Module 5

Software Testing

What is testing

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce a quality product.

Definition

According to ANSI/IEEE 1059 standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Difference between Bug, Defect, Error, Fault & Failure

Bug

In software testing, a bug is the informal name of defects, which means that software or application is not working as per the requirement.

Defect

When the application is not working as per the requirement is knows as defects. It is specified as the aberration from the actual and expected result of the application or software.

In other words, we can say that the bug announced by the programmer and inside the code is called a Defect.

Error

The Problem in code leads to errors, which means that a mistake can occur due to the developer's coding error as the developer misunderstood the requirement or the requirement was not defined correctly. The developers use the term error.

fault

A fault may happen in a program because of the following reasons:

- Lack of resources
- An invalid step
- Inappropriate data definition

Failure

we can say that if an end-user detects an issue in the product, then that particular issue is called a failure.

What is Test case?

A test case is a document, which has a set of test data, preconditions, expected results and post conditions. It is used to test a particular software or component of software against its requirements.

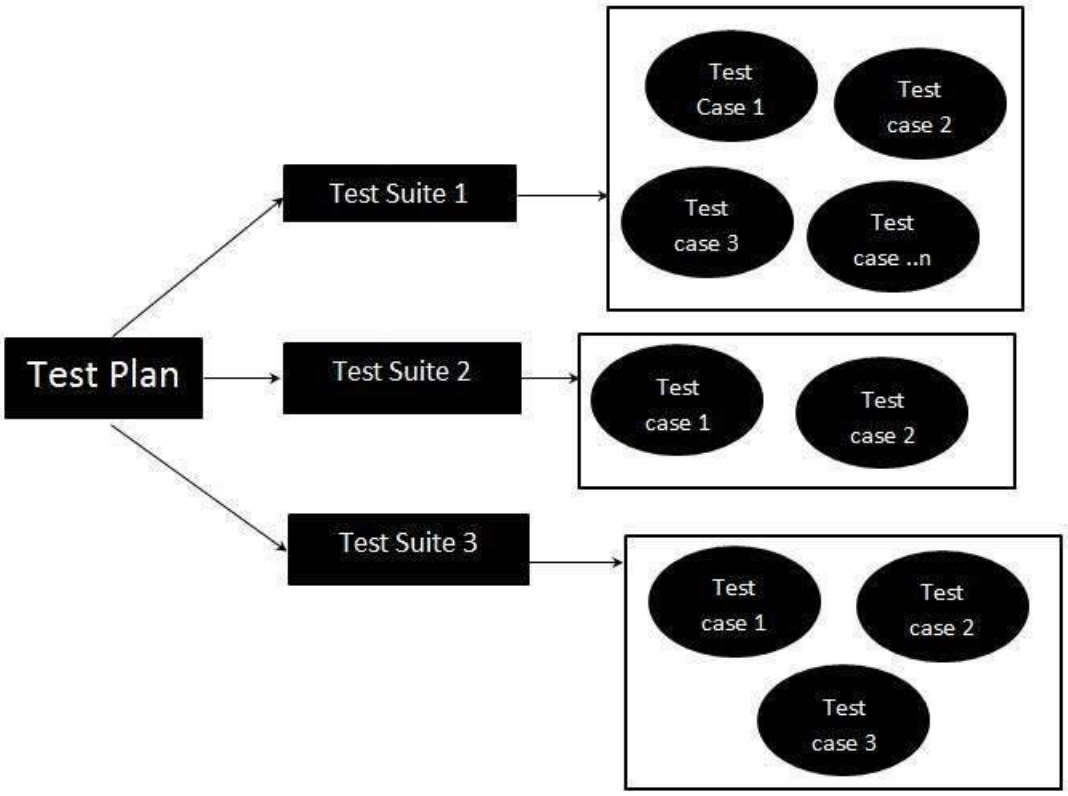
The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not

Test suite

Test suite is a container that has a set of tests which helps testers in executing and reporting the test execution status. It can take any of the three states namely Active, Inprogress and completed.

