Test Cases

Test Cases

- Test case is used to test the function or a specific function of a software.
- Test case consists of test data, and conditions to verify a requirement.
- Test cases can be used to compare expected results and actual results.
- The functionality of the software product as per requirement is validated. E.g., check results of login and password

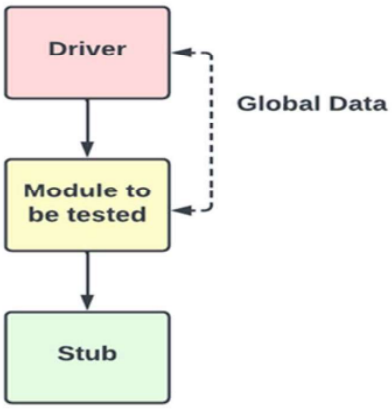
Test Stubs and Test Drivers

- Test stubs and drivers provide the complete environment to check the performance and the results.
- A stub is a simple dummy procedure with the same input-output parameters as the original procedure.
- If the module is not developed or is in progress, then a test stub is used.
- Test stubs are developed by developers to improve testing speed, especially for lower components.

Test Stubs and Test Drivers

- Test stubs consist of the following:
  - Messages to trace the data
  - Input values for components in a module
  - Return value to check the values managed by the component or the module

Test Stubs and Test Drivers



Module Name	Module Activity
Login()	Allows to login into the application
Order()	Performs order placing
Payment()	Allows payment with the use of the card, internet banking, UPI, cash, etc.
Report()	Generate the bill and receipt with the tracking number

Test Cases

- Test cases are designed by the following methods.
- Structure of the components
- Using requirement specifications in black box testing
- Based on the tester's knowledge and experience with the subject

Test Suite

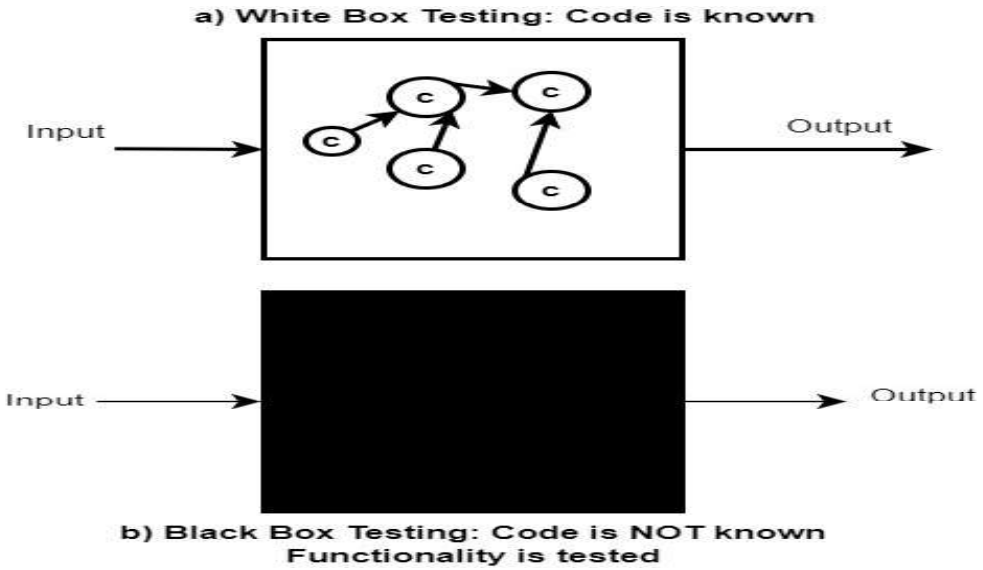
- The test suite consists of multiple test cases that help in test execution and reporting test status.
- The status can be either active, in progress, or completed.
- Test suites are based on functional or non-functional.
- Based on test cases, there are two testing approaches known as white box testing and black-box testing.

