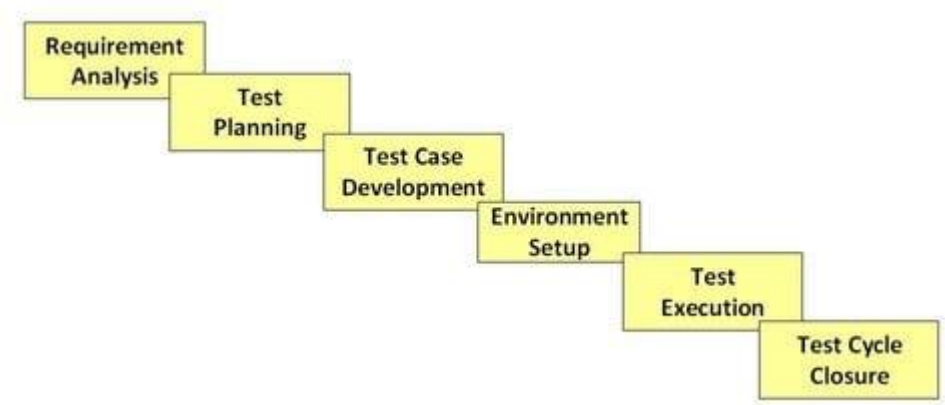
SOFTWARE TESTING LIFE CYCLE (STLC)

What is Software Testing Life Cycle (STLC)?

- **Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met.**
- **STLC involves both verification and validation activities. Contrary to popular belief, Software Testing is not just a single/isolate activity, i.e. testing.**
- **It consists of a series of activities carried out methodologically to help certify your software product.**
- **STLC stands for Software Testing Life Cycle.**

STLC Phases:



STLC Model Phases

1. **Requirement Analysis**
2. **Test Planning**
3. **Test case development**
4. **Test Environment setup**
5. **Test Execution**
6. **Test Cycle closure**

What is Entry and Exit Criteria in STLC?

- **Entry Criteria: Entry Criteria gives the prerequisite items that must be completed before testing can begin.**
- **Exit Criteria: Exit Criteria defines the items that must be completed before testing can be concluded**

1. Requirement Phase Testing

- **Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail.**
- **Requirements could be either functional or non-functional.**
- **Automation feasibility for the testing project is also done in this stage.**

Activities in Requirement Phase Testing

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

Deliverables of Requirement Phase Testing

- RTM
- Automation feasibility report. (if applicable)

2. Test Planning in STLC

- Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project.
- Moreover, the resources, test environment, test limitations and the testing schedule are also determined.
- The Test Plan gets prepared and finalized in the same phase.

Test Planning Activities

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

Deliverables of Test Planning

- Test plan /strategy document.
- Effort estimation document.

3. Test Case Development Phase

- The Test Case Development Phase involves the creation, verification and rework of test cases & test scripts after the test plan is ready.
- Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions.
- Then the QA team starts the development process of test cases for individual units.

Test Case Development Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts

- **Create test data (If Test Environment is available)**

Deliverables of Test Case Development

- **Test cases/scripts**
- **Test data**

4. Test Environment Setup

- **Test Environment Setup decides the software and hardware conditions under which a work product is tested.**
- **It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase.**
- **Test team may not be involved in this activity if the development team provides the test environment.**
- **The test team is required to do a readiness check (smoke testing) of the given environment.**

Test Environment Setup Activities

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

Deliverables of Test Environment Setup

- **Environment ready with test data set up**
- **Smoke Test Results.**

5. Test Execution Phase

- **Test Execution Phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared.**
- **The process consists of test script execution, test script maintenance and bug reporting.**
- **If bugs are reported then it is reverted back to development team for correction and retesting will be performed.**

Test Execution Activities

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

Deliverables of Test Execution

- **Completed RTM with the execution status**
- **Test cases updated with results**
- **Defect reports**

6. Test Cycle Closure

- **Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results.**
- **Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle.**
- **The idea is to remove process bottlenecks for future test cycles.**

Test Cycle Closure Activities

- **Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality**
- **Prepare test metrics based on the above parameters.**
- **Document the learning out of the project**
- **Prepare Test closure report**
- **Qualitative and quantitative reporting of quality of the work product to the customer.**
- **Test result analysis to find out the defect distribution by type and severity.**

Deliverables of Test Cycle Closure

- **Test Closure report**
- **Test metrics**

BUG LIFE CYCLE

What is Defect/Bug Life Cycle?

