

Chap 02 Testing Techniques

***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.

Dynamic Testing:

Dynamic testing is a type of software testing that involves executing the software and evaluating its behavior during runtime. It is also known as functional testing, as it focuses on testing the software's functionality and how it behaves under different inputs and conditions.

During dynamic testing, the software is run and tested against a set of predefined test cases. These test cases are designed to cover a range of inputs and use cases, and they are used to check the software's behavior and output in response to different inputs. This can include testing the software's user interface, functional requirements, and performance.

Dynamic Testing is a type of [Software Testing](#) which is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed. Dynamic Testing is basically performed to describe the dynamic behavior of code.

We all know that Testing is verification and validation, and it takes 2 Vs to make testing complete. Out of the 2 Vs, Verification is called a Static testing and the other "V", Validation is known as Dynamic testing.

Dynamic Testing Example

Let's understand How to do Dynamic Testing with an example:

Suppose we are testing a Login Page where we have two fields say "Username" and "Password" and the Username is restricted to Alphanumeric.

When the user enters Username as "Guru99", the system accepts the same. Whereas when the user enters as Guru99@123 then the application throws an error message. This result shows that the code is acting dynamically **based on the user input**.

Dynamic testing is when you are working with the actual system by providing an input and comparing the actual behavior of the application to the expected behavior. In other words, working with the system with the intent of finding errors.

Types of Dynamic Testing

Dynamic Testing is classified into two categories

- White Box Testing
- Black Box Testing

Advantages of Dynamic Testing

- It discloses very difficult and complex defects.
- It detects the defects that can't be detected by static testing.
- It is easy to implement and does not require any special tools or expertise.
- It can be used to test the software with different input values.
- It can be used to test the software with different data sets.
- It can be used to test the software with different user profiles.
- It can be used to test the functionality of the code.
- It can be used to test the performance of the code.
- It can be used to test the security of the code.

Disadvantages of Dynamic Testing

- It is a time consuming process as in dynamic testing whole code is executed.
- It increases the budget of the software as dynamic testing is costly.
- Dynamic testing may require more resources than static testing.
- Dynamic testing may be less effective than static testing in some cases.
- It is difficult to cover all the test scenarios.

◆ White Box Testing:

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing.

- 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.
- White box testing is a software testing technique that involves testing the internal structure and workings of a software application.
- The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.
- White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure.

Generic steps of white box testing:

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
- In this step testing of internal subroutines takes place. Internal subroutines such as non-public methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security

The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing
- Testing based on the memory perspective
- Test performance of the program

◆ Techniques Used in White Box Testing

Logic Coverage Criteria:

) Logic coverage criteria

- White-box testing provides the degree to which tests cover the logic of the software program. Exhaustive white box testing implies that each path along the program is executed; but rigorous testing is unachievable goal for a program with loops.

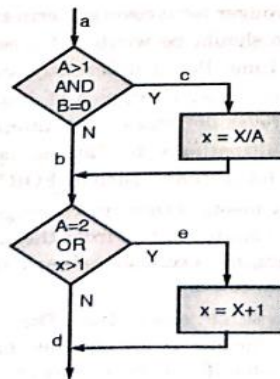


Fig. P. 2.3.2 : Small program to be checked

- If you refuse to conduct path testing, it may seem that it would be worth executing each statement through the program at least once. Unfortunately though, this is a weak criterion for a good back-box test. Suppose that the Figure above shows a small program to be tested. An equivalent program written in a procedural language, PL / 1, follows :

```

M: PROCEDURE (A, B, X);
IF ((A > 1) & (B = 0)) THEN DO;
X = X/A;
END;
IF ((A = 2) | (X > 1)) THEN DO;
X = X + 1;
END; END;
  
```

- You could execute each statement by creating a single test, which traverses the path *ace*. In other words, if you set $A = 2$, $B = 0$, and $X = 3$ at point *a*, each statement would be executed once (in fact, X can take any integer value > 1).
- Unfortunately, this criterion is worse than you may think. For instance, let the first decision be an *or*, not an *and*. If that is the case, this error will not be found. Let the second decision in the program be $X > 0$; if so, this error will not be found. In addition, there is a path in which X does not change (path *abd*). If there is an error, then it will not be detected. Thus, the statement coverage criterion is a poor one that it is usually not used.
- Using services of game testing company you become able to play video games free of charge! All you need you do is to control their quality by finding defects and bugs in them.