A Test case can be added to multiple test suites and test plans. After creating a test plan, test suites are created which in turn can have any number of tests.

Test suite diagram



Verification and validation in software testing

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

Verification: *Are we building the product right?*
Validation: *Are we building the right product?*

Verification:
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is **Static Testing**.

Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.
Validation is the **Dynamic Testing**.
Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing

Difference between verification and validation

Verification	**Validation**
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.

Verification	Validation
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Acceptance testing

Acceptance testing is formal testing based on user requirements and function processing. It determines whether the software is conforming specified requirements and user requirements or not. It is conducted as a kind of Black Box testing where the number of required users involved testing the acceptance level of the system. It is the fourth and last level of software 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.

Alpha Testing

Alpha testing is a type of acceptance testing, which is performed to identify all possible bugs/issues before releasing the product to the end-user. Alpha testing implies a meeting with a software vendor and client to ensure that the developers appropriately meet the client's requirements in terms of the performance, functionality, and durability of the software.

Alpha testing needs lab environment, and usually, the testers are an internal employee of the organization. This testing is called alpha because it is done early on, near the end of the software development, but before beta testing.

Beta Testing

Beta Testing is a type of acceptance testing; it is the final test before shipping a product to the customers. Beta testing of a product is implemented by "real users "of the software application in a "real environment."

In this phase of testing, the software is released to a limited number of end-users of the product to obtain feedback on the product quality. It allows the real customers an opportunity to provide inputs into the design, functionality, and usability of the product. These inputs are essential for the success of the product.

Beta testing reduces product failure risks and increases the quality of the product through customer validation. Direct feedback from customers is a significant advantage of beta testing.

Difference between alpha testing and beta testing

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.

Alpha Testing	Beta Testing
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.

Functional Testing:

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

Functional testing also called as black-box testing, because it focuses on application specification rather than actual code. Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Techniques to design test cases

1. Boundary value analysis

2. Equivalence class testing

3. Decision table based testing

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.

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. Boundary Value Analysis is also called **range checking**.

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

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

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. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduce time required for testing because of small number of test cases.

- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

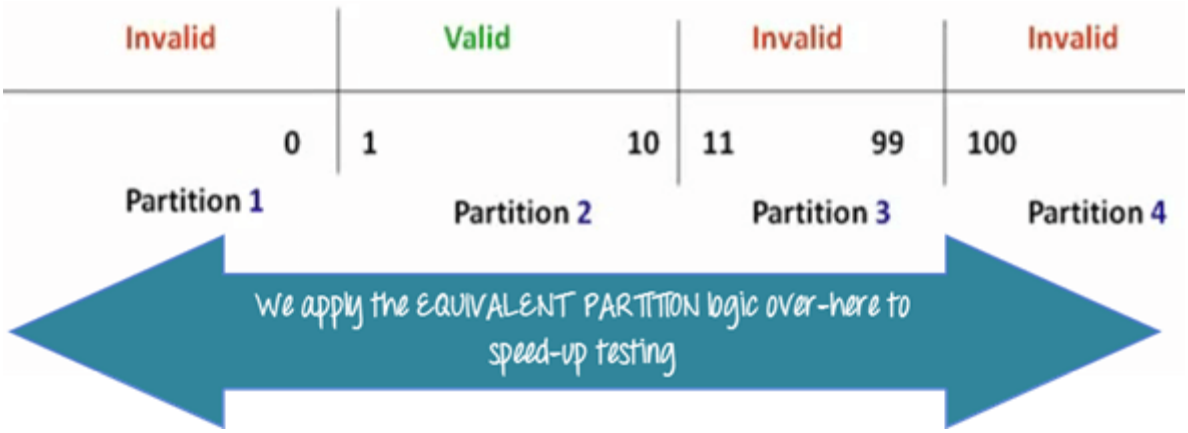
Example 1: Equivalence and Boundary Value