Test Case - Example

Test Case#	Test Case Description	Test Data	Expected Results	Actual Results	Pass/Fail
1	Check response after valid login and password are entered	Email: vaishali@geu.ac.in Password: *******	Successful Login	Login was successful	Pass

White Box Testing – Part 1

White Box and Black Box Testing



White Box Testing

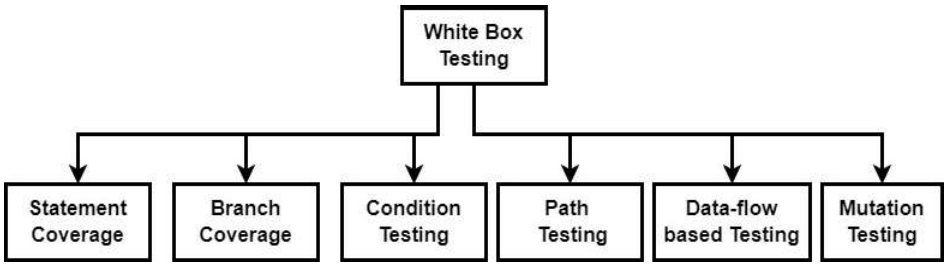
- White box testing is also known as clear box/glass box/code-based testing.
- The structure, coding, security holes, and the functionality of the developed software are tested.
- The tester understands the code, inner working of the application and creates test cases, finds the code coverage.

White Box Testing

- Poorly structured loops, data flow based on input and output, execution of objects, functions, testing of each statement, and expected output are tested in white box testing.
- It also checks for untested parts of code.
- It can be simple or complex, depending on the application.

White Box Testing

- This testing is performed by the software developer and not the test engineer.
- The structure, design, and coding of the developed software are tested.



Statement Coverage Testing

- The testing process aims at designing test cases so that every statement in a program is executed at least once.
- The main idea is to check whether there is a possibility of mistakes in computation, unused statements, missing statements, unused branches, wrong arithmetic calculations, etc.
- If some statement is not executed, then there is a bug in that statement.
- But the drawback is statement coverage may work properly for one set and not for others.

Statement Coverage Testing

Coverage measurement = $\frac{\text{Number of executed statements}}{\text{Total no of statements}}$

Test case a = 2
Number of statements executed = 5,
Total number of statements = 7
Statement coverage = 5/7 * 100 = 71%
Test case a = 3
Number of statements executed = 6
Total number of statements = 7
Statement coverage = 5/7 * 100 = 85%

```
1. check(int a){
2. rem = a % 2
3. if (rem==0)
4. printf("even number)
5. else
6. printf("odd number);
7. }
```

Branch Coverage Testing

- Branch testing is also known as edge testing where each conditional branching is tested for true and false values.
- It guarantees that all the test cases are fulfilled, and it is stronger than the statement coverage.
- Branch may be due to for, while loop, or if-else statements
- Designing test cases is easy in small programs but in large programs, coverage tools generate test cases.

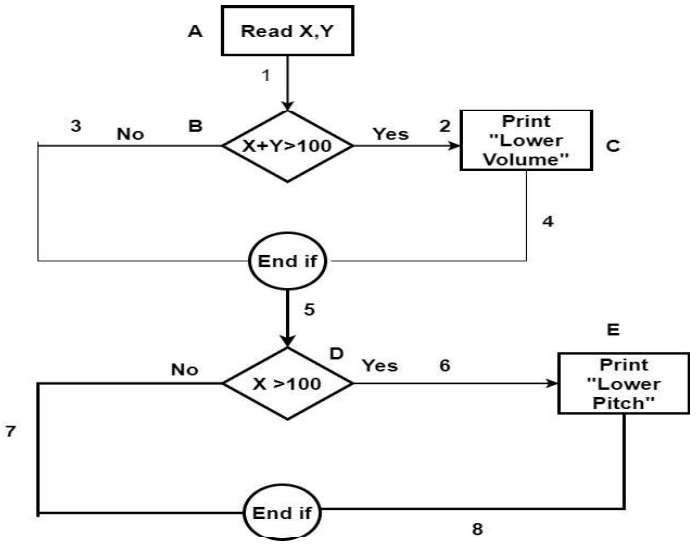
Branch Coverage Testing

- For example, in the program given in Table for testing all the statements we require inputs as: {1,2,3}, {1,3,2} {3,2,1}.

```
int main() {double a, b, c;
  printf("Enter three different numbers: ");
  scanf("%lf %lf %lf", &a, &b, &c);
  if (a >= b && a >= c)
  printf("%.2f is the largest number.", a);
  if (b >= a && b >= c)
   printf("%.2f is the largest number.", b);
  if (c >= a&& c >= b)
    printf("%.2f is the largest number.", c);
  return 0;}
```

