MODULE-4

- **Blackbox testing:** Equivalence class testing, Boundary value testing, Decision table testing, Pairwise testing, State transition testing, Use-case testing;
- **White box testing:** Control flow testing, Data flow testing.

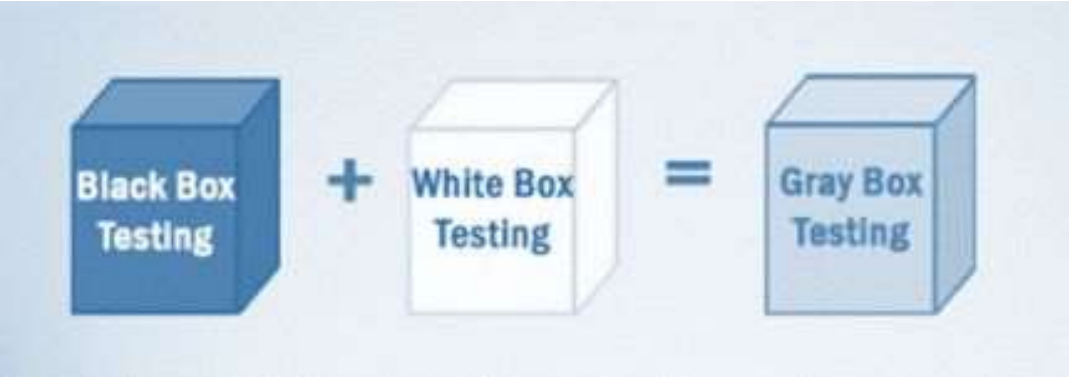
- **Black box testing** (Functional testing, Behavioral Testing, Data-driven testing, and Closed box testing) is a strategy in which testing is based solely on the requirements and specifications.

- Black box testing requires no knowledge of the internal paths, structure, or implementation of the software under test.

- **White box testing** (Structural testing, Clear box testing, Code-based testing, and Transparent testing and Glass box testing) is a strategy in which testing is based on the internal paths, structure, and implementation of the software under test.
- White box testing generally requires detailed programming skills.
- One of the basic goals of white-box testing is to verify a working flow for an application

- An additional type of testing is called **Gray box testing**.
- **Grey Box Testing** is also known as **translucent testing** as the **tester** has limited knowledge of coding.
- Gray Box Testing is a software testing method, which is a combination of both White Box Testing and Black Box Testing method.
- Grey Box Testing or Gray box testing is a software testing technique to test a software product or application with partial knowledge of internal structure of the application.
- The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

- **In White Box testing internal structure (code) is known**
- **In Black Box testing internal structure (code) is unknown**
- **In Grey Box Testing internal structure (code) is partially known**



Here are the generic steps followed to carry out any type of Black Box Testing.

- 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

Black-Box Testing Techniques

1) Equivalence class testing

2) Boundary value testing

3) Decision table testing

4) Pairwise testing

5) State transition testing

6) Use-case testing

Equivalence Class Testing

- **Equivalence Partitioning** or **Equivalence Class Partitioning** is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc.
- Equivalence class testing is a technique used to reduce the number of test cases (Test cases consist of inputs, outputs, and order of execution) to a manageable level while still maintaining reasonable test coverage.

- In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduces time required for testing because of small number of test cases
- It divides the input data of software into different equivalence data classes. All it requires are inputs or outputs that can be partitioned based on the system's requirements.
- An equivalence class consists of a set of data that is treated the same by the module or that should produce the same result.

- Any data value within a class is *equivalent,* in terms of testing, to any other value.
- Specifically, we would expect that:

1. If one test case in an equivalence class detects a defect, *all* other test cases in the same equivalence class are likely to detect the same defect.
2. If one test case in an equivalence class does not detect a defect, *no* other test cases in the same equivalence class is likely to detect the defect.

- **Testing-by-Contract** is based on the design-by-contract philosophy. Its approach is to create test cases only for the situations in which the pre-conditions are met.

- **Defensive Testing:** an approach that tests under both normal and abnormal pre-conditions.