- Let's consider the behaviour of Order Pizza Text Box Below
- Pizza values 1 to 10 are 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"**

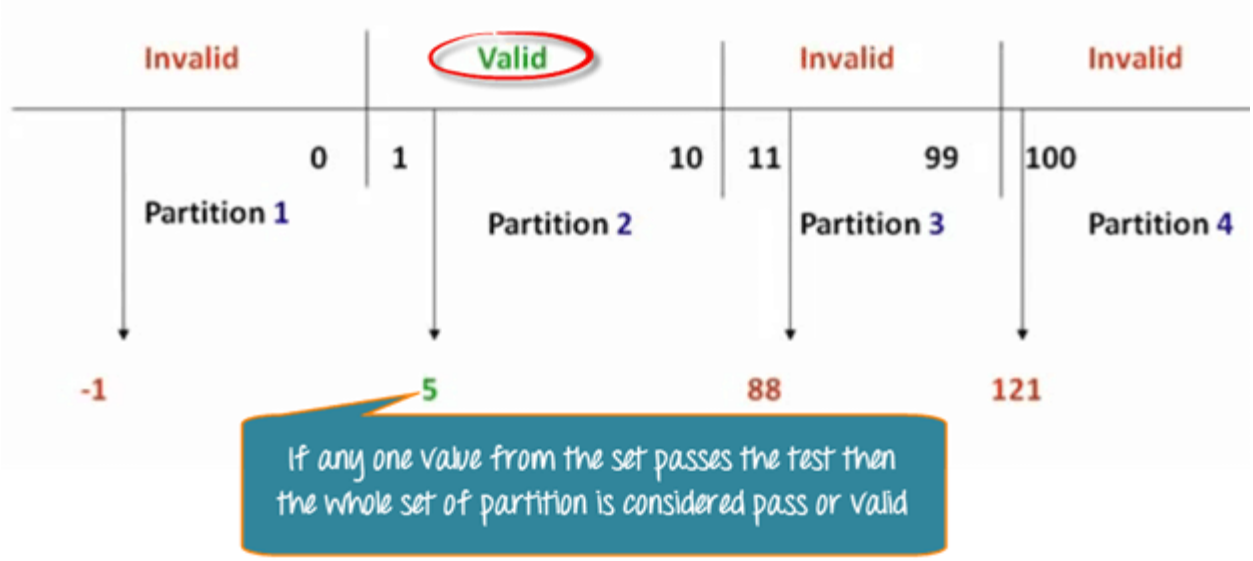
Here is the test condition

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.

We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.



The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.



Decision table based testing

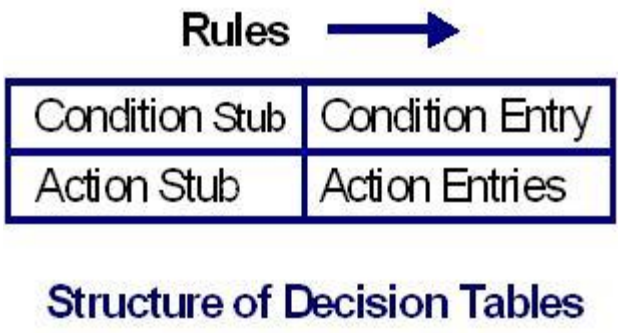
Decision table is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

A decision table is a good way to settle with different combination inputs with their corresponding outputs and also called cause-effect table. Reason to call cause-effect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

It is another popular black box testing. A decision table has following four portions

(a) Condition stub
(b) Action stub
(c) Condition Entry
(d) Action Entries

Structure of decision table: Please refer following decision table



Following steps are used to identify the test cases with decision tables:

Step – 1: For a module, identify input conditions (causes) and action (effect).

Step – 2: Develop a cause-effect graph.

Step – 3: Transform this cause-effect graph to a decision table.

Step – 4: Convert decision table rules to test cases.

