is one that uncovers every possible error. (T/F)
2. A Unit testing can be performed Top-Bottom approach. (T/F) Bottom-up
3. Graph Based techniques are very appropriate for large systems. (T/F)
4. Service level agreements are generally part of a contract signed by the customer and product organization. (T/F)
5. Security testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures. (T/F) Recovery
6. Recovery testing Improves the quality of the system. (T/F)
7. Recovery testing is not a time-consuming process. (T/F)
8. Performance testing Include speed, capacity, scalability, and stability. (T/F)
9. Path testing focuses attention on program logic. (T/F)
10. Graph Based testing is not difficult to choose the important input in limited time. (T/F)

Answer:

(1) True (2) False (3) True (4) True (5) False
(6) True (7) False (8) True (9) True (10) False

Exercises

❖ Answer the Question.

1. What is Unit Testing? Explain Advantages and Disadvantages of Unit Testing.
2. What is Integration Testing? Discuss types of Integration Testing.
3. What is system Testing? Explain Advantages and Disadvantages of System Testing.
4. Write a note on Performance Testing.
5. Write a note on Functional Testing.
6. What is Recovery in testing? Explain
7. What is Acceptance Testing? Explain Acceptance Criteria.
8. Explain Path based technique for testing.

❖ Fill in the blanks. (MCQs)

1. _____ testing focuses on the modules to find the error.
   - A. System
   - B. Unit
   - C. Integration
   - D. Acceptance

2. Testing the Interaction between the Modules and interaction with other systems externally is called _____ testing.
   - A. Integration
   - B. Functional
   - C. System
   - D. Unit

3. _____ are those that provide communication across two modules within a project or product.
   - A. Exported Interfaces
   - B. Internal Interfaces
   - C. Outer Interface
   - D. Inner Interface

4. _____ are those that are visible outside the product to third party developers and solution providers.
   - A. Exported Interfaces
   - B. Internal Interfaces
   - C. Outer Interface
   - D. Inner Interface

5. _____ testing that is based on the specifications of the software.
   - A. Integration
   - B. Functional
   - C. System
   - D. Unit

6. _____ testing is conducted on the complete Integrated system.
   - A. Integration
   - B. Functional
   - C. System
   - D. Unit

7. _____ Testing is a software testing process that checks whether the software can perform a failure-free operation for a specified time period in a particular environment.
   A. Reliability B. Security
   C. Performance D. Recovery

8. _____ testing is a phase after system testing that is normally done by the customers.
   A. System B. Unit
   C. Integration D. Acceptance

9. FIT stands for _____.
   A. Final Integration Testing B. First Integration Testing
   C. Full Integration Testing D. Finish Integration Testing

10. _____ is considering as an acceptance standard.
   A. Product Acceptance B. Procedure Acceptance
   C. Service Level Agreement D. All of Above

Answer:

(1) Unit (2) Integeration (3) Internal Interfaces (4) Exported Interfaces
(5) Functional (6) System (7) Reliability (8) Acceptance
(9) Final Integration Testing (10) All of Above

❖ True/False:

1. A successful test is one that uncovers every possible error. (T/F)
2. A Unit testing can be performed Top-Bottom approach. (T/F) Bottom-up
3. Graph Based techniques are very appropriate for large systems. (T/F)
4. Service level agreements are generally part of a contract signed by the customer and product organization. (T/F)
5. Security testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures. (T/F) Recover
6. Recovery testing Improves the quality of the system. (T/F)
7. Recovery testing is not a time-consuming process. (T/F)
8. Performance testing include speed, capacity, scalability, and stability. (T/F)
9. Path testing focuses attention on program logic. (T/F)
10. Graph Based testing is not difficult to choose the important input in limited time. (T/F)

Answer:

(1) True (2) False (3) True (4) True (5) False
(6) True (7) False (8) True (9) True (10) False

☺ ☺ ☺ ☺ ☺

UNIT-4

Test Management

❖ **Test Planning:**

- Preparing a Test plan, Scope management, Decide Test Approach, Setting up Criteria for testing, Indentifying responsibilities, Staffing, training needs, Resource requirements, Test deliverables, Testing Tasks.