- **Defect Life Cycle or Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life.**
- **The purpose of Defect life cycle is to easily coordinate and communicate current status of defect which changes to various assignees and make the defect fixing process systematic and efficient.**

Defect Status:

- **Defect Status or Bug Status in defect life cycle is the present state from which the defect or a bug is currently undergoing.**
- **The goal of defect status is to precisely convey the current state or progress of a defect or bug in order to better track and understand the actual progress of the defect life cycle.**

Defect States Workflow:

New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.

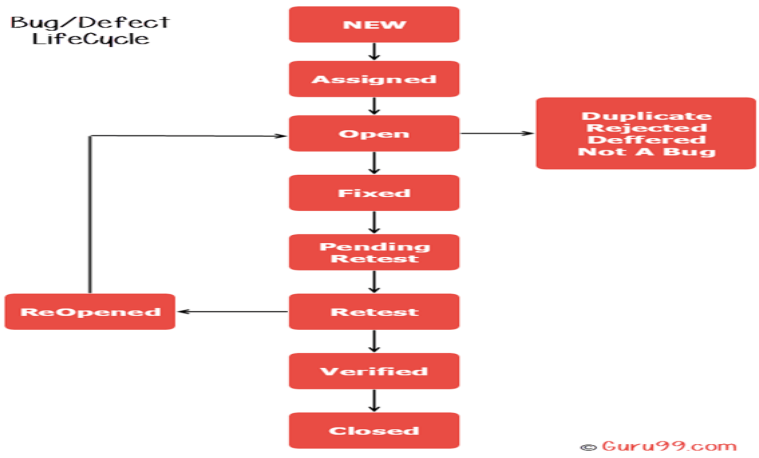
Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team

Open: The developer starts analyzing and works on the defect fix

Fixed: When a developer makes a necessary code change and verifies the change, he or she can make bug status as "Fixed."

Pending retest: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is "pending retest."

Retest: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to "Re-test."



Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."

Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.

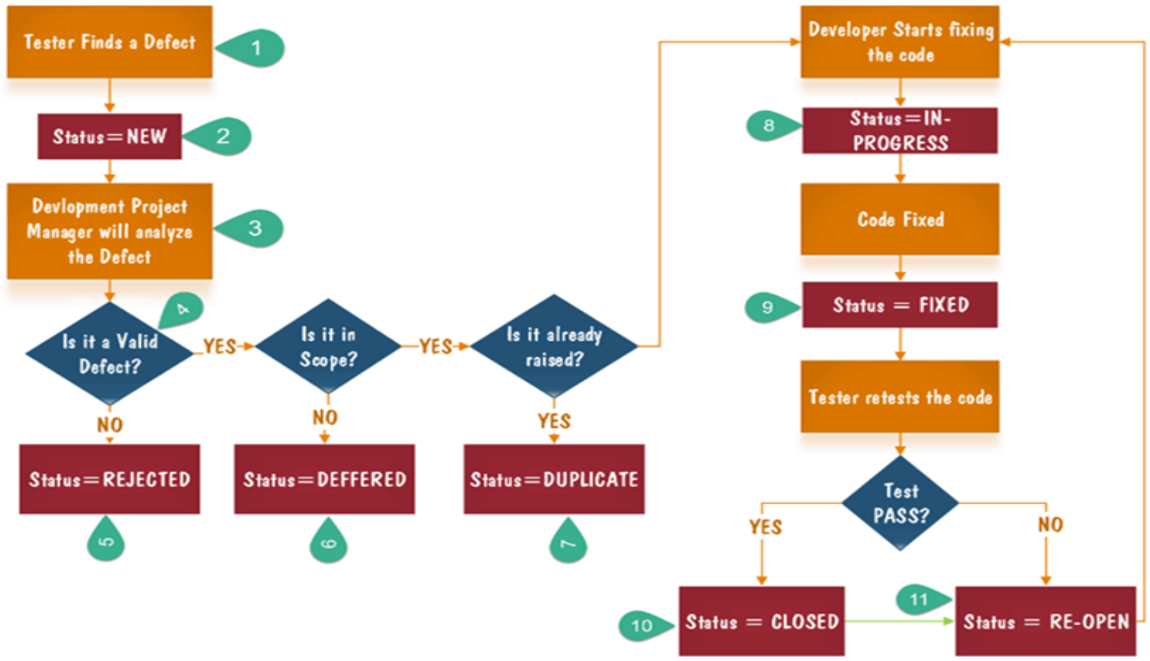
Closed: If the bug is no longer exists then tester assigns the status "Closed."