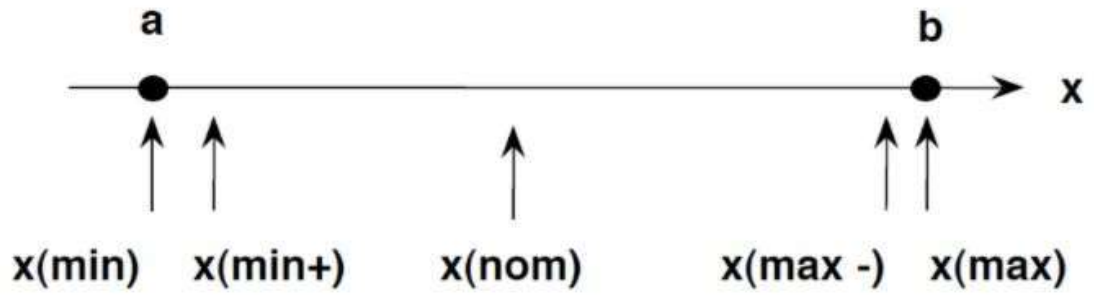
Boundary Value Testing

- **Boundary testing** is the process of testing between extreme ends or boundaries between partitions of the input values.
- Boundary value testing is a technique used to reduce the number of test cases to a manageable size while still maintaining reasonable coverage.
- So these extreme ends like Start - End, Lower - Upper, Maximum -Minimum, Just Inside - Just Outside values are called boundary values and the testing is called ‘boundary testing’.

- Boundary value testing focuses on the boundaries because that is where so many defects hide.
- Boundary Testing comes after the Equivalence Class Partitioning

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

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



Decision Table Testing

- A Decision Table is a tabular representation of inputs versus rules / cases / test conditions.
- It is a very effective tool used for both complex software testing and requirements management.
- Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily.
- Decision table testing is a software testing technique used to test system behavior for different input combinations.

- Decision Table testing can be used whenever the system must implement complex business rules when these rules can be represented as a combination of conditions and when these conditions have discrete actions associated with them.
- This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form
- That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage

Decision tables are an excellent tool to capture certain kinds of system requirements and to document internal system design.

Table 5-1: The general form of a decision table.

	Rule 1	Rule 2	...	Rule p
Conditions				
Condition-1				
Condition-2				
...				
Condition-m				
Actions				
Action-1				
Action-2				
...				
Action-n				

Conditions 1 through m represent various input conditions. Actions 1 through n are the actions that should be taken depending on the various combinations of input conditions. Each of the rules defines a unique combination of conditions that result in the execution ("firing") of the actions associated with that rule.

Pairwise Testing

- **Pairwise Testing** also known as **All-pairs testing** is a testing approach taken for testing the software using combinatorial method.
- When the number of combinations to test is very large, do not to attempt to test all combinations for all the values for all the variables, but test all pairs of variables. This significantly reduces the number of tests that must be created and run.
- It's a method to test all the possible discrete combinations of the parameters involved.

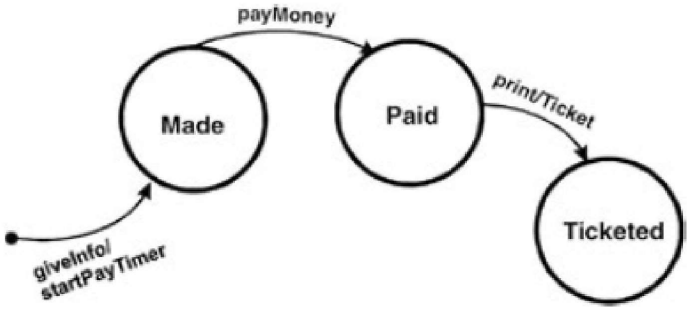
- Assume we have a piece of software to be tested which has got 10 input fields and 10 possible settings for each input field, then there are 10^{10} possible inputs to be tested.
- In this case, exhaustive testing in impossible even if we wish to test all combinations.
- Example: If a system had four different input parameters and each one could take on one of three different values, the number of combinations is 3^4 which is 81. It is possible to cover all the pairwise input combinations in only nine tests.

State Transition Testing

- **State Transition Testing** is a black box testing technique in which changes made in input conditions cause state changes or output changes in the Application under Test (AUT).
- State-Transition diagrams is an excellent tool to capture certain types of system requirements and to document internal system design.
- These diagrams document the events that come into and are processed by a system as well as the system's responses.

- When a system must remember something about what has happened before or when valid and invalid orders of operations exist, state-transition diagrams are excellent tools to record this information.
- State transition testing helps to analyze behaviour of an application for different input conditions.
- Testers can provide positive and negative input test values and record the system behaviour.
- State Transition Testing Technique is helpful where you need to test different system transitions.

- **State** (represented by a circle)-A state is a condition in which a system is waiting for one or more events. States "remember" inputs the system has received in the past and define how the system should respond to subsequent events when they occur. These events may cause state-transitions and/or initiate actions.
- **Transition** (represented by an arrow)-A transition represents a change from one state to another caused by an event.
- **Event** (represented by a label on a transition)-An event is something that causes the system to change state. Generally, it is an event in the outside world that enters the system through its interface.
- **Action** (represented by a command following a "/")-An action is an operation initiated because of a state change.
- **The entry point** on the diagram is shown by a black dot while the exit point is shown by a bulls-eye symbol.

