White-Box Software Testing

Speaker: Jerry Gao Ph.D.

Computer Engineering Department
San Jose State University

email: jerry.gao@sjsu.edu
URL: http://www.engr.sjsu.edu/gaojerry

Presentation Outline

- *What is White-Box Testing?*
  - *Testing Focuses*
  - *Who Perform White-Box Testing?*
- *Basis-Path Program Testing*
- *Branch-Based Program Testing*
- *Data Flow Program Testing*
- *Syntax-Based Program Testing*
- *State-Based Program Testing*
- *Testing Coverage*

What is White-Based Software Testing?

What is white-box software testing?

--> White-box testing, also known as glass-box testing.

Basic idea is to test a program based on the structure of a program.

What do you need for white-box testing?

- A white-box testing model and test criteria

- A white-box test design and generation method

- Program source code

White-box testing methods can be classified into:

- ***Traditional white-box testing methods***
- ***Object-oriented white-box testing methods***
- ***Component-oriented white-box testing methods***

White-Box Testing Objectives

The Major objective of white-box testing is to focus on internal program structure, and discover all internal program errors.

The major testing focuses:

- ***Program structures***
  - ***Program statements and branches***
  - ***Various kinds of program paths***

- ***Program internal logic and data structures***
- ***Program internal behaviors and states.***

Traditional White-Based Software Testing Methods

Test Model: control program chart (graph)

Test case design:

Various white-box testing methods generate test cases based on on a given control program graph for a program

The goal is to:

- Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions one their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Exercise all data define and use paths.

White-Box Software Testing Methods

Basic path testing (a white-box testing technique):

- *First proposed by TomMcCabe [MCC76].*
- *Can be used to derive a logical complexity measure for a procedure design.*
- *Used as a guide for defining a basis set of execution path.*
- *Guarantee to execute every statement in the program at least one time.*

Branch Testing:

Exercise predicate nodes of a program flow graph to make sure that each predicate node has been exercised at least once.

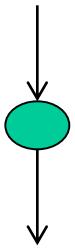
Loop Testing:

Exercise loops of a program to make sure that the inside and outside of loop body are executed.

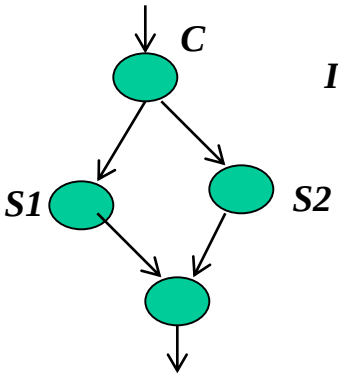
State-Based Testing:

Basic idea is to use a finite state machine as a test model to check the state behaviors of a program process.

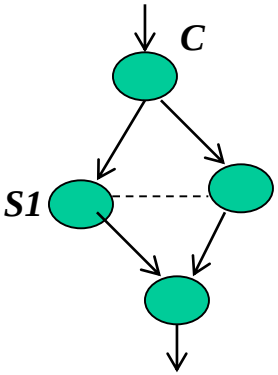
Program Flow Graph (Control Flow Diagram)



Sequential statement block

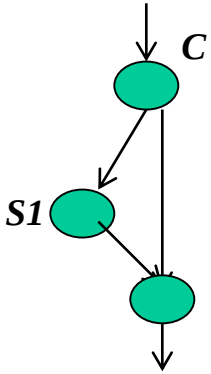


If C Then S1 else S2;



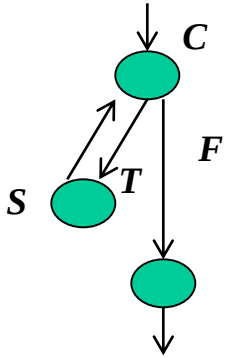
Case C of

L1: S1;
L2: S2;
…
Kn: Sn;
end;