- Basis 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.

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.

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

- **Steps:**

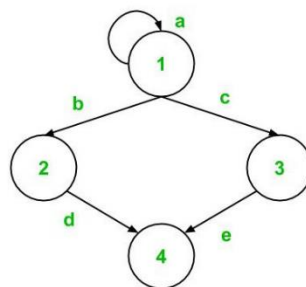
- i. Make the corresponding control flow graph
- ii. Calculate the cyclomatic complexity
- iii. Find the independent paths
- iv. Design test cases corresponding to each independent path
- v. $V(G) = P + 1$, where P is the number of predicate nodes in the flow graph
- vi. $V(G) = E - N + 2$, where E is the number of edges and N is the total number of nodes
- vii. $V(G)$ = Number of non-overlapping regions in the graph
- viii. #P1: 1 – 2 – 4 – 7 – 8
- ix. #P2: 1 – 2 – 3 – 5 – 7 – 8
- x. #P3: 1 – 2 – 3 – 6 – 7 – 8
- xi. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

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

What is a Graph Matrix:

A graph matrix is a square matrix whose size represents the number of nodes in the control flow graph. If you do not know what control flow graphs are, then read this article. Each row and column in the matrix identifies a node and the entries in the matrix represent the edges or links between these nodes. Conventionally, nodes are denoted by digits and edges are denoted by letters.



Let's convert this control flow graph into a graph matrix. Since the graph has 4 nodes, so the graph matrix would have a dimension of 4 X 4. Matrix entries will be filled as follows :

- (1, 1) will be filled with 'a' as an edge exists from node 1 to node 1
- (1, 2) will be filled with 'b' as an edge exists from node 1 to node 2. It is important to note that (2, 1) will not be filled as the edge is unidirectional and not bidirectional
- (1, 3) will be filled with 'c' as edge c exists from node 1 to node 3
- (2, 4) will be filled with 'd' as edge exists from node 2 to node 4
- (3, 4) will be filled with 'e' as an edge exists from node 3 to node 4

The graph matrix formed is shown below :

	1	2	3	4
1	a	b	c	
2				d
3				e
4				

Connection Matrix:

A connection matrix is a matrix defined with edges weight. In simple form, when a connection exists between two nodes of control flow graph, then the edge weight is 1, otherwise, it is 0. However, 0 is not usually entered in the matrix cells to reduce the complexity.

For example, if we represent the above control flow graph as a connection matrix, then the result would be :

	1	2	3	4
1	1	1	1	
2				1
3				1
4				

As we can see, the weight of the edges are simply replaced by 1 and the cells which were empty before are left as it is, i.e., representing 0.

A connection matrix is used to find the **cyclomatic complexity of the control graph**.

Although there are three other methods to find the cyclomatic complexity but this method works well too.

Following are the **steps to compute the cyclomatic complexity** :

- xii. Count the number of 1s in each row and write it in the end of the row
- xiii. Subtract 1 from this count for each row (Ignore the row if its count is 0)
- xiv. Add the count of each row calculated previously
- xv. Add 1 to this total count
- xvi. The final sum in Step 4 is the cyclomatic complexity of the control flow graph

Let's apply these steps to the graph above to compute the cyclomatic complexity.

	1	2	3	4	
1	1	1	1		3 - 1 = 2
2				1	1 - 1 = 0
3				1	1 - 1 = 0
4					Ignore
Cyclomatic complexity = 2 + 0 + 0 + 1 = 3					

We can verify this value for cyclomatic complexity using other methods :

Method-1 :

Cyclomatic complexity

$$= e - n + 2 * P$$

Since here,

$$e = 5$$

$$n = 4$$

$$\text{and, } P = 1$$

Therefore, cyclomatic complexity,

$$= 5 - 4 + 2 * 1$$

$$= 3$$

▲Data Flow 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.
- Data flow testing is a group of testing strategies to examine the control flow of programs in order to explore the sequence of variables according to the sequence of events.
- It mainly focuses on the points at which values assigned to the variables and the point at which these values are used by concentrating on both points, data flow can be tested.

Let us understand this with the help of an example

1. read x;	
2. If(x>0)	(1, (2, t), x), (1, (2, f), x)
3. a= x-F1	(1, 3, x)
4. if(c<=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).

x= 1
Path - 1, 2, 3, 8
Output = 2

If we consider x = 1, in step 1; x is allocated a value of 1 then we move to step 2 (since x > 0 we will move to statement 3 (a = x+1) and at end, it will go to statement 8 and print x = 2.