❖ **Test Management:**

- Choice of standards, Test infrastructure management, Test people management, Integrating with product release
- Test Process: Base line a test plan, Test case specification, Update of traceability
- Test reporting: Recommending product release

UNIT-4 Test Management

❖ Introduction:

Every project has definite beginning and definite end. The product or service is different in some distinguish way from all similar products or services. Because of we integrate Software Testing in each phase of the product and the software products do not have same type of features, testing also has different characteristics and what is tested in each version could not be same.

Testing itself is considered to be a unique project on its own, planning, execution, tracking and periodical reporting is essential. In this chapter we will see how can we plan to manage all these activities, so that the effective testing can be done in the designated time period.

4.1 Test Planning:

- **Preparing a Test Plan:**

Test plan plays an important role in the execution, tracking and reporting of the whole testing project which covers:

1. What things to be tested? This defines the scope of the testing project.
2. How the testing will be performed? This covers how the project will be divided into the small and manageable modules or tasks.
3. What resources (either machines or humans) are required to conduct tests?
4. What time duration is required to test particular function, module or project?
5. What risk is associated with each testing task?

- **Scope Management: Deciding Features to be tested or Not:**

Different testing teams do the testing for various phases of testing. There can be a single test plan for all the phases and all teams, which is known as master test plan or there can be separate test plans are there for different phases and different testing teams. Scope management refers to specify the scope of a test project. The following activities are included in the scope management.

1. Understanding what constitutes a release of a product.
2. Breaking down the product into multiple release by considering their features.
3. Assigning priority to each release based on features.
4. Deciding what features needs testing and what feature doesn't.
5. Conducting details of testing, so that estimation of the resource requirement can be made.

- **Deciding Test Strategy:**
  Once the product is divided into multiple releases based on its features, the next step is to find more details of what is to be tested, so that exact estimation of size, effort can be made and proper scheduling can be arranged. This activity includes:
  1. Type of testing for testing functionalities.
  2. Configurations and scenario for testing functionalities.
  3. Identify the proper integration testing method.
  4. What localization validation would be needed?
  5. Which non-functional tests are needed?

- **How to set Testing Criteria:**
  Different testing phase should have clear entry and exit criteria. To test various features, different type of tests, or combination of different tests has to be used. For each type of test, clear entry criteria have to specify which defines the start of the testing phase. Similarly, for each type of test cycle exit or completion criteria has to be specified which defines the end or completion of particular test cycle.

- **Identify Staff, Responsibility and Training:**
  After completion of the above activities, we need to select proper staff to conduct different tests, assign responsibility to that staff and we also need to do arrangement for the training to the staff if it is needed. Testing project requires different employees to play their role differently. Appointment of test leaders, test manager and testers, and assigning different responsibilities to each role plays vital role in the testing process.

- **Identify Resource Requirements:**
  As a part of the planning, after choosing the staff and assigning the responsibilities, providing resources is equally important activity. In the planning which testing team need which resources is planned out. These resources may include:
  1. Configuration of Machines (CPU, RAM, Hard-Disk etc.).
  2. Test automation tools.
  3. Some supporting tools like compilers, interpreters, data generators, Operating Systems etc.
  4. Any special hardware or software if it is need to conduct special type of test conduction.
  5. Appropriate number of licenses software.

- **Identify Test Deliverables:**
  Test Deliverables are nothing but the outcome, we are expecting from the testing team after completion of the various tests. In the planning phase we need to determine the

exact format of the Test Deliverables. The Test Deliverables may have following things:

1. The test plan itself (Master test plan or any other test plan given to perform).
2. Specification of test case design.
3. Test cases with any automation mention in the test plan.
4. Test logs generated during testing process.
5. Summary report of the test.

- **Estimation of Size and Effort:**

Estimation of size is an important parameter of software development process. After computing size, we can compute effort needs to be given to the software product and finally based on the effort we can estimate the cost of the software product.

Size of the product is obviously determining the amount of testing that needs to be done. If the product is large in size, then the testing effort will be increased. Few of the measures of the size of the product under test are listed below:

1. **LOC:** LOC or Lines of code is one to the parameter to estimate the size of the product. In this method how many lines of code need to write to implement the software product is estimated. Based on number of lines of the code first the size of the software product is estimated and then based on the size, effort and cost will be estimated. Even though, there are some problems are there in this method of size estimation, LOC is still a popular measure for estimation of the size of the software product.
2. **FP:** Function Point is another popular method of estimation of the size of the software product. It is difficult to estimated lines of code when the software is yet to build or under development. In this method the size of product is estimated based on the number of functions needs to write and number of IO screen required to implement the software. Lines of Code (LOC) size estimation method only focus on number of lines of code to be written in the software, whereas Function Point (FP) considers number of functions (code), inputs, outputs, interfaces, external data files and enquires too.