Branch Coverage Testing using Control Path

- Read X, Y
- IF X+Y > 100 THEN
- Print “Lower Volume"
- ENDIF
- If X > 100 THEN
- Print “Reduce Pitch"
- ENDIF
- (1) A1-B2-C4-D6-E8 – For YES
- (2) A1-B3-5-D7- For No
- Hence Branch Coverage is 2.



White Box Testing – Part 2

Condition Testing

- When there is more than one condition or composite condition, with different branch statements, then a condition testing method is used.
- For example, consider a statement ((X OR B) NOT C).
- If there are n expressions in a composite conditional statement, then, 2^n test cases are required.
- The number of test cases increases exponentially in condition testing.

Data-flow based Testing

- The definitions and uses of different variables in a program can be used as test paths.
- Data flow-based testing methods make use of such paths.
- The issues such as undeclared variables, deleting the memory of a variable before its use, or multiple declarations of variables are pointed out in data flow testing.

Data-flow based Testing

- Consider a statement with the number M, then
- DEF(M) = {A/statement M contains a definition of A}, and
- USES(M) = {B/statement M contains a use of A}
- Consider the statement A = B + C.

Data-flow based Testing

- The DEF(M) = {A} and USES(M) = {B, C}.
- Here is the definition of A at statement M. Suppose there is a path from statement A to statement B and statement B does not contain a definition of A, then even statement B can use A as if it is live.
- This is known as a definition-use chain (DU chain).

Data-flow based Testing

- The DU chain can be given as follows:
- A is the variable, and X and Y are the statements.
- A ∈ DEF(X) and A ∈ USES(Y)
- This indicates that the definition of A is live in X and Y statements.
- In data flow testing, every DU chain is checked whether it is covered at least once.

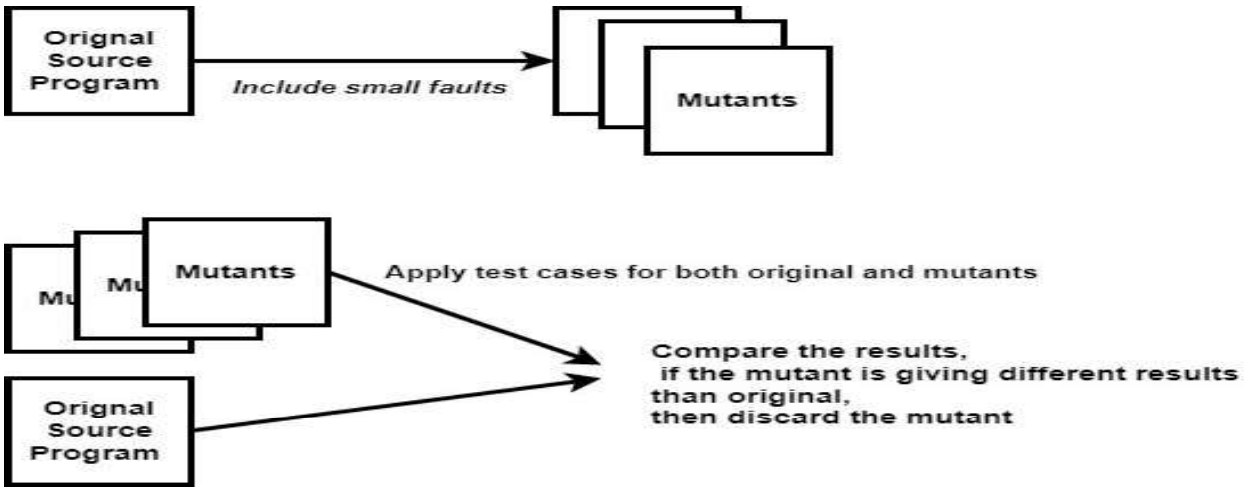
Mutation Testing

- In this testing, some statements of the source program are changed, also known as mutated to verify whether the test cases can locate errors.
- The changes in the program are very small and random.
- The changed program is called a mutant.
- The objective is to build the test cases robust so that they fail the mutated source program.