Duplicate: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to "duplicate."

Rejected: If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."

Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs.

Not a bug: **If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".**



Steps:

1. **Tester finds the defect**
2. **Status assigned to defect- New**
3. **A defect is forwarded to Project Manager for analyze**
4. **Project Manager decides whether a defect is valid**
5. **Here the defect is not valid- a status is given "Rejected."**
6. **So, project manager assigns a status rejected. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a postponed or deferred status.**
7. **Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status duplicate.**
8. **If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status in- progress.**
9. **Once the code is fixed. A defect is assigned a status fixed**
10. **Next, the tester will re-test the code. In case, the Test Case passes the defect is closed. If the test cases fail again, the defect is re-opened and assigned to the developer.**
11. **Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be re-opened.**

LEVELS OF TESTING:

1. Unit testing

2. Integration Testing

3. System testing

4. Acceptance testing

1. UNIT TESTING:

- **Unit testing is a method of testing individual units or components of a software application.**
- **It is typically done by developers and is used to ensure that the individual units of the software are working.**
- **Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.**

ADVANTAGE:

- **It helps to ensure that changes to the code do not introduce new bugs.**
- **It helps to identify bugs early in the development process, before they become more difficult and expensive to fix.**
- **It helps to improve the overall quality and reliability of the software.**

2. INTEGRATION TESTING:

Integration testing is a method of testing how different units or components of a software application interact with each other.

ADVANTAGE:

- **It helps to identify and resolve issues that may arise when different units of the software are combined.**
- **It helps to ensure that the different units of the software work together.**
- **It helps to improve the overall reliability and stability of the software.**

INTEGRATION TESTING CAN BE PERFORMED IN DIFFERENT WAYS:

1. **Top-down integration testing: It starts with the highest level modules and differentiate them with lower-level modules.**

2. **Bottom-up integration testing: It starts with the lowest-level modules and integrates them with higher-level modules.**

3. **Big-Bang integration testing: It combines all the modules and integrates them all at once.**

4. **Incremental integration testing: It integrates the modules in small groups, testing each group as it is added.**

3. SYSTEM TESTING:

- **System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both.**
- **The software is tested such that it works fine for the different operating systems.**

- **It is covered under the black box testing technique.**
- **In this, we just focus on the required input and output without focusing on internal working.**

4.ACCEPTANCE TESTING:

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in requirements.

ADVANTAGE:

- **Improved software quality and reliability**
- **Early identification and fixing of defects**
- **Improved customer satisfaction**
- **Increased stakeholder confidence**
- **Reduced maintenance costs**

DISADVANTAGE:

- **Time-consuming and adds to project cost**
- **Can slow down development process**
- **Not all defects can be found**
- **Can be difficult to fully test complex systems**
- **Potential for human error during testing process**

FUNCTIONAL TESTING:

- **Functional testing focuses on whether the software or system works as intended, i.e., whether it meets the functional requirements.**
- **It involves testing the features and functionalities of the software, such as input/output, error handling, and user interface.**

For example, if you are testing a calculator app, you would check whether it can perform basic arithmetic operations like addition, subtraction, multiplication, and division correctly. You would also check if the user interface is user-friendly, the buttons are working correctly, and the app is responsive to user input.

NON-FUNCTIONAL TESTING:

- **Non-functional testing, on the other hand, is focused on testing the non-functional aspects of the software, such as *performance, security, usability, reliability, and compatibility*.**
- **Non-functional testing helps to ensure that the software meets the quality standards and performs well under different conditions.**

For example, if you are testing a website, you would check whether it can handle many concurrent users without slowing down or crashing. You would also check whether the website is secure from potential cyber threats like hackers or viruses. Additionally, you would check whether the website is accessible and easy for people with disabilities.