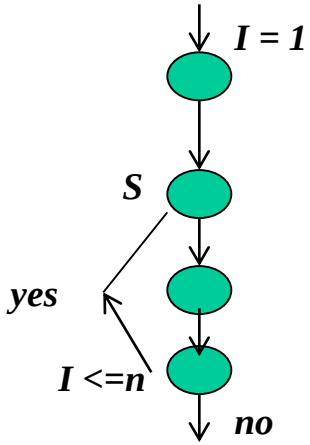


If C Then S1;

Program Flow Graph (Control Flow Diagram)

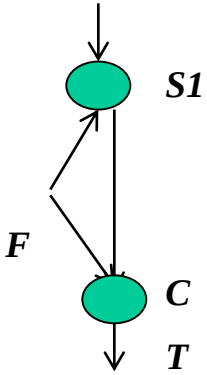


While C do S;



For loop:

for I = 1 to n do S;



Do loop:

do S1 until C;

Cyclomatic Complexity

Cyclomatic complexity is a software metric
-> provides a quantitative measure of the global complexity of a program.
When this metric is used in the context of the basis path testing, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program.

Three ways to compute cyclomatic complexity:

- The number of regions of the flow graph correspond to the cyclomatic complexity.

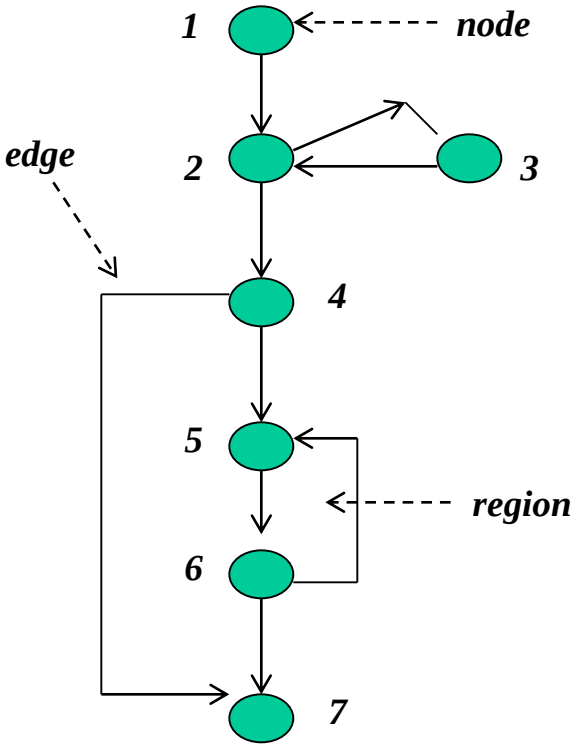
- Cyclomatic complexity, V(G), for a flow graph G is defined as

$$V(G) = E - N + 2$$

where E is the number of flow graph edges and N is the number of flow graph nodes.

- Cyclomatic complexity, V(G) = P + 1
where P is the number of predicate nodes contained in the flow graph G.

An Example



No. of regions = 4
(considering the universal region)

No. of edges = 9

No. of nodes = 7

No. of predicate nodes = 3

V(G) = P + 1 = 3 + 1 = 4

V(G) = E – N + 2 = 9 - 7 + 2 = 4

Deriving Test Cases

Step 1 : *Using the design or code as a foundation, draw a corresponding flow graph.*