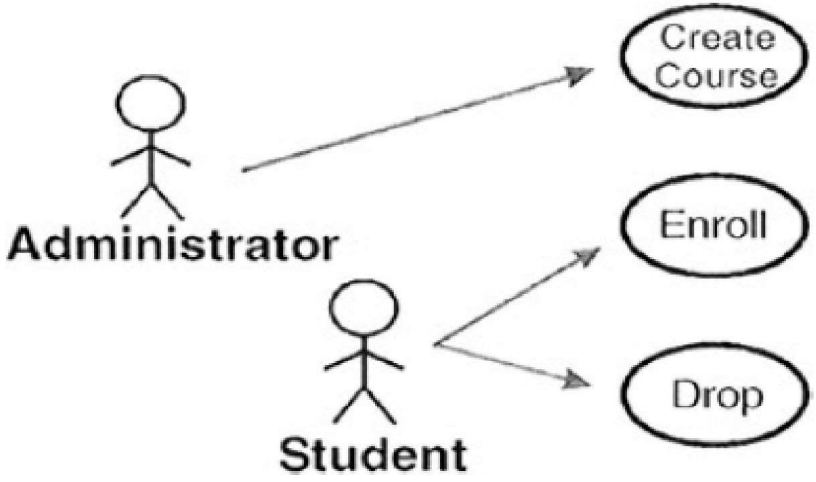


Use-Case Testing

- A **use case** is a scenario that describes the use of a system by an actor to accomplish a specific goal.
- A **scenario** is a sequence of steps that describe the interactions between the actor and the system.
- Use cases are made on the basis of user actions and the response of the software application to those user actions.
- The set of use cases makes up the functional requirements of a system.

- An **actor** is a user, playing a role with respect to the system, seeking to use the system to accomplish something worthwhile within a particular context.
- By "actor" we mean a user, playing a role with respect to the system, seeking to use the system to accomplish something worthwhile within a particular context.
- Actors are generally people although other systems may also be actors.
- The use case is defined from the perspective of the user, not the system.
- It is widely used in developing test cases at system or acceptance level

- Use Case Testing is a software testing technique that helps to identify test cases that cover entire system on a transaction by transaction basis from start to end.
- Test cases are the interactions between users and software application
- The Unified Modeling Language notion for use cases is:



The stick figures represent the actors, the ellipses represent the use cases, and the arrows show which actors initiate which use cases.

- **White box testing** is a strategy in which testing is based on the internal paths, structure, and implementation of the software under test (SUT).
- White box testing is more than code testing—it is **path** testing.
- Generally, the paths that are tested are within a module (unit testing).
- We can apply the same techniques to test paths between modules within subsystems, between subsystems within systems, and even between entire systems.

How do you perform White Box Testing?

Step 1) Understand the source code

Step 2) Create test cases and Execute

- White box testing has four distinct disadvantages.
  - First, the number of execution paths may be so large than they cannot all be tested. Attempting to test all execution paths through white box testing is generally as infeasible.
  - Second, the test cases chosen may not detect data sensitivity errors.
  - Third, white box testing assumes the control flow is correct (or very close to correct).
  - Fourth, the tester must have the programming skills to understand and evaluate the software under test.
- Advantages: When using white box testing, the tester can be sure that every path through the software under test has been identified and tested.

