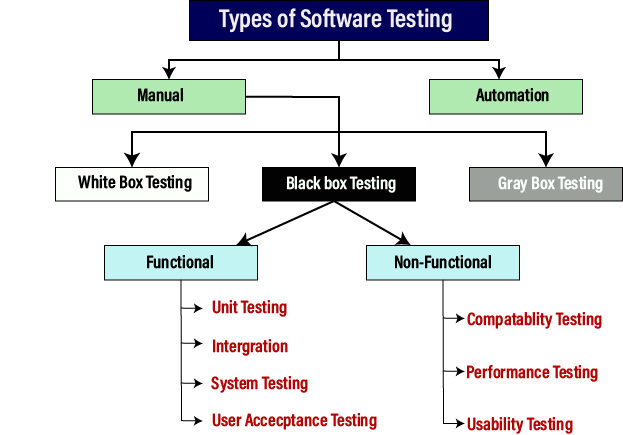
**UNIT IV(Testing Strategies)**

 Software Testing

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.



Manual testing

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Automation testing

Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

**A strategic approach to software testing**

A **strategic approach to software testing** involves a well-planned and structured methodology to ensure effective testing and quality assurance throughout the software development lifecycle. Let’s delve into some key aspects:

1. **Testing Strategies**:
   * **Black Box Testing**: This strategy evaluates the functionality of the software without examining the internal code structure.
   * **White Box Testing**: It focuses on testing the internal code structure and logic of the software.
   * **Unit Testing**: Tests individual units or components to ensure they function as intended.
   * **Integration Testing**: Verifies that different components of the software work together seamlessly.
   * **Functional Testing**: Ensures that the functional requirements of the software are met.
   * **System Testing**: Tests the complete software system against specified requirements.
   * **Acceptance Testing**: Validates whether the software meets customer or end-user expectations.
   * **Regression Testing**: Verifies that changes or modifications haven’t introduced new defects.
   * **Performance Testing**: Evaluates speed, scalability, and stability.
   * **Security Testing**: Identifies vulnerabilities and ensures compliance with security requirements.
2. **Objectives of Software Testing**:
   * **Error Detection**: Uncover defects and errors in the software.
   * **Effective Test Cases**: Design test cases that systematically find different types of errors.
   * **Unknown Error Discovery**: Successful test cases reveal previously undiscovered issues.
   * **Efficiency**: Design tests to minimize time and effort while achieving thorough coverage

**TEST STRATEGIES FOR CONVENTIONAL SOFTWARE**

There are many strategies that can be used to test software.

* 1 At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors.
* This approach simply does not work. It will result in buggy software.
* At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed.
* This approach, although less appealing to many, can be very effective.
* A testing strategy that is chosen by most software teams falls between the two extremes.
* It takes an incremental view of testing,
* Beginning with the testing of individual program units,
* Moving to tests designed to facilitate the integration of the units,Culminating with tests that exercise the constructed system.

**White box testing**

* **White box testing**  techniques analyze internal design, code structure, and the working of the complete software.
* It is also called glass box testing or clear box testing or structural testing.
* White Box Testing is also known as transparent testing or open box testing.
* It is mostly done by software developers.
* t is used to test the software’s internal logic, flow, and structure.

**Testing Techniques**

**1. Statement Coverage**

In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.

**2. Branch Coverage:**

In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

**3. Condition Coverage**

In this technique, all individual conditions must be covered. Here we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and**else** condition.

**5.Path Testing**

In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

**6. Loop Testing**

Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

**Tools required for White box testing:**

* PyUnit
* Sqlmap
* Nmap
* Parasoft Jtest
* Nunit

**Features of White box Testing**

1. **Code coverage analysis:** White box testing helps to analyze the code coverage of an application, which helps to identify the areas of the code that are not being tested.
2. **Access to the source code:** White box testing requires access to the application’s source code, which makes it possible to test individual functions, methods, and modules.
3. **Knowledge of programming languages:** Testers performing white box testing must have knowledge of programming languages like Java, C++, Python to understand the code structure and write tests.
4. **Identifying logical errors:** White box testing helps to identify logical errors in the code, such as infinite loops or incorrect conditional statements.
5. **Integration testing:**White box testing is useful for integration testing, as it allows testers to verify that the different components of an application are working together as expected.
6. **Unit testing:** White box testing is also used for unit testing, which involves testing individual units of code to ensure that they are working correctly.

Advantages of White box testing

* White box testing optimizes code so hidden errors can be identified.
* Test cases of white box testing can be easily automated.
* This testing is more thorough than other testing approaches as it covers all code paths.
* It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

* White box testing is too much time consuming when it comes to large-scale programming applications.
* White box testing is much expensive and complex.
* It can lead to production error because it is not detailed by the developers.
* White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

# Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Black box testing

# Black box testing techniques

# Decision table technique

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 behaviour 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. To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

e.g decision table for email account login.

# Cause and Effect Graph 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. 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.

# 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.

#### Equivalence partitioning

Here ,input values that provide to system are divided into different classes or group based on its similarity in the outcome.Instead of using each and every input value, use any one value from the group to test outcome.

**Identifying and partitioning into equivalence classes:** The input data is partitioned into a minimum of two sets: one set contains valid input values, and the other contains invalid input values. For example, for an age box that can contain ages between 20-40, valid input values can be 21, 25, 30, 39, etc., while invalid input values can be any value less than 20 or greater than 40 (e.g., 10, 15, 45, 55, etc.).

#### Boundary Value analysis:

This strategy tests boundary values to evaluate whether a specific range of values would be acceptable to the system. It test, while entering boundary value, use any one value from the group test outcome

**Advantages of Black Box Testing**

* The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
* It is efficient for implementing the tests in the larger system.
* Tests are executed from the user’s or client’s point of view.
* Test cases are easily reproducible.
* It is used to find the ambiguity and contradictions in the functional specifications.