For the second path, we assign x as 1
Set x = - 1
Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8
Output = 2

Output = 2

- x is set as 1 then it goes to step 1 to allocate x as 1 and then moves to step 2 which is false as x is smaller than 0 ($x > 0$ and here $x = -1$). It will then move to step 3 and then jump to step 4; as 4 is true ($x \leq 0$ and their x is less than 0) it will jump to step 5 ($x < 1$) which is true and it will move to step 6 ($x = x + 1$) and here x is improved by 1.
So,
 $x = -1 + 1$
 $x = 0$
- x become 0 and it goes to step 5 ($x < 1$), as it is true it will jump to step 6 ($x = x + 1$)
 $x = x + 1$
 $x = 0 + 1$
 $x = 1$
- x is now 1 and jump to step 5 ($x < 1$) and now the condition is false and it will jump to step 7 ($a = x + 1$) and set a = 2 as x is 1. At the end the value of a is 2. And on step 8 we get the output as 2.

☀️ Advantages of Data Flow Testing:

Data Flow Testing is used to find the following issues-

- To find a variable that is used but never defined,
- To find a variable that is defined but never used,
- To find a variable that is defined multiple times before it is use,
- Deallocating a variable before it is used.

Disadvantages of Data Flow Testing

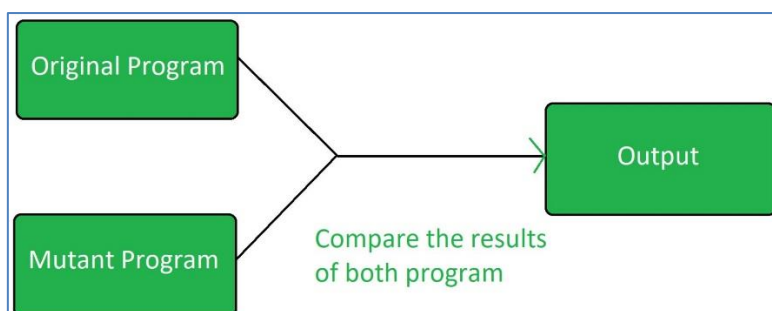
- Time consuming and costly process
- Requires knowledge of programming languages

☑️ Mutation testing:

- Mutation Testing is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to find errors in source code.
- The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source code.
- The changes made in the mutant program should be kept extremely small that it does not affect the overall objective of the program.
- Mutation Testing is also called Fault-based testing strategy as it involves creating a fault in the program and it is a type of White Box Testing which is mainly used for Unit Testing.

✓ History of Mutation Testing:

Richard Lipton proposed the mutation testing in 1971 for the first time. Although high cost reduced the use of mutation testing but now it is widely used for languages such as Java and XML.



Mutation testing can be applied to design models, specifications, databases, tests, and XML. It is a structural testing technique, which uses the structure of the code to guide the testing process. It can be described as the process of rewriting the source code in small ways in order to remove the redundancies in the source code.

#Objective of Mutation Testing:

The objective of mutation testing is:

- **To identify pieces of code that are not tested properly.**
- **To identify hidden defects that can't be detected using other testing methods.**
- **To discover new kinds of errors or bugs.**
- **To calculate the mutation score.**
- **To study error propagation and state infection in the program.**
- **To assess the quality of the test cases.**



Types of Mutation Testing:

Mutation testing is basically of 3 types:

1. Value Mutations:

In this type of testing the values are changed to detect errors in the program. Basically a small value is changed to a larger value or a larger value is changed to a smaller value. In this testing basically constants are changed.

Example:

- **Initial Code:**
- `int mod = 1000000007;`
- `int a = 12345678;`
- `int b = 98765432;`
- `int c = (a + b) % mod;`
- **Changed Code:**
- `int mod = 1007;`
- `int a = 12345678;`
- `int b = 98765432;`
- `int c = (a + b) % mod;`

2. Decision Mutations:

In decisions mutations are logical or arithmetic operators are changed to detect errors in the program.

Example:

Initial Code:

- `if(a < b)`
- `c = 10;`
- `else`
- `c = 20;`
- **Changed Code:**
- `if(a > b)`
- `c = 10;`
- `else`
- `c = 20;`

3. Statement Mutations:

In statement mutations a statement is deleted or it is replaced by some other statement.

