



Object Oriented Software Engineering

1

LECTURE 14

Testing Tactics:

Black-Box Testing

- Equivalence partitioning
- Boundary value analysis

White Box Testing

- Basis Path Testing
- Control Structure Testing: Condition and loop testing



I. Black Box Testing (Functional testing)

- Equivalence Partitioning
- Boundary Value Analysis
- Cause Effect Graphing

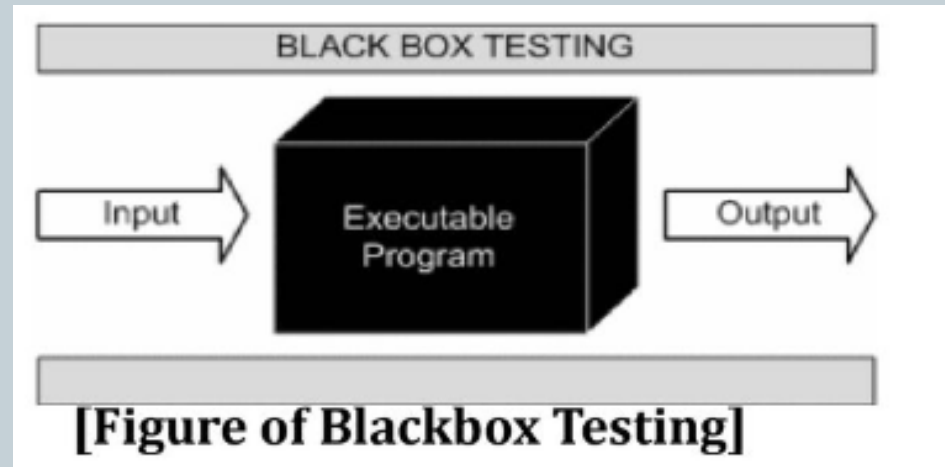
II. White Box Testing (Structural testing)

- Coverage Testing
 - Statement Coverage
 - Branch And Decision Coverage
 - Path Coverage
- Comparison of black box and white box testing

Black box testing

4

- In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.



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



Black box Testing

5

- By applying black-box techniques, you derive a set of test cases that satisfy the following criteria:
- (1) test cases that reduce, by a count that is greater than one the number of additional test cases that must be designed to achieve reasonable testing, and
- (2) test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.



BLACKBOX TESTING

○ Blackbox Techniques/ Methods:

- **Equivalence partitioning:** It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- **Boundary Value Analysis:** It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- **Cause effect graphing:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.



BLACKBOX TESTING

- **1. Equivalence partitioning:**
- Equivalence Partitioning also called as equivalence class partitioning.
- It can be applied at any level of testing and is often a good technique to use first.
- The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'.
- Equivalence partitioning is a testing technique where input values set into classes for testing.
 - Valid Input Class = Keeps all valid inputs.
 - Invalid Input Class = Keeps all Invalid inputs.





BLACKBOX TESTING

- **1. Equivalence partitioning:**
- **Equivalence classes may be defined according to the following guidelines:**
 - 1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
 - 2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
 - 3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
 - 4. If an input condition is Boolean, one valid and one invalid class are defined.
 - By applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed.




Equivalence Partitioning

9

- It divides the i/p domain into various classes of data.
- Guidelines for generating equivalence classes
- A) If i/p condition specify a range

Range → 21 to 25




 <21 Invalid	 21-25 Valid	 >25 Invalid
--	---	--

1- Valid
2- Invalid

Equivalence Partitioning

10

- B) If i/p condition requires a specific value
Value \rightarrow 18

		
<18 Invalid	18 Valid	>18 Invalid

1- Valid
2- Invalid

Equivalence Partitioning

11

- C) If i/p condition specifies a member of set
E.g. Value \rightarrow 4 Set {1,2,3,4,5}

Value of set- **Valid**

Not a value of set- **Invalid**

1- Valid
1- Invalid



- D) If i/p value is Boolean:
- True-Valid
- False- Invalid

1- Valid
1- Invalid

BLACKBOX TESTING

- **1. Equivalence partitioning:**
- **Example-1:**
- A text field permits only numeric characters
- Length must be 6-10 characters long
- Partition according to the requirement should be like this:

0 1 2 3 4 5 | 6 7 8 9 10 | 11 12 13 14
Invalid | Valid | Invalid

Figure- Example of Equivalence partitioning]

- While evaluating Equivalence partitioning, values in all partitions are equivalent that's why 0-5 are equivalent, 6 – 10 are equivalent and 11- 14 are equivalent.



BLACKBOX TESTING

- **1. Equivalence partitioning:**
- At the time of testing, test 4 and 12 as invalid values and 7 as valid one.
- It is easy to test input ranges 6–10 but harder to test input ranges 2-600. Testing will be easy in the case of lesser test cases but you should be very careful.
- Assuming, valid input is 7. That means, you believe that the developer coded the correct valid range (6-10).





Practice Question

15

- **Q #1) One of the fields on a form contains a text box that accepts numeric values in the range of 18 to 25. Identify the invalid Equivalence class.**

- a) 17
- b) 19
- c) 24
- d) 21

Ans. 17

- **Solution:**
- The text box accepts numeric values in the range of 18 to 25 (18 and 25 are also part of the class). So this class becomes our valid class. But the question is to identify invalid equivalence classes. The classes will be as follows:
 - Class I: values $< 18 \Rightarrow$ invalid class
 - Class II: 18 to 25 \Rightarrow valid class
 - Class III: values $> 25 \Rightarrow$ invalid class
- 17 falls under an invalid class. 19, 24 and 21 fall under valid class.

Q #2) In an Examination, a candidate has to score a minimum of 24 marks in order to clear the exam. The maximum that he can score is 40 marks. Identify Valid Equivalence values if the student clears the exam.