Mutation Testing

- This testing is used in unit testing to create a fault on purpose and perform white box testing.
- If the output of the mutant is different than the original, then it is killed.
- The different faults to be introduced may be the change of data type, altering the constant value, changing the order of the operators in an expression, etc.

Mutation Testing



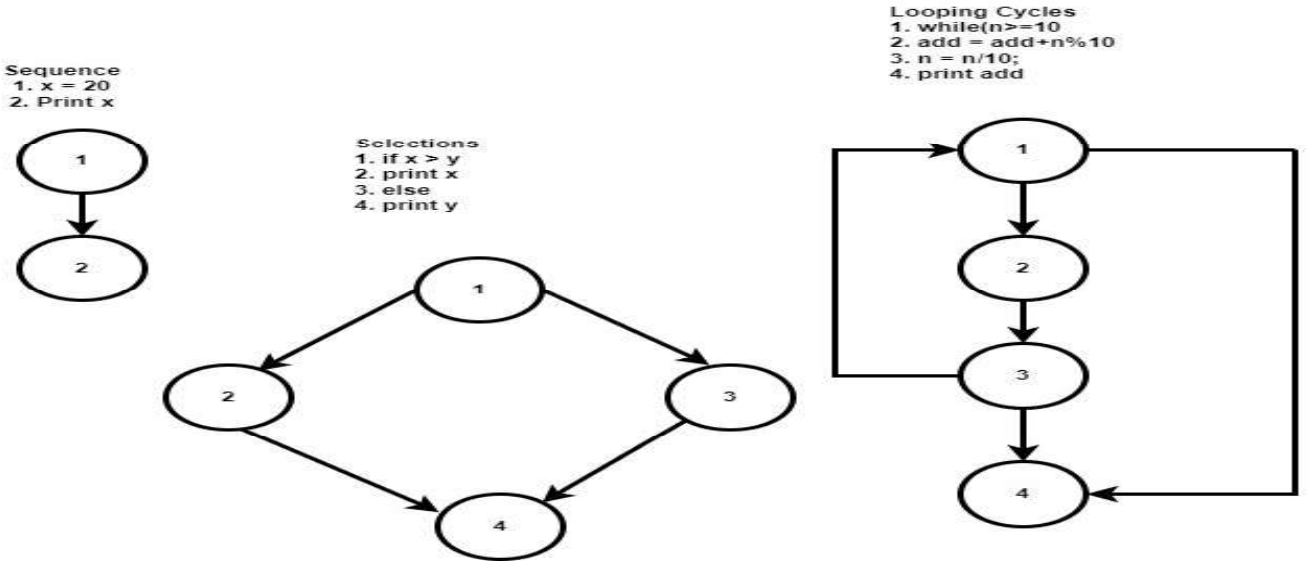
Unit 4 – Testing

Cyclomatic Complexity

Path Testing

- **Path testing** uses test cases so that the paths in a program are tested.
- The paths may be linearly independent.
- A **linearly independent path** can be defined using the **control flow graph (CFG)**.
- The **CFG** presents the sequence of execution of different program statements.

CFG



Cyclomatic Complexity

- The maximum number of linearly independent paths throughout the program can be identified using McCabe's cyclomatic complexity.
- It determines how many paths are to be tested. There are different methods to compute the cyclomatic complexity.
- Consider a CFG. The cyclomatic complexity C(G) can be computed as
- C(G) = E – V + 2
- where V is the number of nodes of the CFG, and E is the number of edges in the CFG

Cyclomatic Complexity

- The number of decision statements of a program can be used to determine the cyclomatic complexity of a program.
- Let us consider a program P with the number of decision statements of the program as N, then McCabe's metric C(G) = N+1.
- The details of each method are given by the example of computing the GCD of two numbers.