**Disadvantages of Black Box Testing**

There is a possibility of repeating the same tests while implementing the testing process.

* Without clear functional specifications, test cases are difficult to implement.
* It is difficult to execute the test cases because of complex inputs at different stages of testing.
* Sometimes, the reason for the test failure cannot be detected.
* It does not reveal the errors in the control structure.
* Working with a large sample space of inputs consumes a lot of time.

**Difference between Black Box Testing and**[**White Box Testing**](https://www.testbytes.net/blog/white-box-testing/)

|  |  |
| --- | --- |
| Used to test software without knowing the internal structure of the software | Performed after knowing the internal structure of the software |
| Carried out by testers | Performed by developers |
| Does not require programming knowledge | Requires programming knowledge |
| Requires implementation knowledge | Does not require implementation knowledge |
| Higher level testing | Lower level testing |
| Consumes less time | Consumes a lot of time |
| Done in the trial and error method | Data domains and boundaries can be tested |
| **Types of black box testing** 1. Functional testing 2. Regression testing 3. Non-functional testing | **Types of white box testing** 1. Path testing 2. Loop testing 3. Condition testing |
| Not suitable for algorithm testing | Suitable for algorithm testing |

Debugging : The process of fixing and resolving the defects is known as debugging.

Defects/bug is an error in an application that is created during building and designing software due to which software start to show abnormal behaviour during its use .

**Software metrics**  is a standard of measurement that contains many activities which involve some degree of measurement.

Software matrics play a crucial role in software engineering by providing quantifiable measures of various aspects of software systems and development processes. Let’s dive into the details:

Software metrics

Product Metrics

Process Metrics

Project Metrics

1. **Product Metrics**:
   * These metrics focus on measuring characteristics of the software product itself. They help assess the quality, reliability, and complexity of the software. Here are some important product metrics:
     + **Size Metrics**:
       - Measure the size of the software code or components. Common examples include **Lines of Code (LOC)** and **Function Points (FP)**.
     + **Complexity Metrics**:
       - Evaluate the intricacy of the software design or implementation. Metrics like **Cyclomatic Complexity** and **Halstead Complexity** fall into this category.
     + **Quality Metrics**:
       - Assess the overall quality of the software. Examples include **defect density**, **reliability**, and **maintainability**.
     + **Performance Metrics**:
       - Measure how efficiently the software performs its intended functions. Response time, throughput, and resource utilization are relevant here.
2. **Process Metrics**:
   * These metrics focus on the software development process itself. They help optimize development and maintenance activities. Examples of process metrics include:
     + **Efficiency Metrics**:
       - Evaluate the efficiency of fault detection, debugging, and testing processes.
     + **Productivity Metrics**:
       - Measure the productivity of development teams. For instance, lines of code produced per developer per day.
     + **Defect Metrics**:
       - Track the number of defects found during development, testing, and post-release phases.
3. **Project Metrics**:
   * Project managers use these metrics to monitor project progress. They rely on historical data from past projects to estimate effort, cost, and time for new software development. Key project metrics include:
     + **Effort Metrics**:
       - Estimate the effort required for development tasks.
     + **Cost Metrics**:
       - Assess the financial resources needed for the project.
     + **Time Metrics**:
       - Track project timelines and deadlines.

**Advantages of Software Metrics**:

* **Comparative Study**: Metrics allow us to compare different design methodologies and programming languages.
* **Quality Specifications**: Metrics aid in preparing software quality specifications.
* **Compliance Verification**: Metrics help verify compliance with software requirements.
* **Effort Estimation**: Metrics provide insights into the effort needed for design and development.
* **Complexity Assessment**: Metrics reveal the complexity of the code.

1. **Product Metrics**:
   * These metrics focus on measuring characteristics of the software product itself. They help assess the quality, reliability, and complexity of the software. Here are some important product metrics:
     + **Size Metrics**:
       - Measure the size of the software code or components. Common examples include **Lines of Code (LOC)** and **Function Points (FP)**.
     + **Complexity Metrics**:
       - Evaluate the intricacy of the software design or implementation. Metrics like **Cyclomatic Complexity** and **Halstead Complexity** fall into this category.
     + **Quality Metrics**:
       - Assess the overall quality of the software. Examples include **defect density**, **reliability**, and **maintainability**.
     + **Performance Metrics**:
       - Measure how efficiently the software performs its intended functions. Response time, throughput, and resource utilization are relevant here.
2. **Process Metrics**:
   * These metrics focus on the software development process itself. They help optimize development and maintenance activities. Examples of process metrics include:
     + **Efficiency Metrics**:
       - Evaluate the efficiency of fault detection, debugging, and testing processes.
     + **Productivity Metrics**:
       - Measure the productivity of development teams. For instance, lines of code produced per developer per day.
     + **Defect Metrics**:
       - Track the number of defects found during development, testing, and post-release phases.
3. **Project Metrics**:
   * Project managers use these metrics to monitor project progress. They rely on historical data from past projects to estimate effort, cost, and time for new software development. Key project metrics include:
     + **Effort Metrics**:
       - Estimate the effort required for development tasks.
     + **Cost Metrics**:
       - Assess the financial resources needed for the project.
     + **Time Metrics**:
       - Track project timelines and deadlines.

**Advantages of Software Metrics**:

* **Comparative Study**: Metrics allow us to compare different design methodologies and programming languages.
* **Quality Specifications**: Metrics aid in preparing software quality specifications.
* **Compliance Verification**: Metrics help verify compliance with software requirements.
* **Effort Estimation**: Metrics provide insights into the effort needed for design and development.
* **Complexity Assessment**: Metrics reveal the complexity of the code.