- a) 22,23,26
- b) 21,39,40
- c) 29,30,31
- d) 0,15,22

Ans. C

• **Solution:**

- The classes will be as follows:

Class I: values $< 24 \Rightarrow$ invalid class

Class II: 24 to 40 \Rightarrow valid class

Class III: values $> 40 \Rightarrow$ invalid class

- We need to identify Valid Equivalence values. Valid Equivalence values will be there in a Valid Equivalence class. All the values should be in Class II.

- **Q #3) One of the fields on a form contains a text box that accepts alphanumeric values. Identify the Valid Equivalence class.**
 - a) BOOK
 - b) Book
 - c) Boo01k
 - d) Book
- Ans. C**
- **Solution:**
 - Alphanumeric is a combination of alphabets and numbers. Hence we have to choose an option which has both of these. A valid equivalence class will consist of both alphabets and numbers. Option 'c' contains both alphabets and numbers.

- **Q #4) A program validates numeric fields as follows: values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following input values cover all of the equivalence partitions?**

- a. 10,11,21
- b. 3,20,21
- c. 3,10,22
- d. 10,21,22

Ans. C

- **Solution:**
- We have to select values that fall in all the equivalence classes (valid and invalid).
- The classes will be as follows:
- Class I: values $\leq 9 \Rightarrow$ invalid class
- Class II: 10 to 21 \Rightarrow valid class
- Class III: values $\geq 22 \Rightarrow$ invalid class

BLACKBOX TESTING

- **2. Boundary Value Analysis:**
- It's widely recognized that input values at the extreme ends of input domain cause more errors in system. More application **errors occur at the boundaries** of input domain.
- 'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.
- Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.
- Boundary value analysis is the process of testing between extreme ends or boundaries between partitions' of the input values.



BLACKBOX TESTING

- **2. Boundary Value Analysis:**
- Boundary value analysis is the process of testing between extreme ends or boundaries between partitions' of the input values.
- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary value analysis testing".
- **Example-1:**
- Suppose you have very important tool at office, accepts valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Valid range 8-12, Invalid range 7 or less than 7 and Invalid range 13 or more than 13.

BLACKBOX TESTING

○ 2. Boundary Value Analysis:



[Figure-Example of Boundary Value Analysis]

- Write Test Cases for Valid partition value, Invalid partition value and exact boundary value.
 - Test Cases 1: Consider password length less than 8.
 - Test Cases 2: Consider password of length exactly 8.
 - Test Cases 3: Consider password of length between 9 and 11.
 - Test Cases 4: Consider password of length exactly 12.
 - Test Cases 5: Consider password of length more than 12.



Boundary Value Analysis (BVA)

22

- Large no. of errors occur at boundaries
- The BVA actually selects the test cases at the “edges” of class.
- **I. Range:** between a and b then test case should be designed beyond a and b.

range (3-5) : 1

2

3

4

5

6



- II. If i/p selects min & max then values just above & just below also tested.
- E.g. {20,18, 21,24}
- Min=18 → just above (17)
- Max = 24 → just below (25)



- **Q #5) A program validates numeric fields as follows: values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following input values cover all of the equivalence partitions?**

- a. 10,11,21
- b. 3,20,21
- c. 3,10,22
- d. 10,21,22

Ans. C

- **Solution:**
- We have to select values that fall in all the equivalence classes (valid and invalid).
- The classes will be as follows:
- Class I: values $\leq 9 \Rightarrow$ invalid class
- Class II: 10 to 21 \Rightarrow valid class
- Class III: values $\geq 22 \Rightarrow$ invalid class

- **Q #6) A program validates numeric fields as follows: values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following covers the MOST boundary values?**
- a. 9,10,11,22
- b. 9,10,21,22
- c. 10,11,21,22
- d. 10,11,20,21

Ans. B

Solution:

We have already come up with the classes as shown in question 5. The boundaries can be identified as 9, 10, 21, and 22.

- Let us assume a test case that takes the speed of a car from 40 to 80 to get best fuel efficiency.
- Min: 40
- Max: 80

Boundary Value Test Case

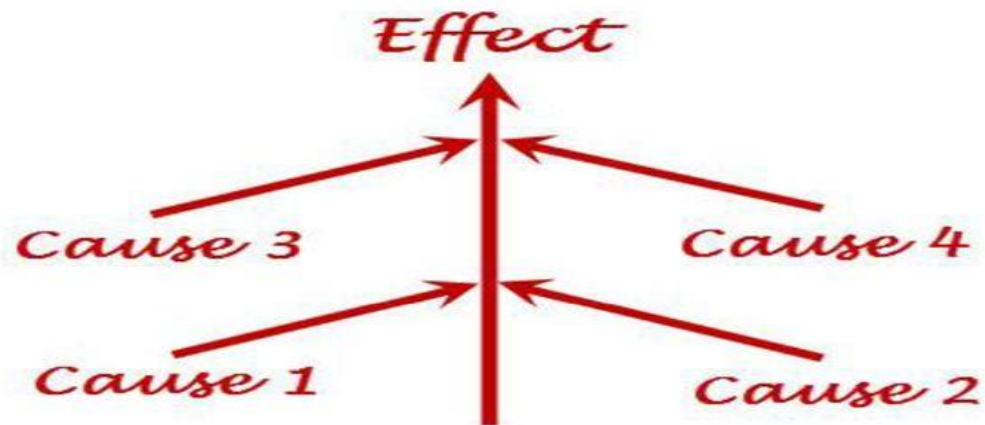
Invalid test Case (Min Value-1)	Valid test case (Min, +Min, -Max,Max)	Invalid Test case (Max value+1)
39	40, 41, 79, 80	81

BLACKBOX TESTING

- **3. Cause effect graphing:** A “Cause” stands for a separate input condition that fetches about an internal change in the system. An “Effect” represents an output condition, a system transformation or a state resulting from a combination of causes.
- It is a testing technique that aids in choosing test cases that logically relate Causes (inputs) to Effects (outputs) to produce test cases.
- **According to Myer Cause & Effect Graphing is done through the following steps:**
 - **Step – 1:** For a module, identify the input conditions (causes) and actions (effect).
 - **Step – 2:** Develop a cause-effect graph.
 - **Step – 3:** Transform cause-effect graph into a decision table.
 - **Step – 4:** Convert decision table rules to test cases. Each column of the decision table represents a test case.