METHODS OF TESTING:

1. **Black box testing**
2. **White box testing**
3. **Grey box testing**

1. BLACK BOX TESTING:

- **Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.**
- **Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.**
- **It is also known as Behavioral Testing.**

For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

TYPES:

1.Functional testing – This black box testing type is related to the functional requirements of a system; it is done by software testers.

2.Non-functional testing – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

3.Regression testing – It is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

HOW TO DO BLACK BOX TESTING IN SOFTWARE ENGINEERING:

- **Initially, the requirements and specifications of the system are examined.**
- **Tester chooses valid inputs (positive test scenario) to check whether SUT (System Under Test) processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.**
- **Tester determines expected outputs for all those inputs.**
- **Software tester constructs test cases with the selected inputs.**
- **The test cases are executed.**
- **Software tester compares the actual outputs with the expected outputs.**
- **Defects if any are fixed and re-tested.**

TOOLS:

For Functional/ Regression Tests, you can use – QTP, Selenium

For Non-Functional Tests, you can use – LoadRunner, Jmeter

TECHNIQUES:

1. **Boundary value analysis**
2. **Equivalence partitioning technique**
3. **Decision table**
4. **State transition technique**
5. **All pair testing technique**
6. **Use case technique**
7. **Error guessing**
8. **Cause effect technique**

1. BOUNDARY VALUE ANALYSIS:

- **Boundary value analysis is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition).**
- **A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries.**
- **Normally Boundary value analysis is part of stress and negative testing.**
- **Using Boundary Value Analysis technique tester creates test cases for required input field.**

EXAMPLE,

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

2. EQUIVALENCE PARTITIONING TECHNIQUE:

- **Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.**
- **If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false.**
- **The equivalence partitions are derived from requirements and specifications of the software.**
- **The advantage of this approach is, it helps to reduce the time of testing due to a smaller number of test cases from infinite to finite.**
- **It is applicable at all levels of the testing process.**

EXAMPLE,

INVALID 1 Test case	INVALID 2 Test case	VALID 3 Test case	VALID
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

3. DECISION TABLE:

- **Decision table technique is one of the widely used case design techniques for black box testing.**
- **This is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form.**
- **That's why it is also known as a cause-effect table.**
- **This technique is used to pick the test cases in a systematic manner; it saves the testing time and gives good coverage to the testing area of the software application.**
- **Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.**
- **This technique is related to the correct combination of inputs and determines the result of various combinations of input.**

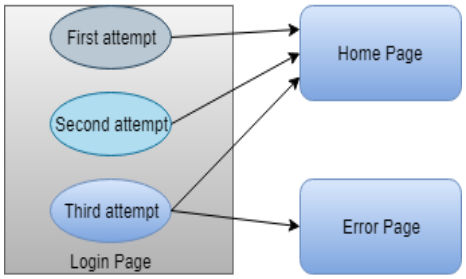
EXAMPLE,

Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email

4. STATE TRANSITION TECHNIQUE:

- **The general meaning of state transition is, different forms of the same situation, and according to the meaning, the state transition method does the same.**
- **It is used to capture the behavior of the software application when different input values are given to the same function.**

EXAMPLE,



STATE TRANSITION TABLE:

STATE	LOGIN	VALIDATION	REDIRECTED
S1	First Attempt	Invalid	S2
S2	Second Attempt	Invalid	S3
S3	Third Attempt	Invalid	S5
S4	Home Page		
S5	Error Page		

5. ALL PAIR TESTING TECHNIQUE:

- **All-pairs testing technique is also known as pairwise testing.**

- It is used to test all the possible discrete combinations of values.
- This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
- Suppose, you have a function of a software application for testing, in which there are 10 fields to input the data, so the total number of discrete combinations is 10 ^ 10 (100 billion), but testing of all combinations is complicated because it will take a lot of time.