Step 2: *Determine the cyclomatic complexity of the resultant flow graph.*

Step 3: *Determine a minimum basis set of linearly independent paths.*

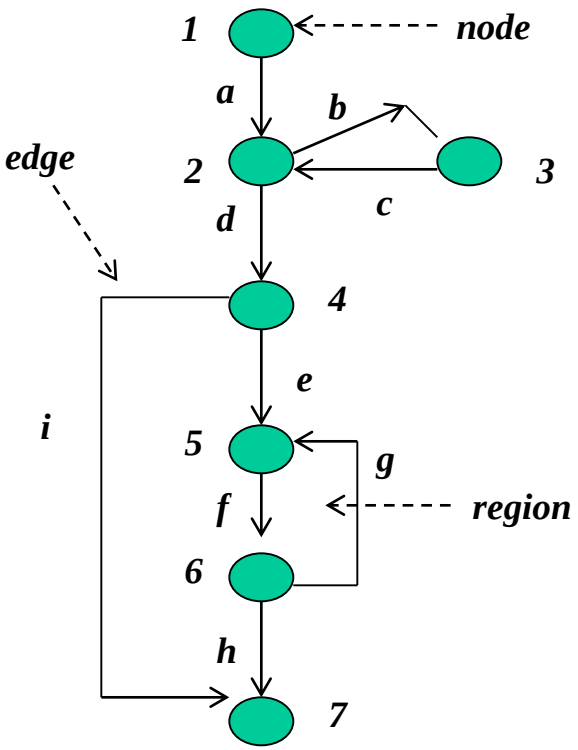
For example,

path 1: 1-2-4-5-6-7
path 2: 1-2-4-7
path 3: 1-2-3-2-4-5-6-7
path 4: 1-2-4-5-6-5-6-7

Step 4: *Prepare test cases that will force execution of each path in the basis set.*

Step 5: *Run the test cases and check their results*

An Example



	1	2	3	4	5	6	7
1		a					
2			b	d			
3		c					
4					e		i
5						f	
6					g		h
7							

	1	2	3	4	5	6	7	
1		1						1 - 1 = 0
2			1	1				2 - 1 =1
3		1						1 -1 = 0
4					1		1	2 - 1 =1
5						1		1 -1 = 0
6					1		1	2 -1 = 1
7								

---

3 +1 = 4

Data Flow Software Testing

Major Objective:

Focus data value definitions and its uses in a program control flow charts.

Major purpose:

Data flow test criteria is developed to improve the data test coverage of a program.

Basic idea:

For a given Var has been assigned the correct value at some point in the program if no test data cause the execution of a path from the assignment point to a usage point in a program flow chart.

Basic Test Model: ***Program Control Flow Chart (Graph)***

Test Criteria: ***Various data flow test criteria***
They are stronger than branch testing but weaker than path testing.

Data Flow Software Testing

Basic Definitions:

V = ***the set of variables.***
N = ***the set of nodes***
E = ***the set of edges***