Each column of the decision table represents a test case.

	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Cause Effect Graphing

Cause Effect Graphing based technique is a technique in which a graph is used to represent the situations of combinations of input conditions. The graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions. But since there may be some critical behaviour to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

Steps used in deriving test cases using this technique are:

1. **Division of specification:**
   Since it is difficult to work with cause-effect graphs of large specifications as they are complex, the specifications are divided into small workable pieces and then converted into cause-effect graphs separately.
2. **Identification of cause and effects:**
   This involves identifying the causes(distinct input conditions) and effects(output conditions) in the specification.
3. **Transforming the specifications into a cause-effect graph:**
   The causes and effects are linked together using Boolean expressions to obtain a cause-effect graph. Constraints are also added between causes and effects if possible.
4. **Conversion into decision table:**
   The cause-effect graph is then converted into a limited entry decision table. If you're not aware of the concept of decision tables
5. **Deriving test cases:**
   Each column of the decision-table is converted into a test case.

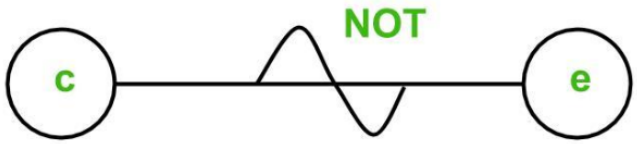
Basic Notations used in Cause-effect graph:

Here **c** represents **cause** and **e** represents **effect**.

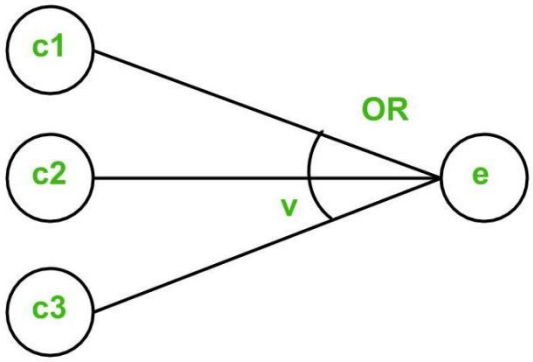
The following notations are always **used between a cause and an effect**:

1. **Identity Function:** if c is 1, then e is 1. Else e is 0.

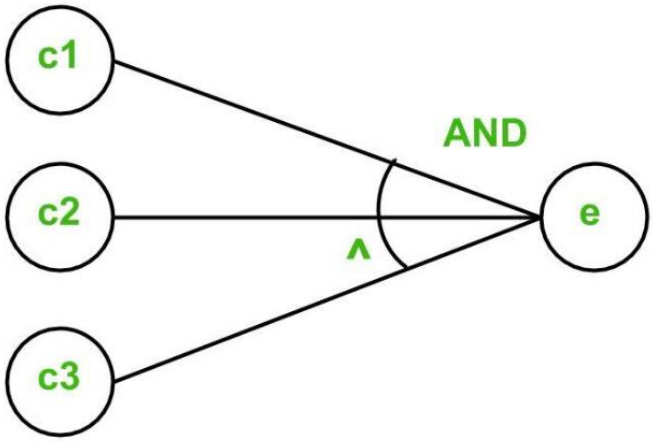
2. **NOT Function:** if c is 1, then e is 0. Else e is 1.



3. **OR Function:** if c1 or c2 or c3 is 1, then e is 1. Else e is 0.



4. **AND Function:** if both c1 and c2 and c3 is 1, then e is 1. Else e is 0.