Example:

- **Initial Code:**
- `if(a < b)`
- `c = 10;`
- `else`
- `c = 20;`
- **Changed Code:**
- `if(a < b)`
- `d = 10;`
- `else`
- `d = 20;`

Tools used for Mutation Testing :

- Judy
- Jester
- Jumble
- PIT
- MuClipse.

Advantages of Mutation Testing:

- It brings a good level of error detection in the program.
- It discovers ambiguities in the source code.
- It finds and solves the issues of loopholes in the program.
- It helps the testers to write or automate the better test cases.
- It provides more efficient programming source code.

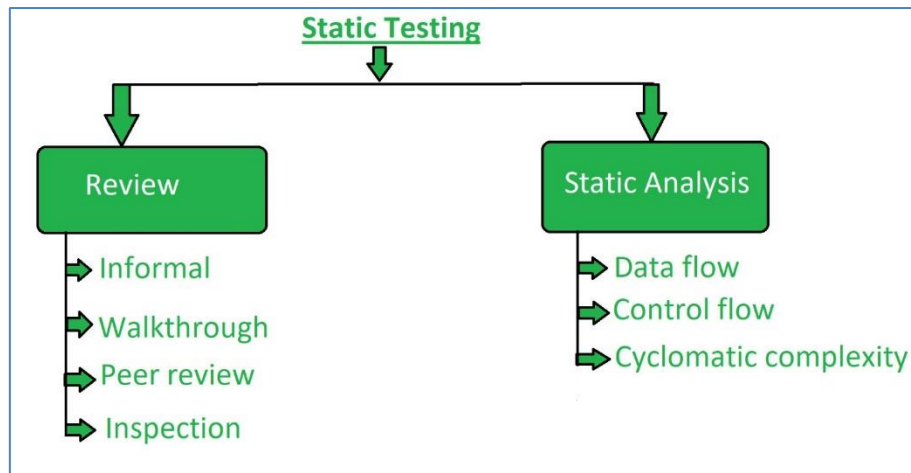
Disadvantages of Mutation Testing:

- It is highly costly and time-consuming.
- It is not able for Black Box 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.
- Whereas in Dynamic Testing checks, the code is executed to detect the defects.
- 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.
- The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing.

Static Testing Techniques: There are mainly two type techniques used in Static Testing:



1. **Review:** In static testing review is a process or technique that is performed to find the potential defects in the design of the software. It is process to detect and remove errors and defects in the different supporting documents like software requirements specifications. People examine the documents and sorted out errors, redundancies and ambiguities. Review is of four types:

- Informal: In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.
- Walkthrough: It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.
- Peer review: Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.
- Inspection: Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

2. **Static Analysis:** Static Analysis includes the evaluation of the code quality that is written by developers. Different tools are used to do the analysis of the code and comparison of the same with the standard. It also helps in following identification of following defects:

- (a) Unused variables
- (b) Dead code
- (c) Infinite loops
- (d) Variable with undefined value
- (e) Wrong syntax

Static Analysis is of three types:

- Data Flow: Data flow is related to the stream processing.
- Control Flow: Control flow is basically how the statements or instructions are executed.
- Cyclomatic Complexity: Cyclomatic complexity defines the number of independent paths in the control flow graph made from the code or flowchart so that minimum number of test cases can be designed for each independent path.

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.

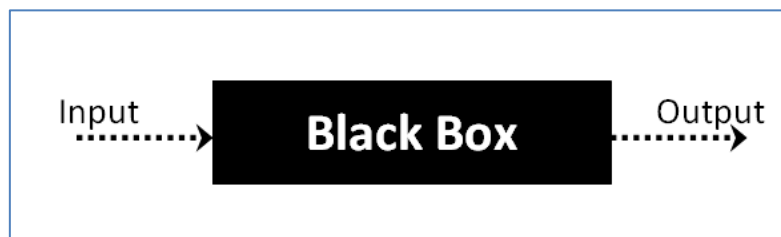
Disadvantages of White box testing

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.

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.

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.



The above Black-Box can be any software system you want to test. 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.