WhiteBox Testing Techniques

- Control flow testing
- Data flow testing

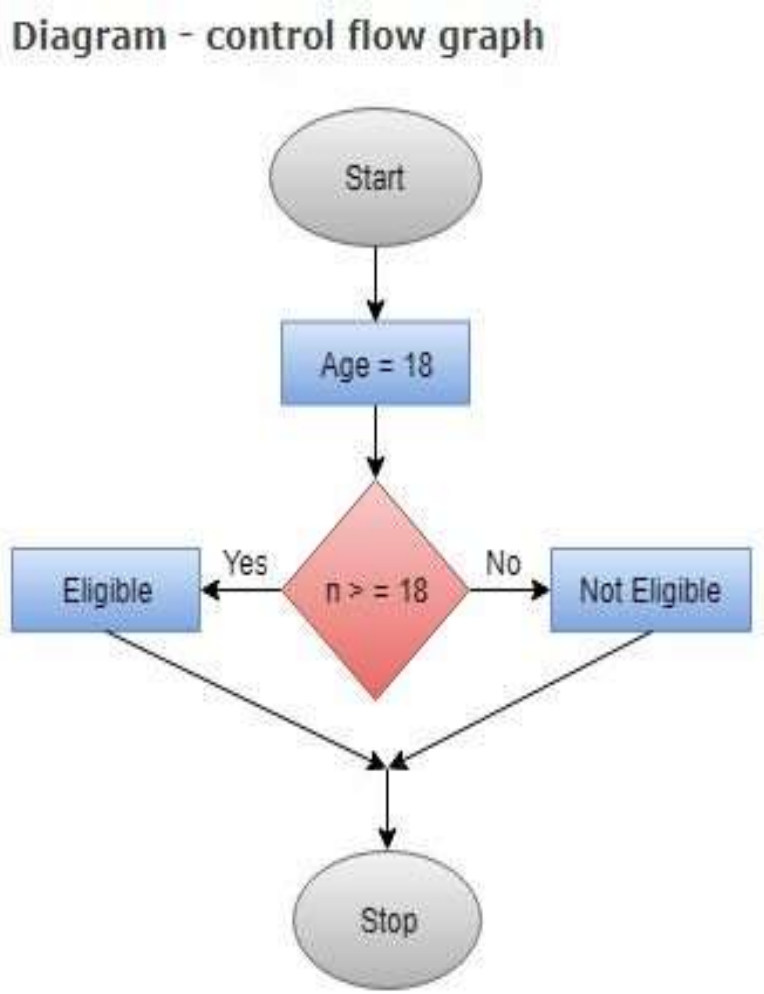
Control Flow Testing

- **Control flow testing** identifies the execution paths through a module of program code and then creates and executes test cases to cover those paths.
- **Path:** A sequence of statement execution that begins at an entry and ends at an exit.
- The aim of Control flow testing 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

- Test cases represented by the control graph of the program.
- Control flow graphs are the foundation of control flow testing. These graphs document the module's control structure. Modules of code are converted to graphs, the paths through the graphs are analyzed, and test cases are created from that analysis.
- **Control Flow Graph** is formed from the node, edge, decision node, junction node to specify all possible execution path.

Notations used for Control Flow Graph

1) **Node** : Used to create a path of procedures. Basically, it represents the sequence of procedures
2) **Edge** : Used to link the direction of nodes
3) **Decision Node** : Used to decide next node of procedure as per the value
4) **Junction node** : Point where at least three links meet



- Control flow testing is the cornerstone of unit testing.
- It should be used for all modules of code that cannot be tested sufficiently through reviews and inspections.
- Its limitations are that the tester must have sufficient programming skill to understand the code and its control flow.
- In addition, control flow testing can be very time consuming because of all the modules and basis paths that comprise a system.

Data Flow Testing

- **Data flow testing** is a powerful tool to detect improper use of data values due to coding errors.
- Data Flow Testing is a specific strategy of software testing that focuses on data variables and their values
- Variables that contain data values have a defined life cycle. They are created, they are used, and they are killed (destroyed).

- A **data flow graph** is similar to a control flow graph in that it shows the processing flow through a module.
- We will construct these diagrams and verify that the define-use-kill patterns are appropriate.
    - First, we will perform a static test of the diagram. By "static" we mean we examine the diagram (formally through inspections or informally through look-sees).
    - Second, we perform dynamic tests on the module. By "dynamic" we mean we construct and execute test cases.

- It keeps a check at the data receiving points by the variables and its usage points.
- The process is conducted to detect the bugs because of the incorrect usage of data variables or data values.
- A common programming mistake is referencing the value of a variable without first assigning a value to it.
- Data flow testing builds on and expands control flow testing techniques.
- As with control flow testing, it should be used for all modules of code that cannot be tested sufficiently through reviews and inspections.

- Its limitations are that the tester must have sufficient programming skill to understand the code, its control flow, and its variables.
- Like control flow testing, data flow testing can be very time consuming because of all the modules, paths, and variables that comprise a system.

❖ Comparison of Black Box and White Box Testing:

Which to choose ???

Black Box Testing	White Box Testing
the main focus of black box testing is on the validation of your functional requirements.	White Box Testing (Unit Testing) validates internal structure and working of your software code
Black box testing gives abstraction from code and focuses on testing effort on the software system behavior.	To conduct White Box Testing, knowledge of underlying programming language is essential. Current day software systems use a variety of programming languages and technologies and its not possible to know all of them.
Black box testing facilitates testing communication amongst modules	White box testing does not facilitate testing communication amongst modules