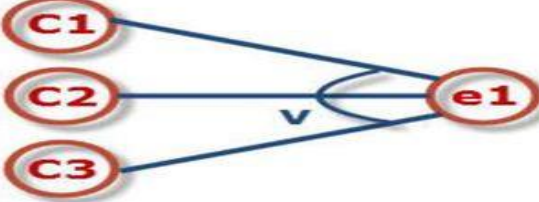
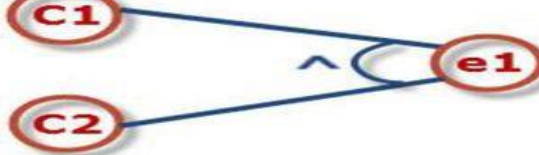
BLACKBOX TESTING

Cause Effect - Flow Diagram



[Figure of Cause Effect-Flow Diagram]

BLACKBOX TESTING

Notation	Meaning
	Identify
	NOT
	OR
	AND

[Symbols used in Cause Effect-Flow Diagram]





BLACKBOX TESTING

○ Example:

- The “Print message” is software that read two characters and, depending of their values, messages must be printed.
- The first character must be an “A” or a “B”.
- The second character must be a digit.
- If the first character is an “A” or “B” and the second character is a digit, the file must be updated.
- If the first character is incorrect (not an “A” or “B”), the message X must be printed.
- If the second character is incorrect (not a digit), the message Y must be printed.

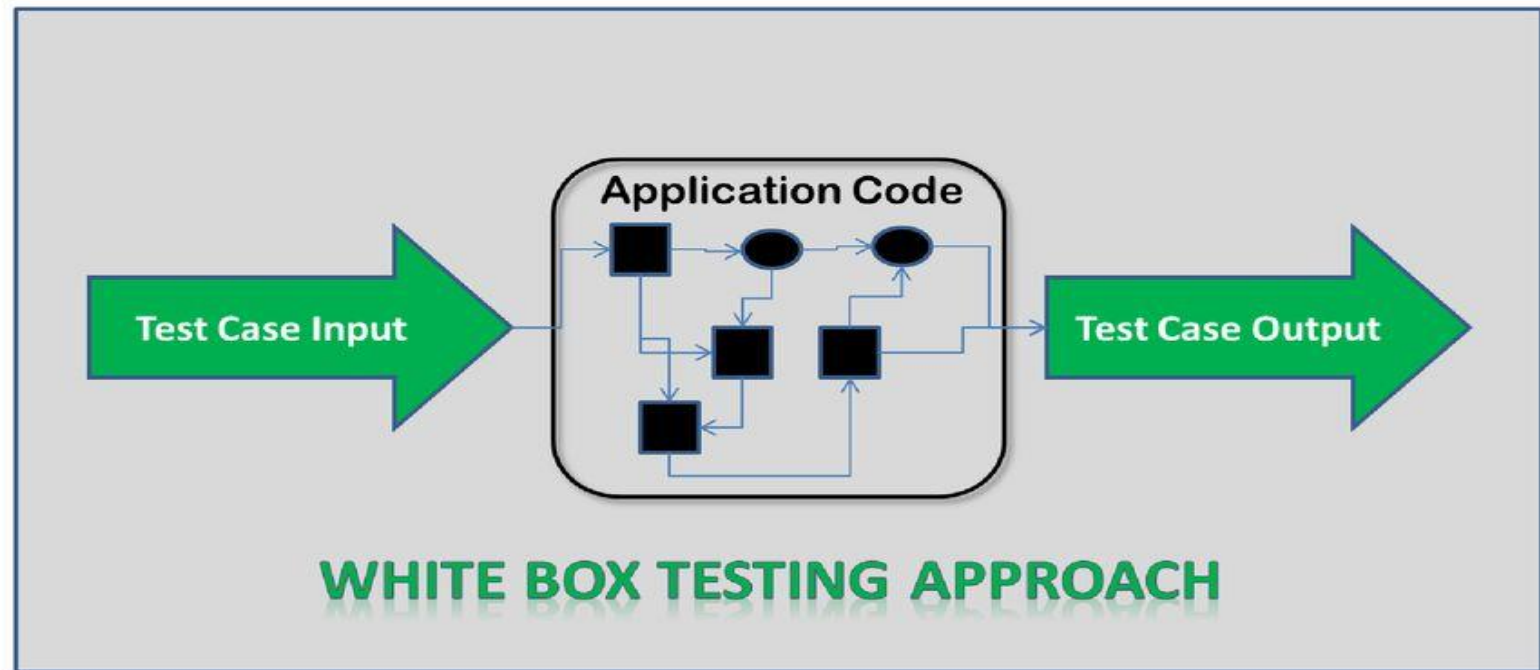




:WHITEBOX TESTING:

- **White Box Testing** (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
- White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.
- Using white-box testing methods, you can derive test cases that
- (1) guarantee that all independent paths within a module have been exercised at least once
- (2) exercise all logical decisions on their true and false sides
- (3) execute all loops at their boundaries and within their operational bounds
- (4) exercise internal data structures to ensure their validity.

WHITEBOX TESTING:



[Figure of Whitebox Testing Approach]



:WHITEBOX TESTING:

- **Why and When White-Box Testing:**
- White box testing is mainly used for **detecting logical errors** in the program code.
- It is used for **debugging a code, finding random typographical errors, and uncovering incorrect programming** assumptions.
- White box testing is done at low level design and implementable code.
- **It can be applied at all levels of system development especially Unit, system and integration testing.**
- White box testing can be used for other development artifacts like requirements analysis, designing and test cases.