Generic steps of black box testing

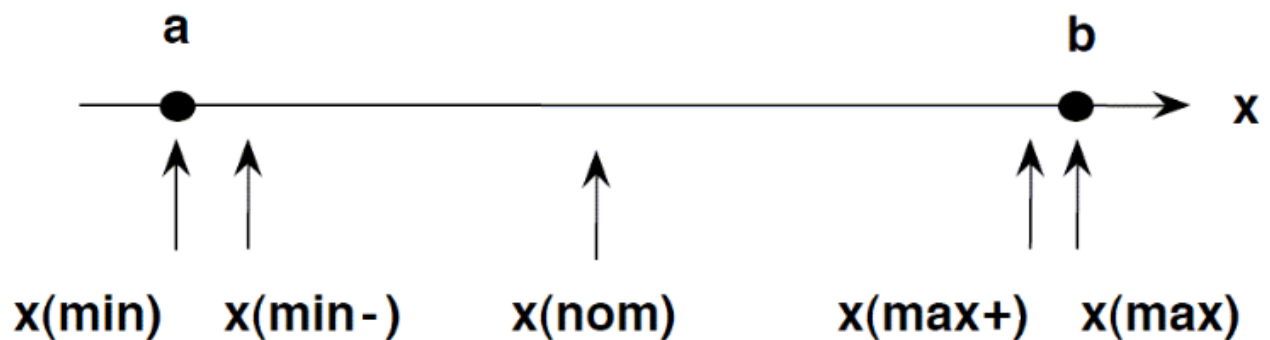
- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again

Techniques Used in Black Box Testing

- Boundary Value Analysis,
- Equivalence Class Testing,
- State Table Based testing,
- Cause Effect Graphing Based Testing,
- Error Guessin

Boundary value analysis

- **Boundary value analysis** is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.
- Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.
- Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.
- For each variable we check-
 - Minimum value.
 - Just above the minimum.
 - Nominal Value.
 - Just below Max value.
 - Max value.



- In Boundary Testing, Equivalence Class Partitioning plays a good role
- Boundary Testing comes after the Equivalence Class Partitioning.

Equivalence Class Testing:

- 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 principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

Example 1: Equivalence and Boundary Value

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order Pizza:

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

Advantages and disadvantages of Equivalence Partitioning technique

Following are pros and cons of equivalence partitioning technique:

Advantages	disadvantages
It is process-oriented	All necessary inputs may not cover.
We can achieve the Minimum test coverage	This technique will not consider the condition for boundary value analysis.
It helps to decrease the general test execution time and also reduce the set of test data.	The test engineer might assume that the output for all data set is right, which leads to the problem during the testing process.

State Transition

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 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 behavior.

It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

State Transition Testing Technique is helpful where you need to **test different system transitions**.

Objectives of State Transition Testing:

The objective of State Transition testing is:

- To test the behavior of the system under varying input.
- To test the dependency on the values in the past.
- To test the change in transition state of the application.
- To test the performance of the system.

*Transition States:

- **Change Mode:**
When this mode is activated then the display mode moves from TIME to DATE.
- **Reset:**
When the display mode is TIME or DATE, then reset mode sets them to ALTER TIME or ALTER DATE respectively.
- **Time Set:**
When this mode is activated, display mode changes from ALTER TIME to TIME.
- **Date Set:**
When this mode is activated, display mode changes from ALTER DATE to DATE.

State Transition Diagram:

State Transition Diagram shows how the state of the system changes on certain inputs.

It has four main components:

- (a) States
- (b) Transition
- (c) Events
- (d) Actions

*State Transition Diagram and State Transition Table

There are two main ways to represent or design state transition, State transition diagram, and state transition table.

In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.

In state transition table all the states are listed on the left side, and the events are described on the top. Each cell in the table represents the state of the system after the event has occurred. It is also called State Table. It is useful in identifying invalid transitions.

	Correct Password	Incorrect Password
S1) Start	S6	S2
S2) 1 st Try	S6	S3
S3) 2nd Try	S6	S4
S4) 3rd Try	S6	S5
S5) 4th Try	S6	S7
S6) Access	-	-
S7) Close Application	-	-

All the valid states are listed on the left side of the table while invalid states on the right side

In a State Table, all the valid states are listed on the left side of the table, and the events that cause them on the top.

Each cell represents the state system will move to when the corresponding event occurs.


For example, while in S1 state you enter a correct password you are taken to state S6 (Access Granted). Suppose if you have entered the wrong password at first attempt you will be taken to state S3 or 2nd Try

Advantages of State Transition Testing:

- State transition testing helps in understanding the behavior of the system.
- State transition testing gives the proper representation of the system behavior.
- State transition testing covers all the conditions.

Disadvantages of State Transition Testing:

- State transition testing can not be performed everywhere.

 **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.

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.

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.