def(i) = ***{ x in V | x has a global definition in block I}***

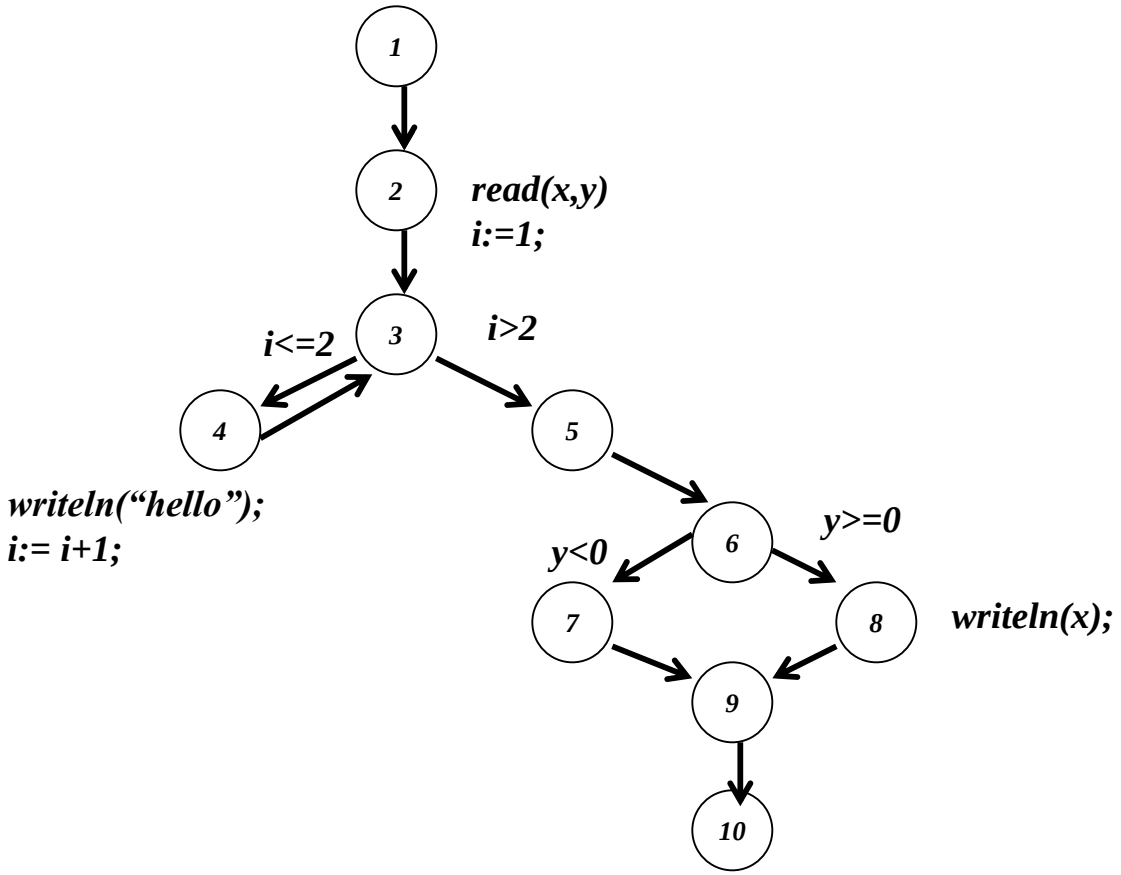
c-use(i) = ***{ x in V | x has a global c-use in block i }***

p-use(i,j) = ***{ x in V } x has a p-use in edge (i,j) }***

dcu(x, j) = ***{ j in N | x in c-use(j) and there is a def-clear path wrt x from i to j }***

dpu(x, j) = ***{(j,k) in E | x in p-use(j,k) and there is a def-clear path wrt x from i to (j,k) }***