:WHITEBOX TESTING:

- **Whitebox Testing Techniques:**
- **Following are Whitebox testing techniques:**
 - **Statement coverage:** This technique is aimed at exercising **all programming statements** with minimal tests.
 - **Branch and decision coverage:** This technique is running a series of tests to ensure that **all branches** are tested at least once.
 - Tools: An example of a tool that handles branch coverage testing for C, C++ and Java applications is TCAT-PATH
 - **Path coverage:** This technique corresponds to testing **all possible paths which means that each statement and branch is covered.**





:WHITEBOX TESTING:

- **1. Statement coverage:**
- Statement coverage is a white box testing technique, which involves the execution of **all the statements at least once in the source code.**
- Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
- The statement coverage **covers only the true conditions.** The main drawback of this technique is that we cannot test the false condition in it.
- The statement coverage is count as per below formula:
- **(Statement coverage = No of statements Executed/Total no of statements in the source code * 100)**





:WHITEBOX TESTING:

○ Example:

○ Read A
Read B
if A>B
Print "A is greater than B"
else
Print "B is greater than A"
endif

○ Set1 :If A =5, B =2

No of statements Executed: 5
Total no of statements in the source code: 7
Statement coverage = $5/7 \times 100 = 71.00 \%$

○ Set1 :If A =2, B =5 [False-Not supported by Statement coverage]

No of statements Executed: 6
Total no of statements in the source code: 7
Statement coverage = $6/7 \times 100 = 85.20 \%$





:WHITEBOX TESTING:

- **2. Branch/Decision coverage:**
- Branch coverage is also known as **Decision coverage or all-edges coverage.**
- It **covers both the true and false conditions** unlikely the statement coverage.
- A branch is the outcome of a decision, so **branch coverage** simply measures which decision outcomes have been tested.
- This sounds great because it takes a more in-depth view of the source code than simple statement coverage
- The formula to calculate decision coverage is:
- **Decision Coverage=(Number of decision outcomes executed/Total number of decision outcomes)*100%**





:WHITEBOX TESTING:

- **Example:**
- **READ X**
- **READ Y**
- **IF (X > Y)**
- **PRINT "X is greater than Y"**
- **ENDIF**
- To get 100% statement coverage only one test case is sufficient for this pseudo-code.
- **TEST CASE 1: X=10 Y=5**
- However this test case won't give you 100% decision coverage as the FALSE condition of the IF statement is not exercised.
- In order to achieve 100% decision coverage we need to exercise the FALSE condition of the IF statement which will be covered when X is less than Y.



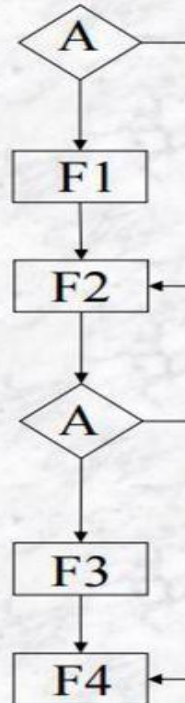
:WHITEBOX TESTING:

- So the final TEST SET for 100% decision coverage will be:
- **TEST CASE 1: X=10, Y=5**
TEST CASE 2: X=2, Y=10
- Note: 100% decision coverage guarantees 100% statement coverage but 100% statement coverage does not guarantee 100% decision coverage.
- **3. Path coverage:**
- The basis path method enables the test-case designer to derive a logical
- complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.



:WHITEBOX TESTING:

```
if (A)
    F1 ();
F2 ();
if (A)
    F3 ();
F4 ();
```



Paths:

A-F1-F2-A-F3-F4
A-F2-A-F3-F4
A-F1-F2-A-F4
A-F2-A-F4

Problem: only two
are feasible

A=T
A=F

[Figure of Path coverage Example]



White-Box Testing Techniques

41

- **Basis path testing**
- Basis path testing is a technique of selecting the paths in the [control flow graph](#), that provide a basis set of execution paths through the program or module. Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure.
- The basis path method (proposed by McCabe) makes it possible to obtain a measure of the complexity of a procedural design, and to use this measure as a guide for the definition of a series of basic paths of execution, designing test cases that guarantee that each path is run at least once.



White-Box Testing Techniques

42

- **Basis path testing**
- To design test cases using this technique, four steps are followed :
- Construct the Control Flow Graph
- Compute the Cyclomatic Complexity of the Graph
- Identify the Independent Paths
- Design Test cases from Independent Paths



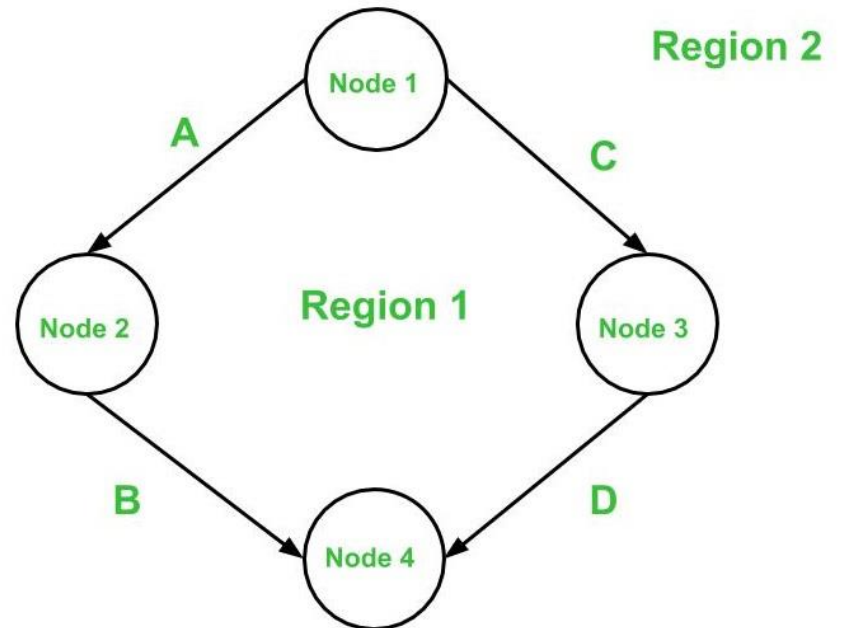
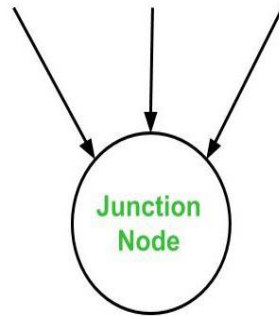
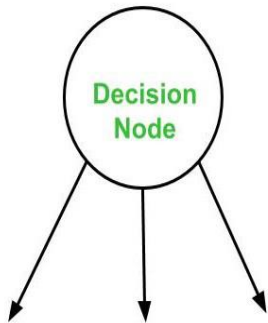
Construct the Control Flow Graph