Validation Activities:

Unit validation:

- Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.
- Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness.
- A unit may be an individual function, method, procedure, module, or object.
- Unit tests help to fix bugs early in the development cycle and save costs.
- It helps the developers to understand the testing code base and enables them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

What is Integration Testing?

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

Example of Integration Test Case

Integration [Test Case](#) differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in [Unit Testing](#). But check how it's linked to the Mail Box Page.

Types of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following
 - Top Down Approach
 - Bottom Up Approach
 - Sandwich Approach – Combination of Top Down and Bottom Up

Big Bang Testing

Big Bang Testing is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

Incremental Testing

In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up
- Top Down

Stub: Is called by the Module under Test.

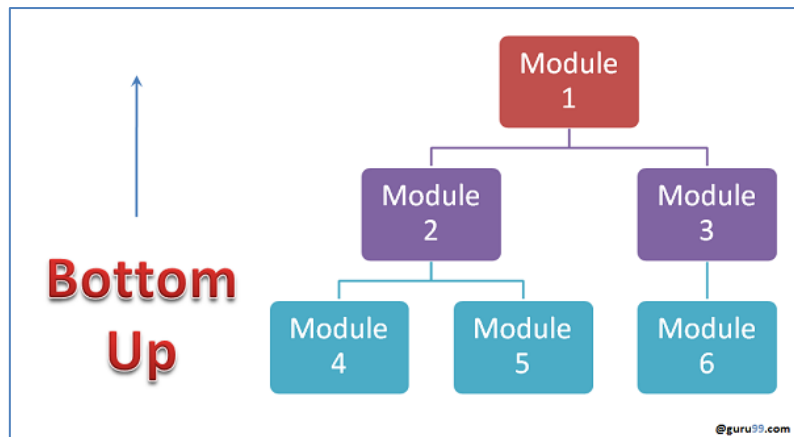
Driver: Calls the Module to be tested.

Bottom-up Integration Testing

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

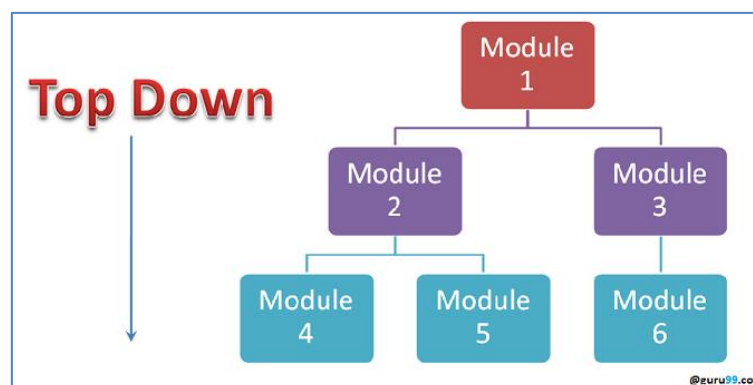


Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

🔗 Top-down Integration Testing

Top Down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.



Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

🔗 Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called Hybrid Integration Testing. It makes use of both stubs as well as drivers.

🔍 What is System Testing?

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is defined as a series of different tests whose sole purpose is to exercise the full computer-based system.

Acceptance Testing

Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not.

S.No	System Testing	Acceptance Testing
1.	System testing is done to check whether the software or product meets the specified requirements or not.	Acceptance testing is the type of testing which is used to check whether the software meets the customer requirements or not.
2.	System testing is used by developers as well as testers.	Acceptance testing is used by testers, stakeholders as well as clients.
3.	System Testing is both functional and non-functional testing.	Acceptance testing is only functional testing.
4.	System Testing is the constitute of System and integration testing.	Acceptance testing is the constitute of alpha and beta testing.
5.	System testing is done before the Acceptance testing.	Acceptance testing is done after the System testing.
6.	System testing is the constitute of positive as well as negative test cases.	Acceptance Testing is the constitute of positive test cases.
7.	In system testing, system is checked for dummy inputs.	In acceptance testing, system is checked for random inputs.