Data Flow Software Testing

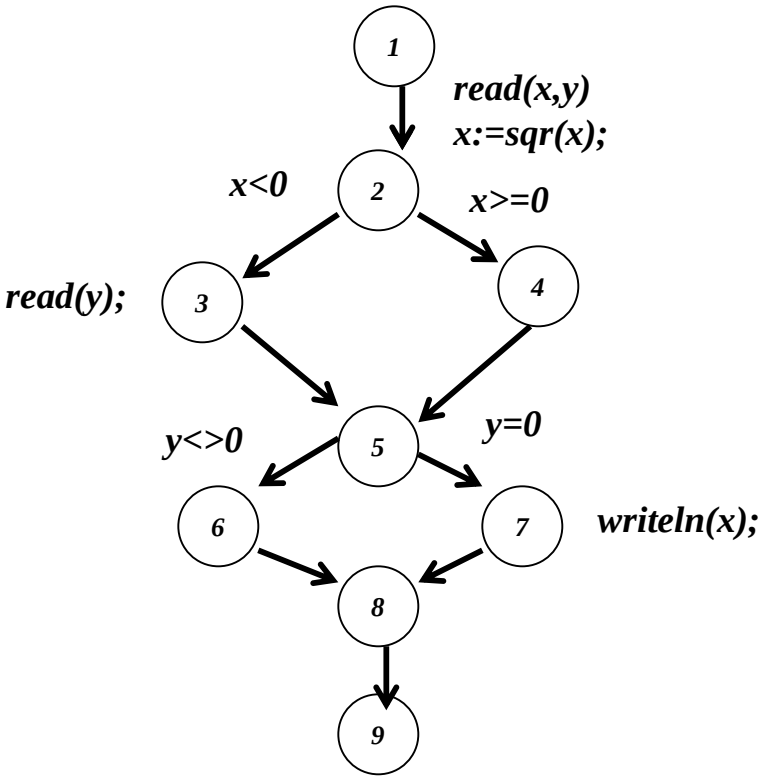


Data Flow Software Testing

Path	*With Respect to*	*Executable*
(1,2)	*input ^*	*Yes*
(2,3,4)	*i*	*Yes*
(2,3,5)	*i*	*No*
(4,3,4)	*i*	*Yes*
(4,3,5)	*i*	*Yes*
(2,3,5,6,7,9,10)	*input ^*	*No*
(2,3,5,6,8,9,10)	*input ^*	*No*

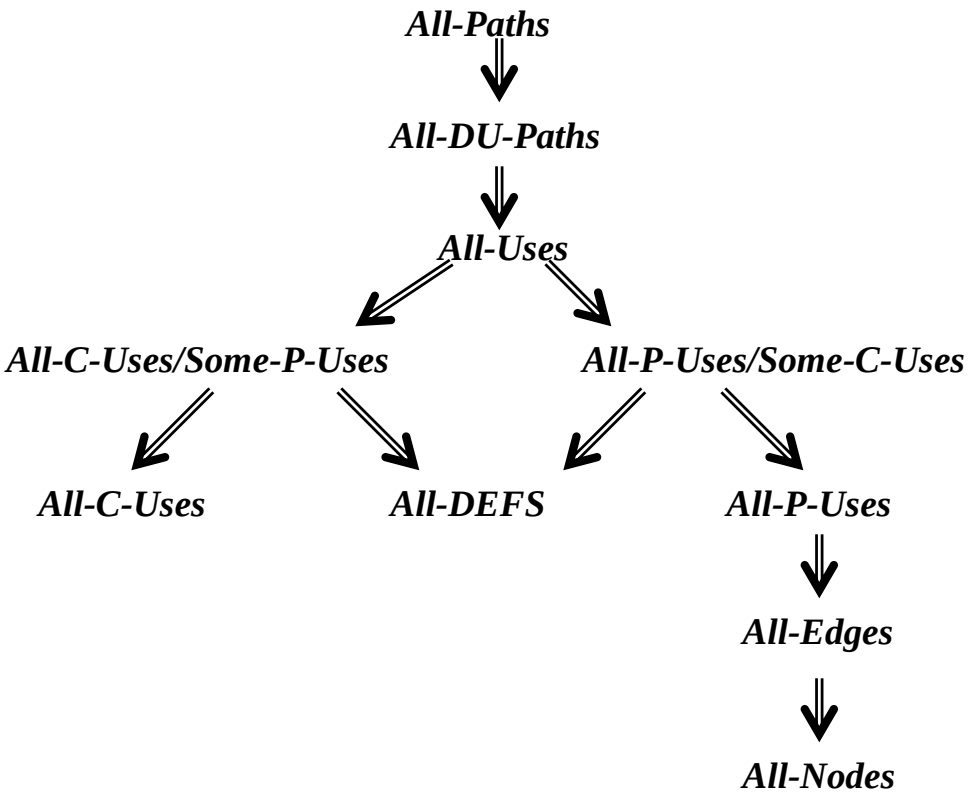
Program demonstrating that (all-du-paths)* fails to include the other criteria

Data Flow Software Testing



Program demonstrating that (all-p-uses) fails to include (all-edges)**

Relationships among the Data Flow Software Testing Criteria



Branch Testing

What is branch testing for software? (known as decision testing in programs)

Design test cases to exercise each branch-based path in a program flow graph.