43

- **1. Control Flow Graph –**
- A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have :
- **Junction Node** – a node with more than one arrow entering it.
- **Decision Node** – a node with more than one arrow leaving it.
- **Region** – area bounded by edges and nodes (area outside the graph is also counted as a region.).

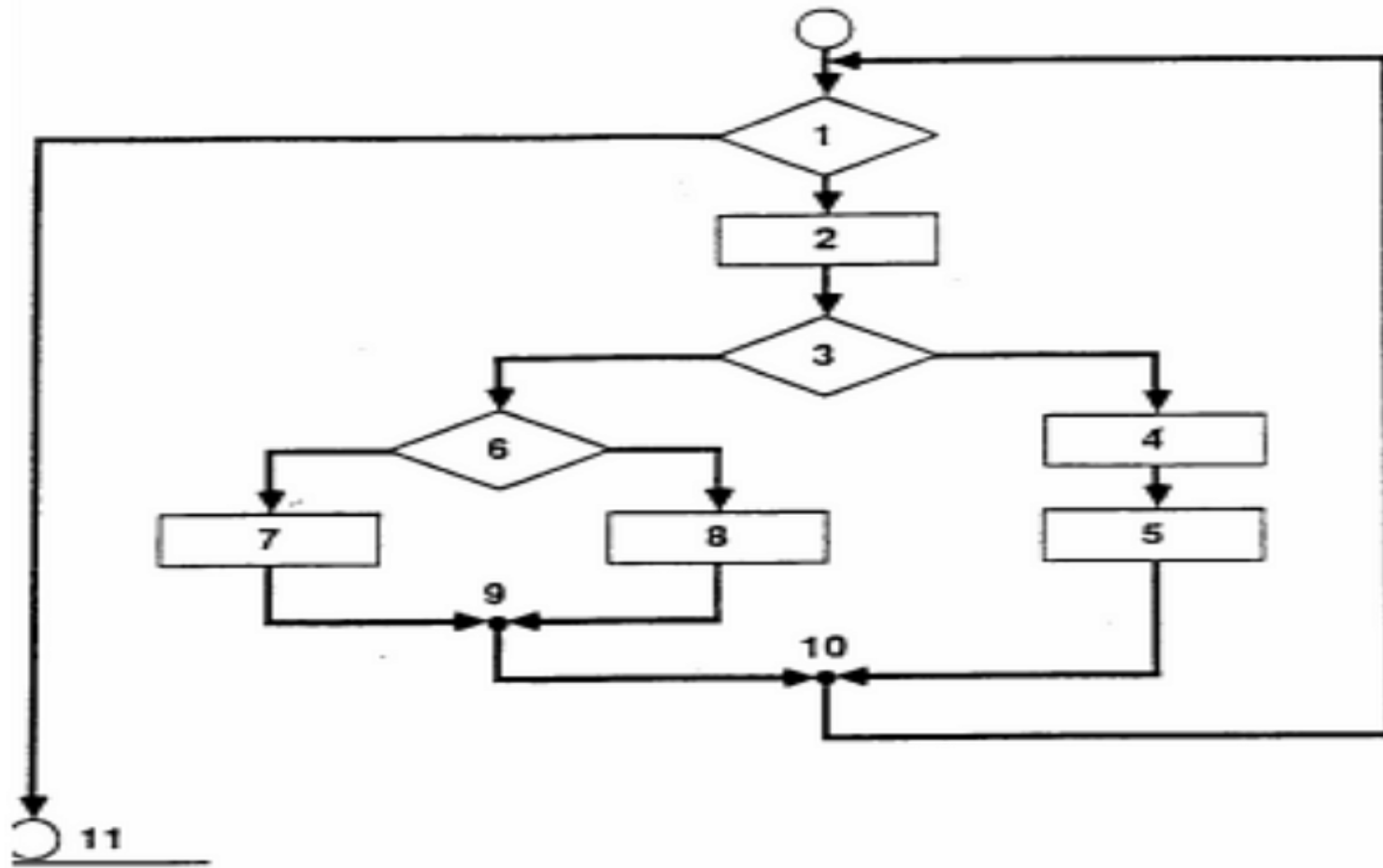
Construct the Control Flow Graph

44



Flow chart representing the control structure of the program.

45



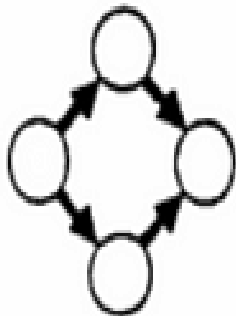
Notation of the flow graph or program graph