Regression Testing:

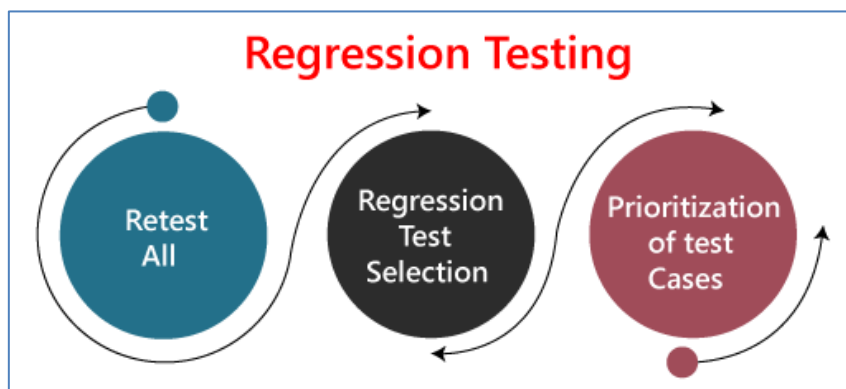
- Regression testing is a black box testing techniques. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.
- Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.
- Regression testing can be performed on a new build when there is a significant change in the original functionality. It ensures that the code still works even when the changes are occurring. Regression means Re-test those parts of the application, which are unchanged.
- Regression tests are also known as the Verification Method.

Need of Regression Testing:

This testing is required when there is any:

- Change in requirements and code is modified as per the changed requirements
- Added new features in product
- Bug fixing
- Fixing of performance related issues.

Regression testing can be performed using the following **techniques**:



1. Re-test All:

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection:

- In this technique, a selected test-case suit will execute rather than an entire test-case suit.
- The selected test case suits divided in two cases
 - Reusable Test cases.
 - Obsolete Test cases.
- Reusable test cases can use in succeeding regression cycle.
- Obsolete test cases can't use in succeeding regression cycle.

3. Prioritization of test cases:

Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

Types of Regression Testing:

The different types of Regression Testing are as follows:

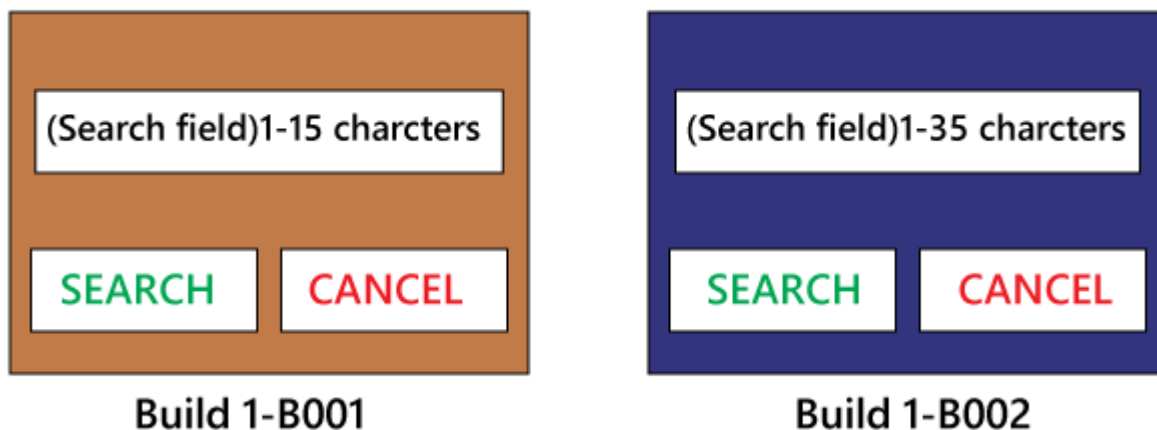
- (a) Unit Regression Testing [URT]
- (b) Regional Regression Testing[RRT]
- (c) Full or Complete Regression Testing [FRT]

1) Unit Regression Testing [URT]:

In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.

Example1

In the below application, and in the first build, the developer develops the **Search** button that accepts **1-15 characters**. Then the test engineer tests the Search button with the help of the **test case design technique**.



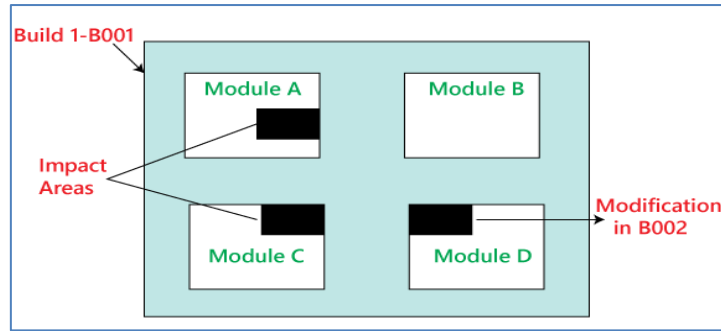
Now, the client does some modification in the requirement and also requests that the **Search button** can accept the **1-35 characters**. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.