EXAMPLE,

1. Check Box = 2

2. List Box = 1

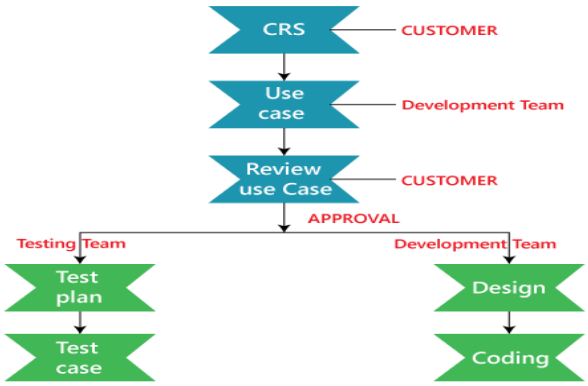
3. Radio Button = 2

4. Text Box = 100

5. Total number of test cases = 2*10*2*10 = 4000

6. USE CASE TECHNIQUE:

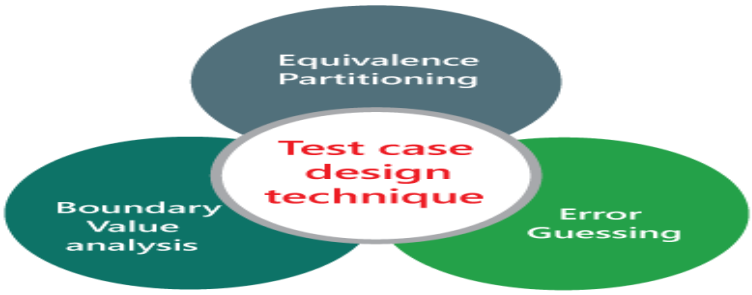
- The use case is functional testing of the black box testing used to identify the test cases from the beginning to the end of the system as per the usage of the system.
- By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.
- It is a graphic demonstration of business needs, which describe how the end-user will cooperate with the software or the application.



7. ERROR GUESSING TECHNIQUE:

- Error guessing is a technique in which there is no specific method for identifying the error.
- It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
- It is a type of black box testing technique which does not have any defined structure to find the error.

- In software testing, we have three different test case design techniques which are as follows:
    - Error Guessing
    - Equivalence Partitioning
    - Boundary Value Analysis[BVA]



- The main purpose of this technique is to identify common errors at any level of testing by exercising the following tasks:
    1. Enter blank space into the text fields.
    2. Null pointer exception.
    3. Enter invalid parameters.
    4. Divide by zero.
    5. Use maximum limit of files to be uploaded.
    6. Check buttons without entering values.

EXAMPLE,

A function of the application requires a mobile number which must be of 10 characters. Now, below are the techniques that can be applied to guess error in the mobile number field:

- What will be the result, if the entered character is other than a number?
- What will be the result, if entered characters are less than 10 digits?
- What will be the result, if the mobile field is left blank?

After implementing these techniques, if the output is similar to the expected result, the function is considered to be bug-free, but if the output is not similar to the expected result, so it is sent to the development team to fix the defects.

However, error guessing is the key technique among all testing techniques as it depends on the experience of a tester, but it does not give surety of highest quality benchmark. It does not provide full coverage to the software. This technique can yield a better result if combined with other techniques of testing.

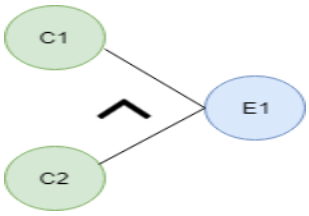
7. CAUSE EFFECT TECHNIQUE:

- Cause-effect graph comes under the black box testing technique which underlines the relationship between a given result and all the factors affecting the result.
- It is used to write dynamic test cases.

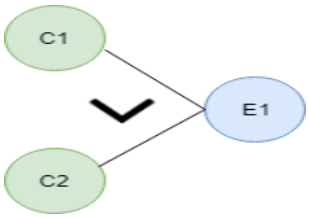
- The dynamic test cases are used when code works dynamically based on user input.
- Cause-Effect graph technique is based on a collection of requirements and used to determine minimum possible test cases which can cover a maximum test area of the software.

NOTATIONS:

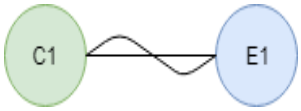
AND - E1 is an effect and C1 and C2 are the causes. If both C1 and C2 are true, then effect E1 will be true.