46

Sequence



If



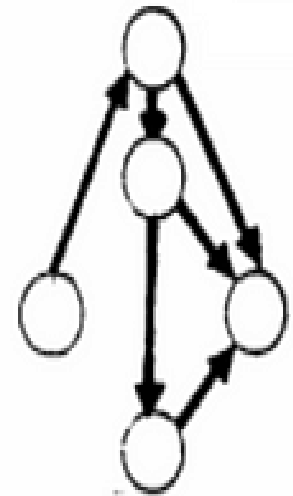
While



Until



Case





Cyclomatic complexity

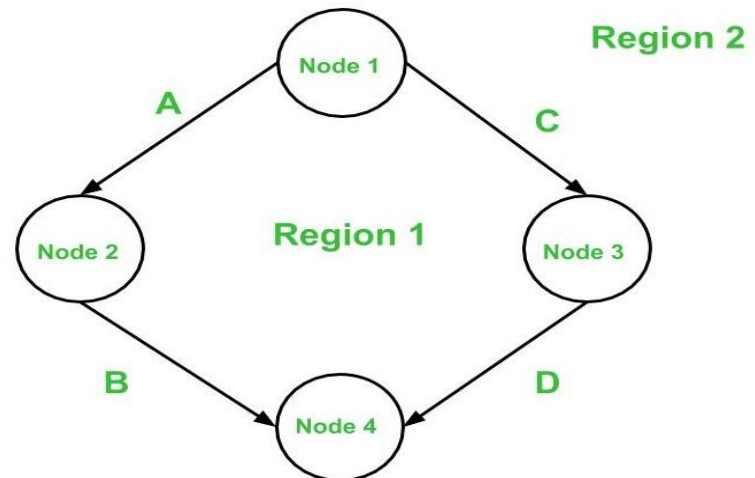
47

- It is a measure that provides an idea of the logical complexity of a program.
- Cyclomatic complexity coincides with the number of regions of the flow graph.
- The cyclomatic complexity, $V(G)$, of a flow graph G , is defined as $V(G) = \text{Edges} - \text{Nodes} + 2$
- The cyclomatic complexity, $V(G)$, of a flow graph G , is also defined as $V(G) = \text{predicate nodes} + 1$
- The cyclomatic complexity, $V(G)$, of a flow graph G , is also defined as $V(G) = \text{No. of internal regions} + 1$

Cyclomatic complexity

48

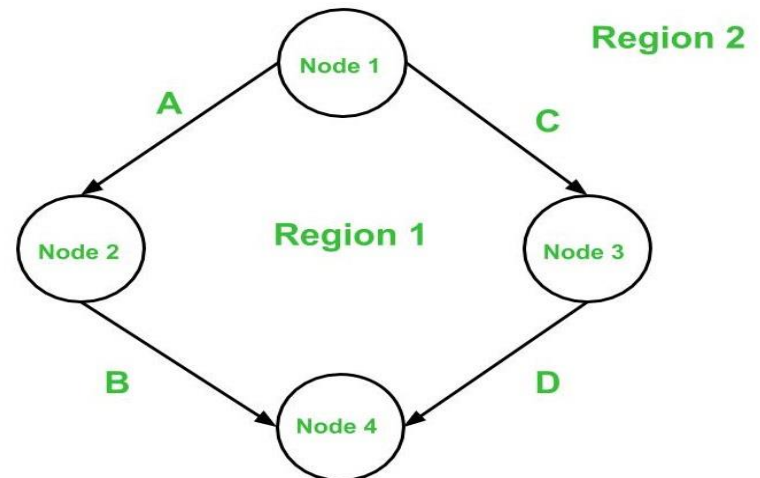
- The cyclomatic complexity $V(G)$ is said to be a measure of the logical complexity of a program. It can be calculated using three different formulae :
- **1. Formula based on edges and nodes :**
 - **$V(G) = e - n + 2 * P$**
- Where, e is number of edges, n is number of vertices, P is number of connected components. For example, consider first graph given above,
- where, $e = 4$, $n = 4$ and $p = 1$
- So,
- Cyclomatic complexity $V(G)$
- $= 4 - 4 + 2 * 1$
- $= 2$



Cyclomatic complexity

49

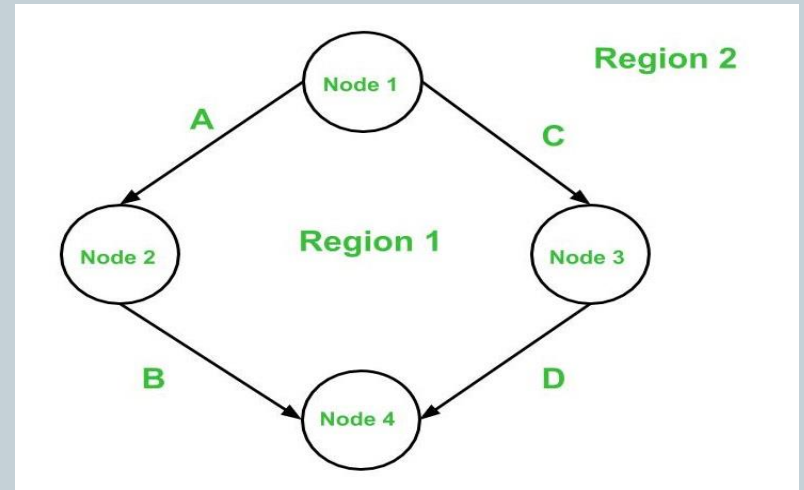
- **2. Formula based on predicate Nodes :**
- $V(G) = \pi + 1$
- where, π is number of predicate nodes,
- So,
- Cyclomatic Complexity $V(G)$
- $= 1 + 1$
- $= 2$