2) Regional Regression testing [RRT]:

In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.

For example:

In the below image as we can see that we have four different modules, such as **Module A, Module B, Module C, and Module D**, which are provided by the developers for the testing during the first build. Now, the test engineer will identify the bugs in **Module D**. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.

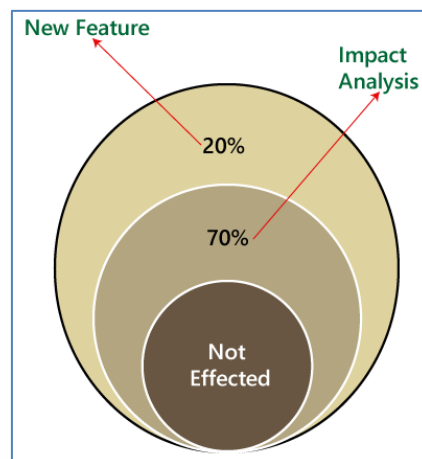


In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in **Module A and Module C**. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in **Module A and Module C**. Therefore, this testing is known as **Regional regression testing**.

3.Full Regression testing [FRT]

During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.

Therefore, we can say that testing the **modified features** and **all the remaining (old) features** is called the **Full Regression testing**.



Advantages of Regression Testing

Advantages of Regression Testing are:

- Regression Testing increases the product's quality.
- It ensures that any bug fix or changes do not impact the existing functionality of the product.
- Automation tools can be used for regression testing.
- It makes sure the issues fixed do not occur again.

Disadvantages of Regression Testing

There are several advantages of Regression Testing though there are disadvantages as well.

- Regression Testing should be done for small changes in the code because even a slight change in the code can create issues in the existing functionality.
- If in case automation is not used in the project for testing, it will time consuming and tedious task to execute the test again and again.

Aspect	Black Box Testing	White Box Testing
Focus	External behaviour and functionality	Internal logic, code structure, and implementation
Knowledge	Tester doesn't need knowledge of internal code	Tester has knowledge of internal code
Objective	Validate correctness as per requirements/specifications	Verify code structure, optimization, and logic
Testing Type	Functional, non-functional, integration, acceptance	Unit testing, code coverage, security testing
Approach	Tests based on inputs and expected outputs	Tests based on code paths and data flow
Test Design	Inputs determined by requirements	Inputs designed to exercise code paths
Test Cases	Designed without knowing internal logic	Designed with knowledge of internal logic
Independence	Can be done by external parties (e.g., users)	Usually performed by developers or testers

Aspect	Progressive Testing	Regressive Testing
Definition	Testing performed to evaluate new functionality or features as they are added to the software.	Testing conducted to ensure that recent changes, bug fixes, or enhancements have not adversely affected existing features.
Timing	Typically conducted during the development phase, often alongside new feature development.	Usually performed after development is complete or during a separate testing phase.
Scope	Focused on testing the specific changes or new features introduced in the software.	Involves testing the entire application to ensure that existing functionalities have not regressed due to recent changes.
Objective	To verify that the new features or changes meet their intended requirements and do not introduce new defects.	To confirm that existing features and functionalities remain intact and work as expected after recent changes or fixes.
Test Cases	Test cases are often designed to target the new functionalities and changes, and they may not cover the entire application.	Test cases cover the full range of functionalities and features in the application, including both new and existing ones.
Automation	Test automation is common for repetitive test cases in progressive testing, especially for regression test suites.	Test automation is widely used for regressive testing to efficiently retest existing functionalities.
Frequency	Progressive testing is ongoing and happens continuously as new features are developed.	Regressive testing is usually performed in dedicated cycles before a release or as part of the release process.
Defect Discovery	New defects or issues are often discovered in progressive testing, particularly in the newly introduced features.	Existing defects or regressions in previously working features may be identified in regressive testing.
Feedback Loop	Feedback is provided to the development team regarding the quality and functionality of the newly implemented features.	Feedback helps ensure that changes or fixes do not negatively impact the overall stability of the application.
Test Environment	Testing environments for progressive testing may be less complex, focusing on the new features being developed.	Regressive testing environments should closely mirror the production environment to mimic real-world usage.
Resource Allocation	Development and testing resources are allocated concurrently to develop.	Regressive testing typically requires a dedicated testing effort and may involve a larger testing team.