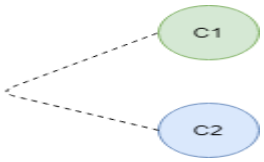
OR - If any cause from C1 and C2 is true, then effect E1 will be true.



NOT - If cause C1 is false, then effect E1 will be true.



Mutually Exclusive - When only one cause is true.



For example,

While using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.

---

2. WHITE BOX TESTING:

- White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security.
- In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

WHAT DO YOU VERIFY IN WHITE BOX TESTING?

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

HOW DO YOU PERFORM IN WHITE BOX TESTING?

Step-1. Understand the code

Step-2. Create test cases and execute

TECHNIQUES:

1. Statement testing

2. Decision testing

3. Control flow testing

4. Data flow/ Path testing

5. Branch testing

1. STATEMENT TESTING:

- This technique involves execution of all statements of the source code at least once.
- It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.

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

EXAMPLE,

If a = 5, b = 4

```
print (int a, int b) {

int sum = a+b;

if (sum>0)

print ("This is a positive result")

else
```

print ("This is negative result")

}

Total number of statements = 7

Number of executed statements = 5

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

Statement coverage = 5/7*100

= 500/7

= 71%

2. DECISION TESTING:

- Decision coverage technique comes under white box testing which gives decision coverage to Boolean values.
- This technique reports true and false outcomes of Boolean expressions.
- Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.
- Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two.
- The percent of decision coverage can be found by dividing the number of exercised outcome with the total number of outcomes and multiplied by 100.

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} *100$$

EXAMPLE,

Value of a is 7 (a=7)

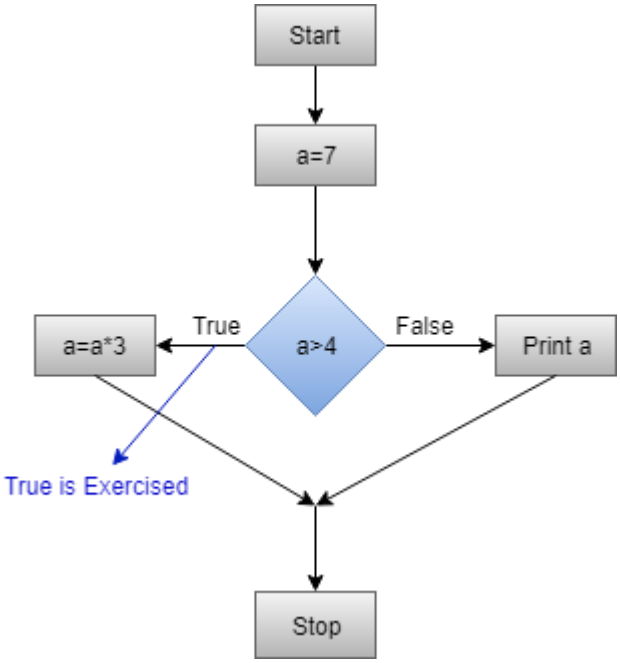
Test (int a=7)

{ if (a>4)

a=a*3

print (a)

}



$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

Decision Coverage = ½*100 (Only "True" is exercised)
=100/2
= 50

Decision Coverage is 50%

3. CONTROL FLOW TESTING:

- **The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure.**
- **The control structure of a program is used to develop a test case for the program.**
- **In this technique, a particular part of a large program is selected by the tester to set the testing path.**
- **It is mostly used in unit testing**

NOTATIONS:

1. **Node**
2. **Edge**
3. **Decision Node**
4. **Junction node**

1.Node - Nodes in the control flow graph are used to create a path of procedures.

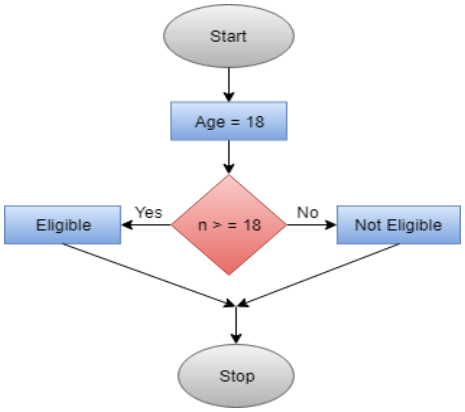
2.Edge - Edge in control flow graph is used to link the direction of nodes.