Structural Testing (White box Testing)

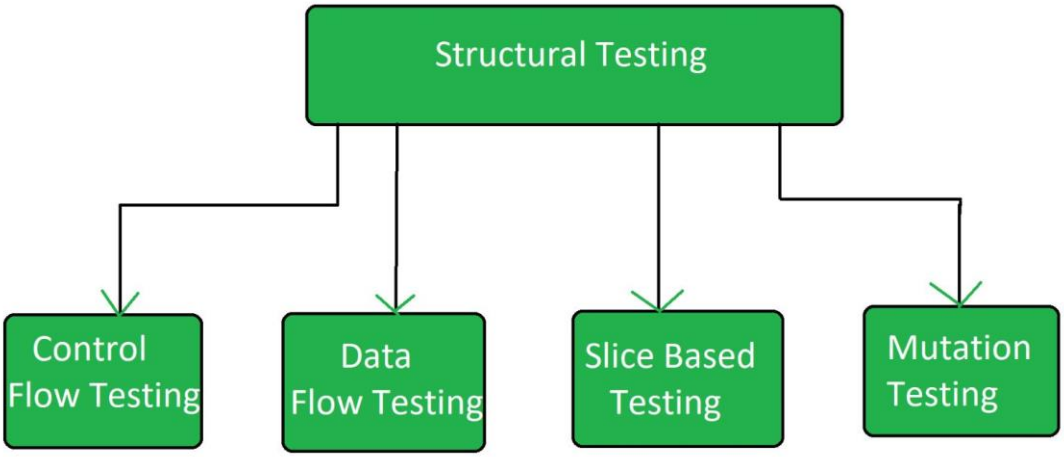
Structural testing is a type of software testing which uses the internal design of the software for testing or in other words the software testing which is performed by the team which knows the development phase of the software, is known as structural testing.

Structural testing is basically related to the internal design and implementation of the software i.e. it involves the development team members in the testing team. It basically tests different aspects of the

software according to its types. Structural testing is just the opposite of behavioral testing.

Structural testing is also known as white-box testing, **glass box testing**, and **clear-box testing**. Developers mostly implement it to identify the issue and fix them quickly.

Types of Structural Testing:
There are 4 types of Structural Testing:



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

Control flow testing

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the

program. In this technique, a particular part of a large program is selected by the tester to set the testing path

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
2. Edge
3. Decision Node
4. Junction node

Node represents the sequence of procedures which procedure is next to come so on

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

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

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

Slice based testing

It was originally proposed by Weiser and Gallagher for the software maintenance. It is useful for software debugging, software maintenance, program understanding and quantification of functional cohesion. It divides the program into different slices and tests that slice which can majorly affect the entire software.

Data flow testing

Data Flow Testing is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams.

It is concerned with:

- Statements where variables receive values,
- Statements where these values are used or referenced.

Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program. Anomalies in the flow of the data are detected at the time of associations between values and variables.

These anomalies are:

- A variable is defined but not used or referenced,
- A variable is used but never defined,
- A variable is defined twice before it is used

Advantages of Data Flow Testing:

Data Flow Testing is used to find the following issues-

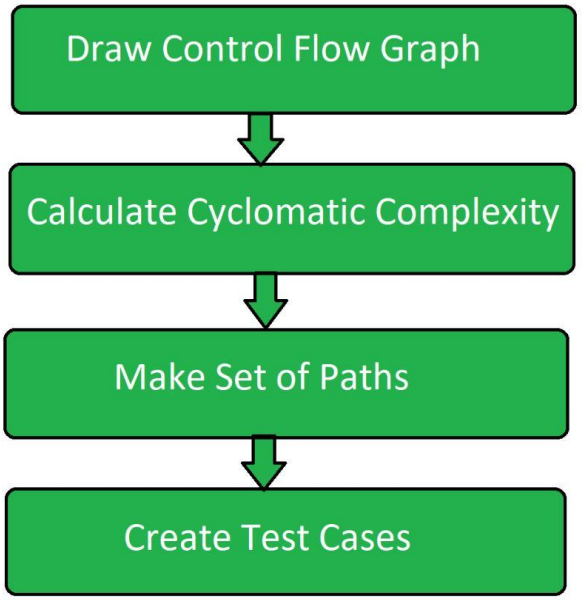
- To find a variable that is used but never defined,
- To find a variable that is defined but never used,
- To find a variable that is defined multiple times before it is use,
- De-allocating a variable before it is used.