4.2 Test Management:

Test management most commonly refers to the activity of managing the testing process of the computer. A test management tool is software used to manage tests that can be an automated or manual, that have been previously specified by a test procedure. It is often associated with automation software. Test management tools often include requirements and specifications management modules that allow automatic generation

of the requirement test matrix, which is one of the main metrics to indicate functional coverage of a system under test.

Test Management includes building some bundles of test cases and execute them (or scheduling their execution). Execution can be either manual or automatic. The user will have to perform all the test steps manually and inform the result of the tests. There are several ways to run tests. The simplest way to run a test is to run a test case. The test case can be associated with other test artifacts such as test plans, test scripts, test environments, test case execution records, and test suites.

The ultimate goal of test management tools is to deliver sensitive metrics that will help the manager in evaluating the quality of the system under test before releasing.

4.2.1 Choice of Standards:

Standards plays important role in the planning of an organization. There are two types of standards – External Standards and Internal Standards.

External Standards are standards that a product should obey with, which are externally visible and are usually stipulated by external people. From a testing point of view, these standards include standard tests supplied by external people and acceptance tests supplied by customers. Compliance of the external standards is usually done by external parties.

Internal standards are standards formulated by the internal staff of the testing organization to bring consistency and predictability. Testing organization itself standardize the processes and working methods withing the organization. Internal Standards includes:

1. Naming conventions for the test artifacts.
2. Standards of document produced and used.
3. Test coding standards.
4. Test reporting standards.

⇨ **Naming convention for the test artifacts:** Every test artifact should have proper and meaningful name. If the test artifacts have proper and meaningful names then it made it easy to identify the tests artifacts. Proper and meaningful naming convention process also made easy mapping between the functionality and its corresponding test cases. For an example: Product P with modules M01, M02 and M03 are mapped with test suits PM01nnnn.<file type>, PM02nnnn.<file type>, PM03nnnn.<file type>. This two-way mapping between the product functionalities and tests enables identification of proper tests to be modify and run when product functionality changes.

⇨ **Documentation Standards:** Most organization set standards for documentation and coding for their ease communication convenience within the internal staff of the organization. Documentation standards are set of rules of preparing documentation. Documentation Standards specifies how to capture information of the particular test and its result. The documentation standard may include:

1. Suitable header to the beginning of a file that indicates the functions to be served by that particular test.
2. Enough in-line comments, which explains the functions served by the test.
3. Up-to-date information of change history and recording of all events.

⇨ **Test coding standards:** Test coding standards enforce how the test cases themselves are written. It goes one level deeper into the tests. These standards may be:

1. Enforce the right way to initialize the test, and indicates how the test makes the results autonomous of other tests.
2. Indicates the ways of how to name variables within the code to make sure the program is readable and understand it easily.
3. Test coding standard also encourage of reusability of test artifacts.
4. It also includes standards interfaces like hardware, operating system and so on to the external entities.

⇨ **Test reporting standards:** Testing process is tightly interlinked with the quality of the product; all the stakeholders must get timely and consistent view of the test progress. Test reporting addresses this issue and provides guidelines to the testers, up to which of details should be produced in the reports.

4.2.2 Test Infrastructure Management:

Testing needs a robust infrastructure to be planned in advance. The essential elements of the Test Infrastructure management are listed below:

1. Test Case Database (TCDB)
2. Defect Repository (DR)
3. Software Configuration Management Tool and Repository (SCM).

Test Case Database (TCDB) is a database which records all required information about all test cases of an organization. Some entities and its attributes are given in the following table.

Entity	*Attributes*	*Description*
TestCase	TestCaseID TestCaseName TestCaseOwner	Stores the static details of the test cases.

	AssociatedFiles	
TestCase_Product_CrossReference	TestCaseID ModuleID	This table stores the mapping details between various modules and its test cases.
TestCaseExecutionDetails	TestCaseID Execution_Date Time_taken TestResult	This table records the details of the execution of a test case and its results
TestCase_Defect	TestCaseID DefectIDRef (Points to defect Repository)	This table maps the defect with particular test is executed on particular project.