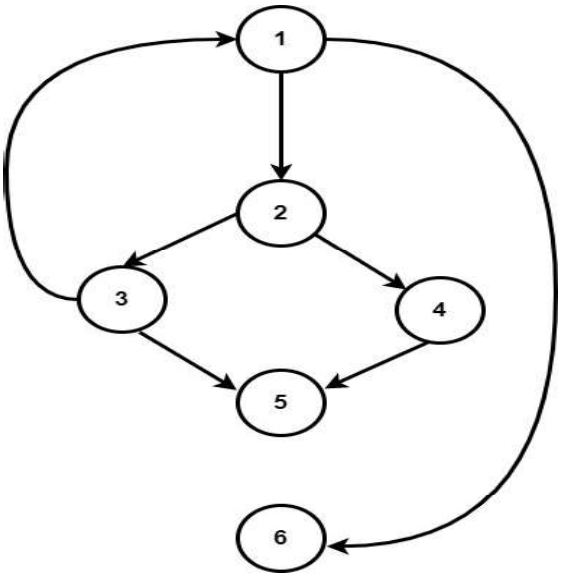
Cyclomatic Complexity

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

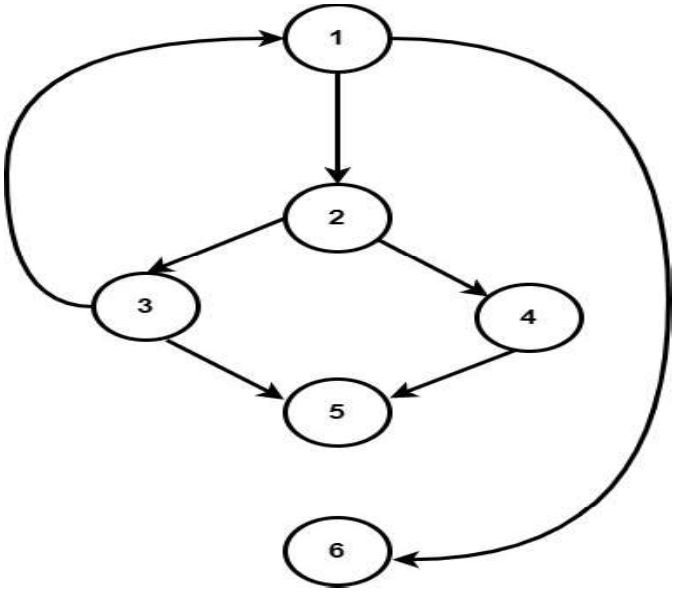
C(G) = N + 1

= 2 + 1

= 3



Cyclomatic Complexity



Method 1: C(G) = E – N + 2
= 7 – 6 + 2 =3
Method 2: C(G) = Total number of bounded areas + 1 = 2 + 1 =3
Method 3: C(G) = N + 1 = 2 + 1 = 3

Black Box Testing

Black Box Testing

- Black box testing involves testing functionalities of the software using requirement specifications.
- This testing is performed without the knowledge of design, or code.
- Black box testing is also known as behavioral testing.
- E.g., testing the working of websites such as gmail.com, makemytrip.com, irctc. in etc.

Black Box Testing - Steps

- Examine the requirement specifications of the system.
- Select a set of proper inputs and determine the possible output.
- Prepare test cases for the inputs and execute the test cases.
- Compare the results with expected outcomes.
- If there are any undesired results or errors, then correct the errors and repeat the testing.

Black Box Testing Strategies

- Equivalence Class Testing
- Boundary Value Testing
- Decision Table Testing

Equivalence Class Partitioning

- The equivalence class partitioning method is used to reduce the building of possible test cases.
- The set of input values is divided into different classes.
- The test cases are built in such a way that testing one class is like testing other classes.

Equivalence Class Partitioning

Age valid	Age Invalid	Age suitable for Voting	Age not suitable for voting
1 to 110	0 and <0	Age >=18	Age<18

A range of age groups, here the different inputs can be tested against the equivalent partitions

Boundary Value Analysis

- Here the software is tested for the input boundaries.
- The general practice is to test the value from the middle value in the input range.
- The programmers often miss the less than or greater than levels, so this technique is useful in such cases.