Disadvantages of Data Flow Testing

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

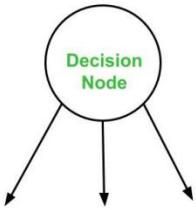
Path testing

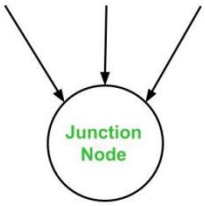
Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution. In this method Cyclomatic Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path.

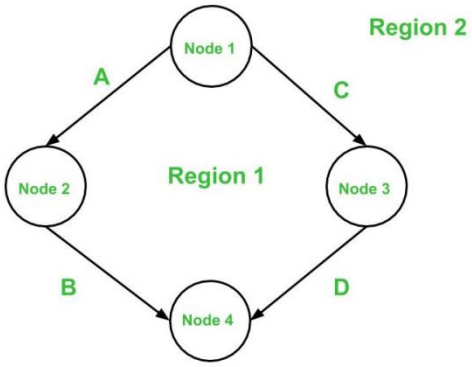


Control Flow Graph:

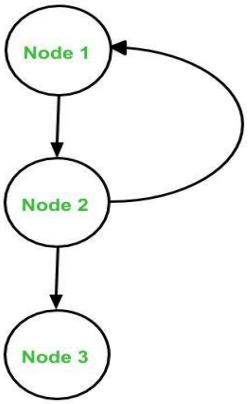
It is also called flow graph. We should draw the corresponding control flow graph of the program in which all the executable paths are to be discovered. Notations are