Defect repository records all the details related to defect reported on particular product. The defect repository plays an important role in communication that affect work flow within an organization. The Defect Repository records the following information:

Entity	*Attributes*	*Description*
Defect_Details	DefectID Defect_Priority Description Affected_Products Product_Version Environment_Info Problem_Statement DateTime_Defect	Stores all the static information about defect.
Defect_Test	TestCaseID DefectID	Map the Test Case with particular Defect.
Fix_Details	DefectID Defect_Fix_Details	Map the Defect with Steps taken to fix the defect.
Communication	TestCaseID DefectID Communication_Details	Records the communication done between internal staff to resolve defect on particular test case.

Apart from Test Case Database (TCDB) and Defect Repository (DR) one more repository is needed in Test Management is Software Configuration Management repository. It is also as Configuration Management or CM repository, which maintain change control and version control of all the files which are associated to particular software product. Change control ensures the following things:

1. Changes made in the particular Test file, is done by the authorize person and in controlled fashion.
2. Changes made by a test engineer should not be lost or overwritten by another test engineer.
3. Each change (at any particular time) generates a unique new version of the file.
4. Every test engineer should have access to the recent version of test file at any point of time.

Test Case Database (TCDB), Defect Repository (DR) and Software Configuration Management (SCM) repository are complements to each other and work together in integrated manner as shown in the following figure. DR links different defects, their fixes and tests, where the necessary files will be in SCM. Meta data about the modified files of tests will be in TCDB.

4.2.3 Test Process:

Test process is a process of testing any software project to improve the quality of that software. Test process is a process of series of main activities like: [1] Preparing test plans [2] Designing test cases [3] Execution of test cases and [4] Reporting outcome of test cases. The test process can be explained with following activities:

[1] Baselining Test Plan:

A Test Plan represent multiple points in the form of single document, which will act as an anchor for the whole testing process. An organization design several templates for the test cases and each testing project has to be tested based on the template. Any change in the template is applied after careful deliberations. Before any modification in the template, the test plan is revied by many designated people of an organization. It will be then approved by competent authority and then it will be applied into the template and then test plan is baselined into the Software Configuration Management (SCM) repository.

[2] Test Case Specification:

With the help of test plan, testing team will design the test case specifications, which will become the base in the preparation of individual test cases. The test case should have following things:

1. Purpose of Test Case.

2. Items to be tested with their versions and releases.
3. Environment needed to conduct the test case.
4. Input data to be used in the test case.
5. Steps to execute test case.
6. Expected result or outcome.
7. Steps to compare actual result with expected result.
8. Relationship if any between this test case to other test cases.

[3] Developing Test Cases:

Test engineer needs to develop Test Cases based on Test Case Specification. In this phase, test engineer can choose any method, that testing is done using manual test cases or by using automation system. Here the specification is transformed into the Test Cases. If the test engineer has chosen, automation instead of writing manual test cases, then in this phase a test engineer has to write automation scripts.

[4] Executing Test Cases:

The prepared test cases will be executed at appropriate time on the project. During the execution of the test cases the defect repository will be updated with:

1. Defects from the earlier test cycles that are fixed in the recent build.
2. New defects if any, found during the current test run.

[5] Preparation of Test Summary Report:

After completion of a test cycle, a detailed summary report of the testing is produced. This test summary report will be observed by senior management and decide the fitness of the product to be release.

4.2.4 Test Reporting:

Process of testing needs continues communication between the testing team and other teams. There are two types communications that are needed: [1] Test Incident report and [2] Test summary report.

[1] Test incident report:

Test incident report is basically a communication report, used during the testing cycle when the defect is found in the project. Test incident report is nothing but data entry is made in the defect repository when the defect is detected during the testing process. Those defects which have high impact on the project is mentioned in the test summary report.

[2] Test Cycle Report:

Testing project needs many test cycles to complete. During each test cycle planning and execution of various test cases are conducted. After completion of each test cycle a

newer build of product is form, which will be more stable version of the product then it's previous build. At the end of each test cycle, a test cycle report is produced which gives the following information:

1. A summary of all activities conducted during entire test cycle.
2. Details of the uncovered defects, detected during the test cycle.
3. Progress of the current build compare to the previous build of product.
4. Defects which are pending to be fix in the current test cycle.
5. Variation if any observed in effort or schedule.