3.Decision node - Decision node in the control flow graph is used to decide next node of procedure as per the value.

4.Junction node - Junction node in control flow graph is the point where at least three links meet.

EXAMPLE,

```
public class VoteEligiblityAge{

public static void main(String []args){

int n=45;

if(n>=18)

{

 System.out.println("You are eligible for voting");

} else

{

System.out.println("You are not eligible for voting");

}

}

}
```



4.DATA FLOW/ PATH TESTING:

- **Data flow testing is used to analyze the flow of data in the program.**
- **It is the process of collecting information about how the variables flow the data in the program.**
- **It tries to obtain particular information of each particular point in the process.**

EXAMPLE,

1. read x;	
2. If(x>0)	(1, (2, t), x), (1, (2, f), x)
3. a= x+1	(1, 3, x)
4. if (x<=0) {	(1, (4, t), x), (1, (4, f), x)
5. if (x<1)	(1, (5, t), x), (1, (5, f), x)
6. x=x+1; (go to 5)	(1, 6, x)
else	
7. a=x+1	(1, 7, x)
8. print a;	(6,(5, f)x), (6,(5,t)x)
	(6, 6, x)
	(3, 8, a), (7, 8, a).

OUTPUT:

Set x= 1

Path - 1, 2, 3, 8

Set x= -1

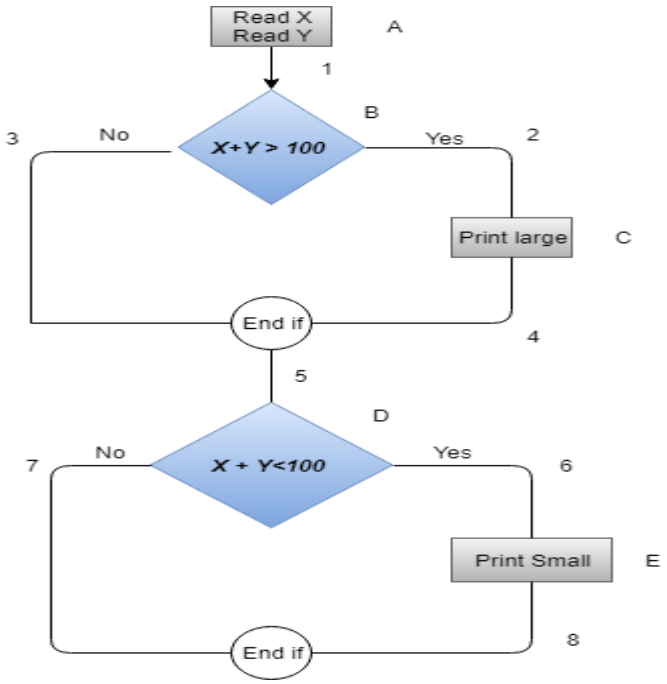
Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

5.BRANCH TESTING:

- **Branch coverage technique is used to cover all branches of the control flow graph.**
- **It covers all the possible outcomes (true and false) of each condition of decision point at least once.**
- **Branch coverage technique is a whitebox testing technique that ensures that every branch of each decision point must be executed.**

EXAMPLE,

```
Read X
Read Y
IF X+Y > 100 THEN
Print "Large"
ENDIF
If X + Y<100 THEN
Print "Small"
ENDIF
```



In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is A1-B2-C4-D6-E8, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

Path 1 - A1-B2-C4-D6-E8

Path 2 - A1-B3-5-D7

Branch Coverage (BC) = Number of paths =2

ADVANTAGE:

- **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.**

DISADVANTAGE:

- **White box testing can be quite complex and expensive.**
- **Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed and can lead to production errors.**
- **White box testing requires professional resources with a detailed understanding of programming and implementation.**
- **White-box testing is time-consuming, bigger programming applications take the time to test fully.**

DIFFERENCE BETWEEN BLACK BOX AND WHITE BOX TESTING

White-box testing	Black box testing
The developers can perform white box testing.	The test engineers perform the black box testing.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.