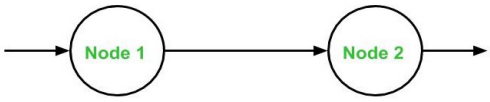




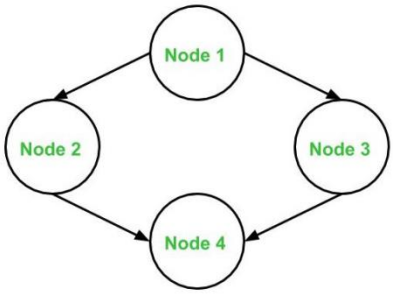
Do - While

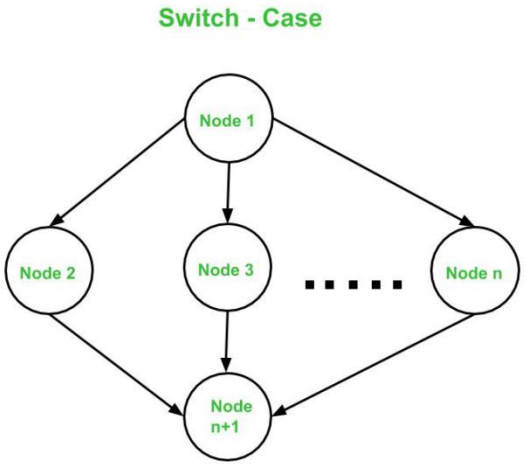


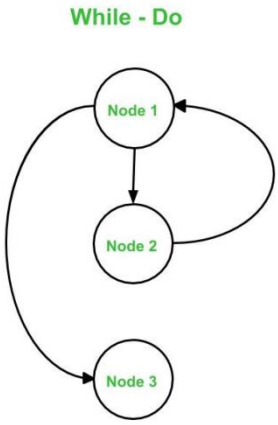
Sequence



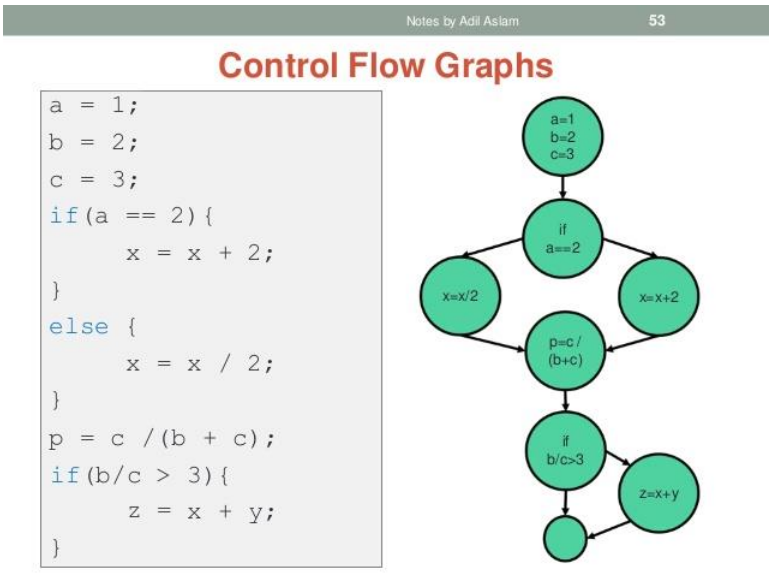
If - Then - Else




Switch - Case
Node 1
Node 2
Node 3
Node n
Node n+1


While - Do
Node 1
Node 2
Node 3

Example


Notes by Adil Aslam 53
Control Flow Graphs
a = 1;
b = 2;
c = 3;
if(a == 2){
x = x + 2;
}
else {
x = x / 2;
}
p = c /(b + c);
if(b/c > 3){
z = x + y;
}
a=1
b=2
c=3
if
a==2
x=x/2
x=x+2
p=c /
(b+c)
if
b/c>3
z=x+y

Calculating Cyclomatic Complexity

After the generation of the control flow graph, calculate the cyclomatic complexity of the program using the following formula.
McCabe's Cyclomatic Complexity = E - N + 2P

Where,

E = Number of edges in control flow graph

N = Number of vertices in control floe graph

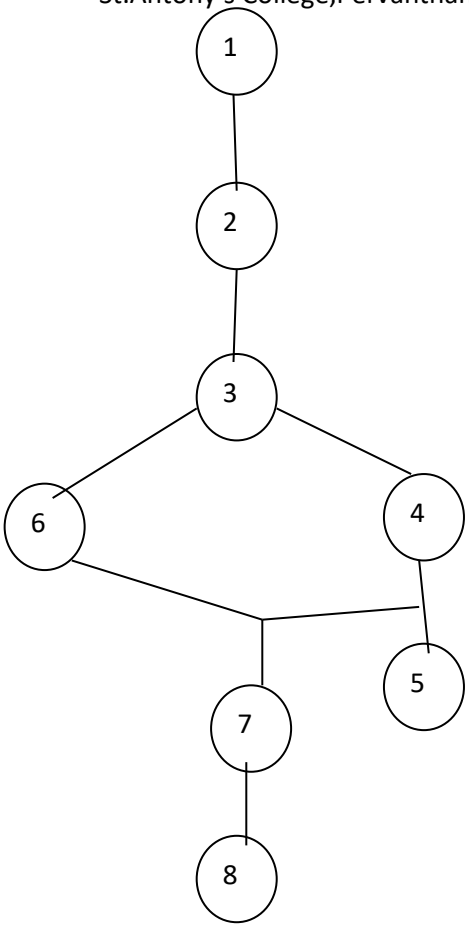
P = number of connected components.