[3] Test Summary Report:

Test summary report is the final report produced after completion of all test cycle and software product is ready to release. Test summary report summarizes the results of a test cycle reports produce during each test cycle.

Usually, test summary reports are of two types:

1. Phase-wise test summary report.
2. Final test summary report. (Which is produced at the time of product release)?

A summary report should present the following things:

1. A detailed summary of the activity conducted during the test cycle.
2. Difference if any observed between the activity performed during testing and actually planned.
3. Summary of the result like test that failed with its reasons, impact of the defect on the project which is uncovered yet etc.
4. Complete evaluation and recommendation for the release.

Exercises

❖ Answer the following Questions in brief.

1. What is test management? Explain choice of standards in brief.
2. Explain elements of test infrastructure management in test management.
3. Write a short note on test process.
4. What is test reporting? Explain different types of test summary reports.
5. What is test planning? List all the steps of it. Explain any two in detail.

❖ Fill in the blanks.

1. The _____ criteria specify when a test activity is started.
2. The _____ criteria specify when a test cycle can be completed.
3. A test cycle is an isolated activity. [True/ False].
4. _____ is done based on estimation of effort involved and availability of time for release.
5. _____ standards are externally visible to the customers.
6. _____ standards are formulated by a testing organization.
7. The elements of test infrastructure management are _____, _____ and _____.
8. TCDB stands for _____.
9. DR stands for _____.
10. SCM stands for _____.
11. The _____ contains all the information about the test cases in an organization.
12. The _____ captures all the details of defects reported for a product.

13. A _____ keeps track of change control and version control of all the files.

14. A _____ is a tool used to validate that every requirement is tested.

15. Using the test plan, the _____ is designed.

16. A report that summarizes the results of a test cycle is the _____ report.

17. A _____ is a communication that happens through the testing cycle when defects are encountered.

18. A _____ gives summary of the activities carried out during the testing cycle.

Answers:

Que	Answer	Que	Answer
1	Entry	10	Software Configuration Management
2	Exit	11	TCDB
3	False	12	DR
4	Staffing	13	SCM or CM
5	External	14	Traceability Matrix
6	Internal	15	Test case specification
7	TCDB, DR, CM	16	Test summary report
8	Test Case Database	17	Test incident report
9	Defect Repository	18	Test cycle report

GUJARAT UNIVERSITY
April – 2021
B.C.A SEM – VI

Subject: CC-307 Software Testing

Time: 2:00 Hrs. **Model Paper** **Marks: 50**

Instruction: All Questions of **section I** carry equal marks.

Attempt any **two** Questions in **section I**

Question 5 in section II **is COMPULSORY,** Attempt any **Five.**

Section- 1

Q.1	A	What is Software Testing? Define the goals of Software Testing.	10
	B	Explain Software Testing Life Cycle.	10
Q.2	A	What is Black Box Testing? Explain Boundary Value Analysis in detail.	10
	B	Explain Inspection Process of Static Testing.	10
Q.3	A	What is Unit Testing? Explain Drivers and Studs.	10
	B	Explain Function testing in detail.	10
Q.4	A	Explain the key element of Test Management.	10
	B	Explain The Hierarchy of Testing Group in detail.	10

Section- 2

Q.5 **Answers the following Questions (Any 5 x 2 marks each)** 10

1. Testing is the process of _____errors.

   (A) Hiding (B) Finding

   (C) Removing (D) All the above

2. Risk management is _____goal of software testing

   (A) Long-Term (B)Short-Term

   (C) Post-implementation (D) All the above

3. Testing is a Random Process (True/False).

4. Black-Box Testing is functional Testing (True/False).

5. Whit-box Testing is_____Testing.

   (A) Static (B) Dynamic (C) Both A and B (D) None of these

6. A node with more than one arrow leaving it is called a _____

   (A) decision node (B) junction node

   (C) region (D) all of the above

7. In static testing, a bug is found at its_____location.

   (A) Exact (B) Nearby

   (C) Both A and B (D) None of the above

8. Unit Testing is implement on Module of Software(True/False)

9. A test cycle is an isolated activity.(True/False)

10. TCDB stands for _____

    (A) Test Case Design Building (B) Test Case Database

    (C) Test Case Design Block (D) Test Case Database Block