Cyclomatic complexity

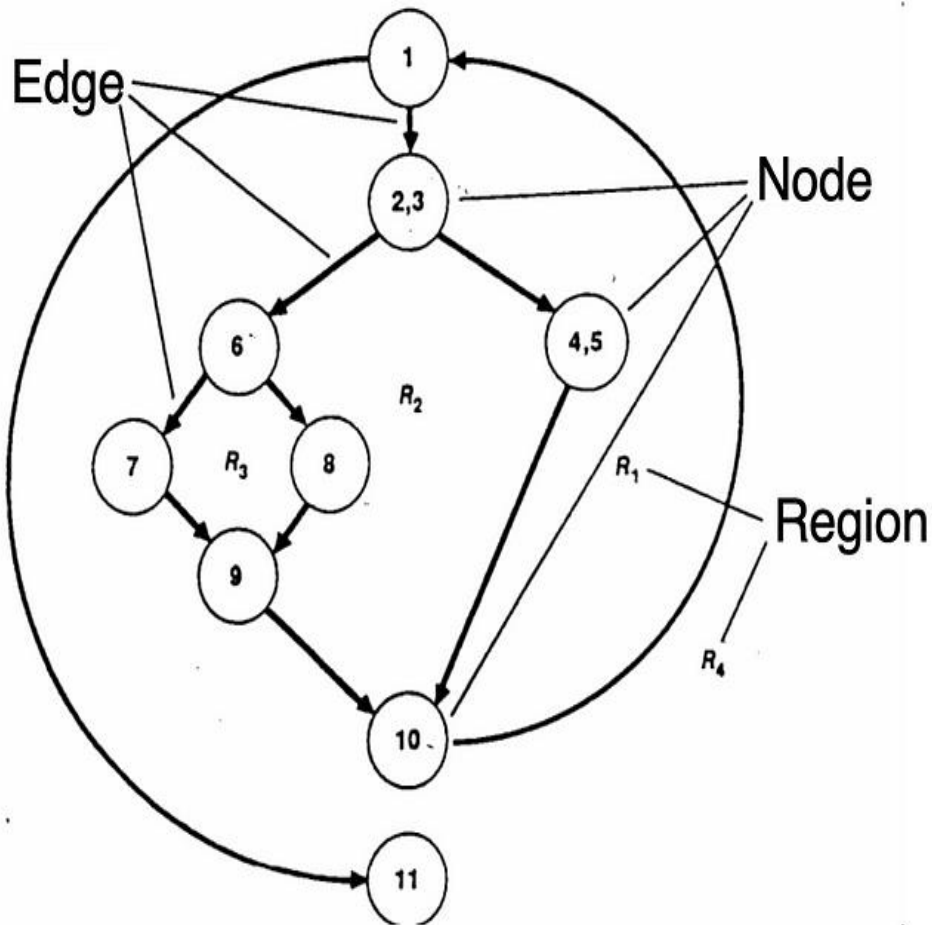
50

- **3. Formula based on Regions :**
- $V(G)$ = number of regions in the graph
- So,
- Cyclomatic complexity $V(G)$
- = 1 (for Region 1) + 1 (for Region 2)
- = 2



Cyclomatic complexity

51



From the flow graph of the figure, the cyclomatic complexity would be:

- Since the graph has four regions, $V(G) = 4$
- Since the graph has 11 edges and 9 nodes,
• $V(G) = 11 - 9 + 2 = 4$
- Since the graph has 3 predicate nodes,
• $V(G) = 3 + 1 = 4$



Cyclomatic complexity

52

- From the value of cyclomatic complexity we obtain the number of independent paths, which give us a limit value for the number of tests we have to design. In the example, the number of independent roads is 4, and the independent roads are:
- 1–11
- 1–2–3–4–5–10–1–11
- 1–2–3–6–7–9–10–1–11
- 1–2–3–6–8–9–10–1–11

Cyclomatic complexity

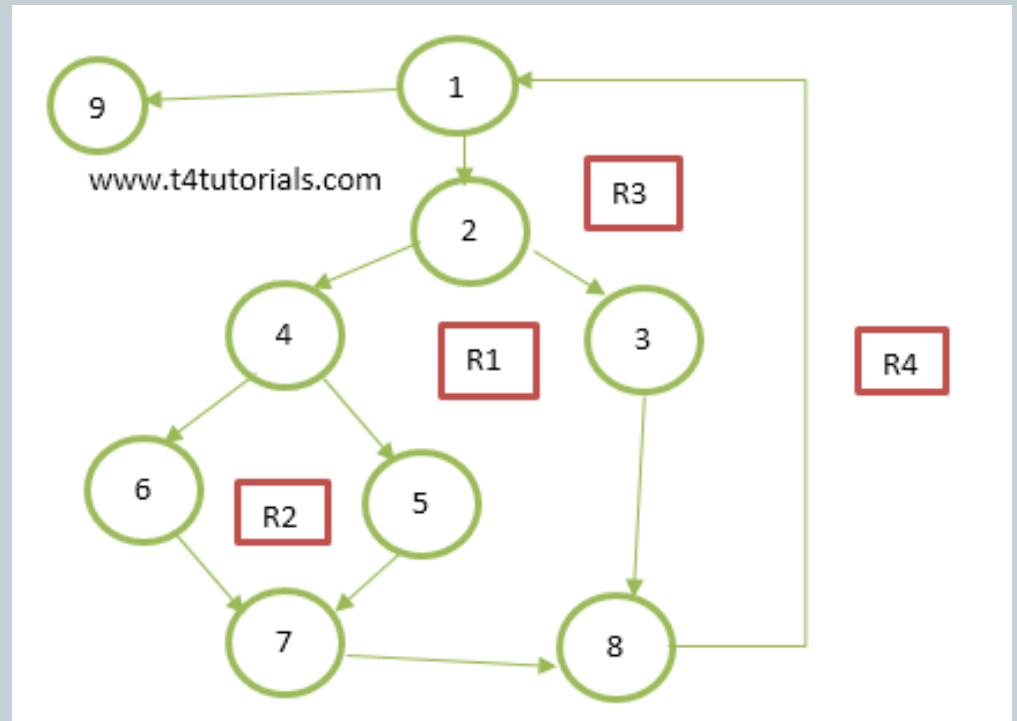
53

There are three methods;

1.Count the number of regions on the graph: 4

2.No of edges – no. of nodes + 2: $11-9+2=4$

3.No. of predicates + 1
: $3+1=4$





Independent Paths

54

- An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined. The cyclomatic complexity gives the number of independent paths present in a flow graph. This is because the cyclomatic complexity is used as an upper-bound for the number of tests that should be executed in order to make sure that all the statements in the program have been executed at least once. Consider first graph given above here the independent paths would be 2 because number of independent paths is equal to the cyclomatic complexity. So, the independent paths in above first given graph :



Independent Paths

55

- Path 1:
- A \rightarrow B
- Path 2:
- C \rightarrow D
- **Note** – Independent paths are not unique. In other words, if for a graph the cyclomatic complexity comes out to be N, then there is a possibility of obtaining two different sets of paths which are independent in nature. **Design Test Cases** : Finally, after obtaining the independent paths, test cases can be designed where each test case represents one or more independent paths.