Path Testing Techniques:

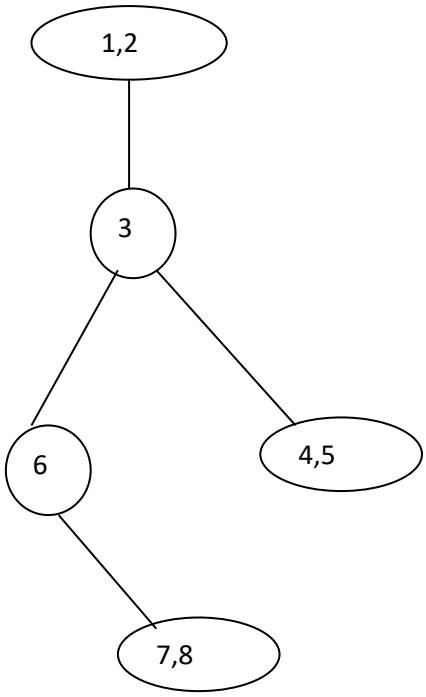
- **Control Flow Graph:**
  The program is converted into control flow graph by representing the code into nodes and edges.
- **Decision to Decision path:**
  The control flow graph can be broken into various Decision to Decision paths and then collapsed into individual nodes.
- **Independent paths:**
  Independent path is a path through a Decision to Decision path graph which cannot be reproduced from other paths by other methods.

Decision to Decision path(DD Path Graph)

Second step in path testing is to draw a dd graph from the flow graph.The nodes of the flow graph which are in sequences are combined into a single node in DD graph



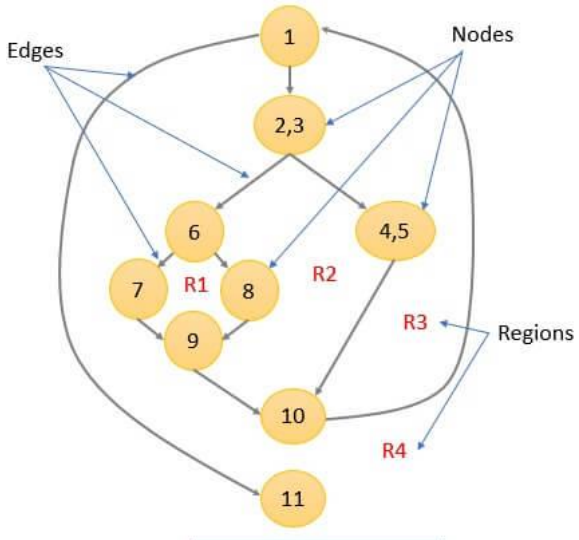
Flow graph



DD Path graph

Independent Path

The independent path traces the edges in the flow graph that are not traversed before the path is defined.



Flow Graph for above Flow Chart

As in the above flow graph, we can observe that the starting or the first node or procedural statement is node 1 and the termination of the flow graph is node 11. So, all the independent path that can be traced in the flow graph should start from 1 and end to 11:

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

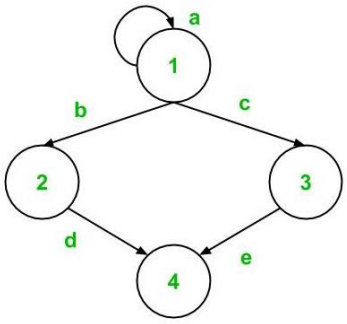
Path 4: 1-2-3-6-7-9-10-1-11

Path 1, 2, 3 & 4 are the independent paths as each path introduces a new edge i.e. all the paths are **unique**.

Graph Matrix

A graph matrix is a square matrix whose size represents the number of nodes in the control flow graph. 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 take an example.



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

Levels of Software Testing

Software Testing is an activity performed to identify errors so that errors can be removed to obtain a product with greater quality. To assure and maintain the quality of software and to represents the ultimate review of specification, design, and coding, Software testing is required. There are different levels of testing :

Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself.

Unit Testing is defined as a type of software testing where individual components of a software are tested.An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

Integration Testing

In this testing, two or more modules which are unit tested are integrated to test i.e. technique interacting components and are then verified if these integrated modules work as per the expectation or not and interface errors are also detected.

The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

System Testing

In system testing, complete and integrated Software's are tested i.e. all the system elements forming the system is tested as a whole to meet the requirements of the system.

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between

the units that are integrated together. System testing detects defects within both the integrated units and the whole system

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer.

Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?