Boundary Value Analysis

- Example: In an online banking application, more than 10 fund transfers are not allowed.
- While testing this application, you may start with middle values such as 5, and 8 and test up to 10.
- When the value 10 is crossed the system should generate the exception error message.

Decision Tables

- In this method the inputs and test conditions are represented using a decision table.
- The test conditions are given TRUE (T) and FALSE (F).
- For software with large requirements management, these tools are effective.
- The different combinations of inputs and the conditions are captured in a tabular form along with the output.

Decision Tables- Example

T: Correct login ID/Password
F: Wrong login ID/Password
E: Error Message
D: Display Inbox

Condition	Rule 1	Rule 2	Rule 3	Rule 4
Login ID (T/F)	F	T	T	F
Password (T/F)	F	T	F	T
Output (E/D)	E	D	E	E

Static Testing

Static Testing Strategies

- Static testing strategies are used to check the system without executing a code.
- It is an economical process, and it is done at the early stages of software development.
- Software is verified either manually or using automated tools.
- It helps to reduce the development cost and time.

Static Testing Strategies

- The defects can be identified at the early stages, and it requires less effort in fixing the bugs.
- The missing requirements, inconsistencies in the interface design, and deviation from the target problems are identified.
- Static testing strategies are classified as:
- Review and Static Analysis

Review

- In the review process, the code is reviewed using software requirement specifications and other software design-related documents.
- The review is conducted by different people to identify any errors, duplicate or confusing statements if any.

Types of Review

- **Informal**: Informal meeting of the developer with his team.
- **Walk-through**: A subject expert is assigned to go through the code to avoid any errors in the future development stage.
- **Peer Review**: Team members test and verify each other's documents to identify and correct the defects.
- **Inspection**: The SRS is used to verify the code by the higher authorities

Static Analysis

Static analysis

- In the static analysis strategy, the quality of the developed code is evaluated using different standard tools.
- The defects such as syntax errors ignored in compilation, variables that are not serving any use, unused or unwanted code, infinite loops, and variables without initialization are identified in this method.

Static analysis

- Static analysis is done using three types..
- 1) Data flow for checking the processing of data streams at various stages.
- 2) Control flow of different statements during execution
- 3) Identification of linearly independent paths using cyclomatic complexity.

Compliance with Design and Coding Standards

- To pass through the code review, the coding standards must be maintained.
- As there is more than one member in the team, following a standard coding style gives a uniform appearance.
- Maintainability and readability of the code can be improved by using standard coding styles.
- In general, no coding standard is perfect for any application.

Compliance with Design and Coding Standards

- Structure of a program
- Naming rules and conventions
- Formatting
- Code Documentation
- Use of features in a programming language

Some Coding Standards

- **ISO:** International Organization for Standardization
- **BSI:** British Standards Institute
- **ANSI:** American National Standards Institute
- **IEEE:** Institute for Electronic and Electrical Engineers

Unit 4 – Testing

Automated Testing

Automated Testing

- Automated testing is used for the automatic review and validation of a software product using a test case suite.
- Manual testing can be slow.
- The quality of a software product is tested to check whether it fulfills the standard coding style, experience, and business of a user.

Automated Testing

- Automated testing is used for the automatic review and validation of a software product using a test case suite.
- The quality of a software product is tested to check whether it fulfills the standard coding style, experience, and business of a user.
- Manual testing can be time-consuming and may not cover all the test cases

Automated Testing

- Using automated testing a user need not be physically present in front of a computer.
- Automation can perform the tests, compare the expected and actual results, and generate detailed test reports.
- The drawback of automated testing is that it is not cost-effective in some cases.

Automated Testing

- Compare two images (e.g., photos, signature, captcha, etc.)
- Testing a database and web-based application for more than 1000000 users (e.g., irctc website in Indian Railway Reservation Online Ticket Booking)
- Testing a developed software on different operating system platforms concurrently.

Automated Testing Tools

- **LambdaTest: This tool is used for testing the web application for different web browsers, operating systems, and devices.**
- **BrowserStack: This testing tool is used to test the application on the web and mobile application**
- **TestProject: This tool provides a free test automation platform for web and mobile applications**
- **WorkSoft: This tool is used to give a platform for test automation, documentation support, and business process.**