Control Structure Testing: condition and loop testing

56

- **1. Condition Testing :** Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors. The errors present in logical conditions can be incorrect boolean operators, missing parenthesis in a booleans expression, error in relational operators, arithmetic expressions, and so on. The common types of logical conditions that are tested using condition testing are-
 - A relation expression, like $E1 \text{ op } E2$ where 'E1' and 'E2' are arithmetic expressions and 'OP' is an operator.
 - A simple condition like any relational expression preceded by a NOT (\sim) operator. For example, $(\sim E1)$ where 'E1' is an arithmetic expression and 'a' denotes NOT operator.



Control Structure Testing: condition and loop testing

57

- A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis. For example, $(E1 \ \& \ E2) | (E2 \ \& \ E3)$ where $E1, E2, E3$ denote arithmetic expression and ' $\&$ ' and ' $|$ ' denote AND or OR operators.
- A Boolean expression consists of operands and a Boolean operator like 'AND', OR, NOT. For example, ' $A | B$ ' is a Boolean expression where ' A ' and ' B ' denote operands and $|$ denotes OR operator.



Control Structure Testing: condition and loop testing

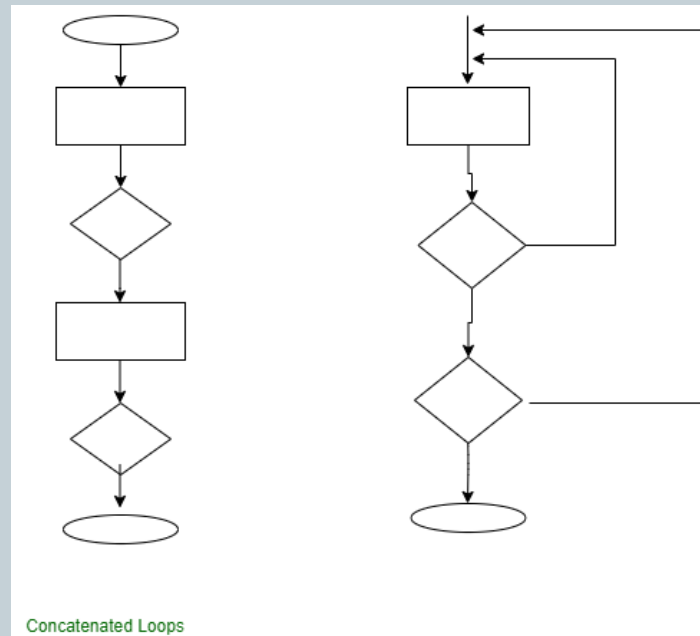
58

- **2. Loop Testing** : Loop testing is actually a white box testing technique. It specifically focuses on the validity of loop construction. Following are the types of loops.
- **A. Simple Loop** – The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n .
 - Skip the entire loop.
 - Traverse the loop only once.
 - Traverse the loop two times.
 - Make p passes through the loop where $p < n$.
 - Traverse the loop $n-1$, n , $n+1$ times.

Control Structure Testing: condition and loop testing

59

B. Concatenated Loops – If loops are not dependent on each other, concatenated loops can be tested using the approach used in simple loops. If the loops are interdependent, the steps are followed in nested loops.





Control Structure Testing: condition and loop testing

60

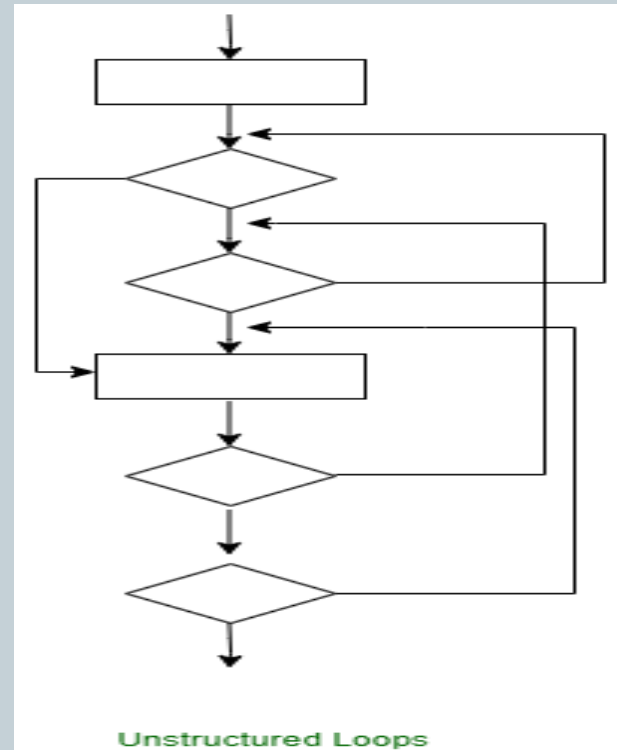
C. Nested Loops – Loops within loops are called as nested loops. when testing nested loops, the number of tested increases as level nesting increases. The following steps for testing nested loops are as follows-

- Start with inner loop. set all other loops to minimum values.
- Conduct simple loop testing on inner loop.
- Work outwards.
- Continue until all loops tested.

Control Structure Testing: condition and loop testing

61

- **D. Unstructured loops** – This type of loops should be redesigned, whenever possible, to reflect the use of unstructured the structured programming



:COMPARISON OF BLACKBOX TESTING AND WHITEBOX TESTING:

#	Black Box Testing	White Box Testing
1	Black box testing is the <u>Software testing method</u> which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.



COMPARISON OF BLACKBOX TESTING AND WHITEBOX TESTING:

#	Black Box Testing	White Box Testing
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.

THANK YOU