Test focus is on: **Each branch in a program structure**

Test coverage: **For each decision in a program structure, each branch must be exercised at least once.**

Test model: **Program control flow graph**

Limitation:
- **Treats a compound conditional as a single *statement***
- **Ignores implicit paths that result from compound conditionals.**

White-Box : Branch Testing

- Branch testing == decision testing
- Execute every branch of a program :

  each possible outcome of each decision occurs at least once
- Example:
  - IF b THEN s1 ELSE s2
  - IF b THEN s1; s2
  - CASE x OF

    1 : ....

    2 : ....

    3 : ....

Branch Testing Example

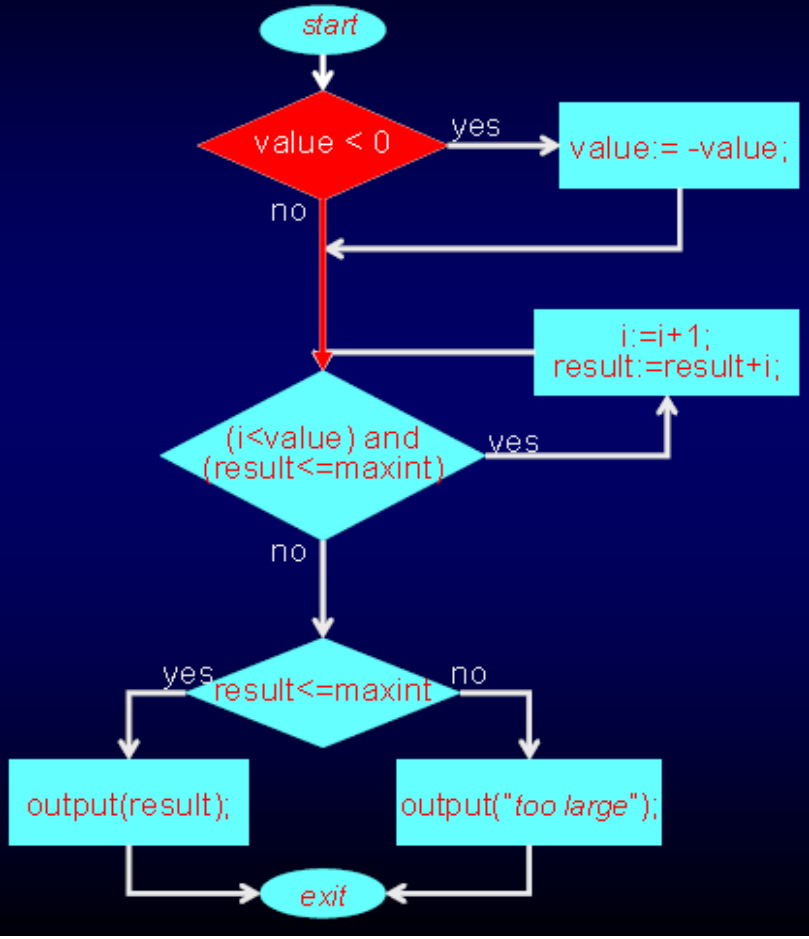
```
**  Branch Testing Exercise
 *  Create test cases using branch test method for this program
 */
1.    declare Length as integer
2.    declare Count as integer
3.   READ Length;
4.   READ Count;
5.   WHILE (Count <= 6) LOOP
6.       IF (Length >= 100) THEN
7.             Length = Length - 2;
8.       ELSE
9.          Length = Count * Length;
10.      END IF
11.      Count = Count + 1;
12.  END Loop;
13       PRINT Length;
14.            End of program
```

Branch	Possible outcome	T1	T2	T3
Count <=6	T	X	X	
	F			X
Length >=100	T	X		
	F		X	

Branch Testing Example

Test Case #	Input values - Count	Input values - Length	Expected Outcomes	Actual Outcomes
T1	5	101	594	
T2	5	99	493	
T3	7	99	99	

Example : Branch Testing



Tests for complete statement coverage:

maxint	*value*
10	-1
0	-1

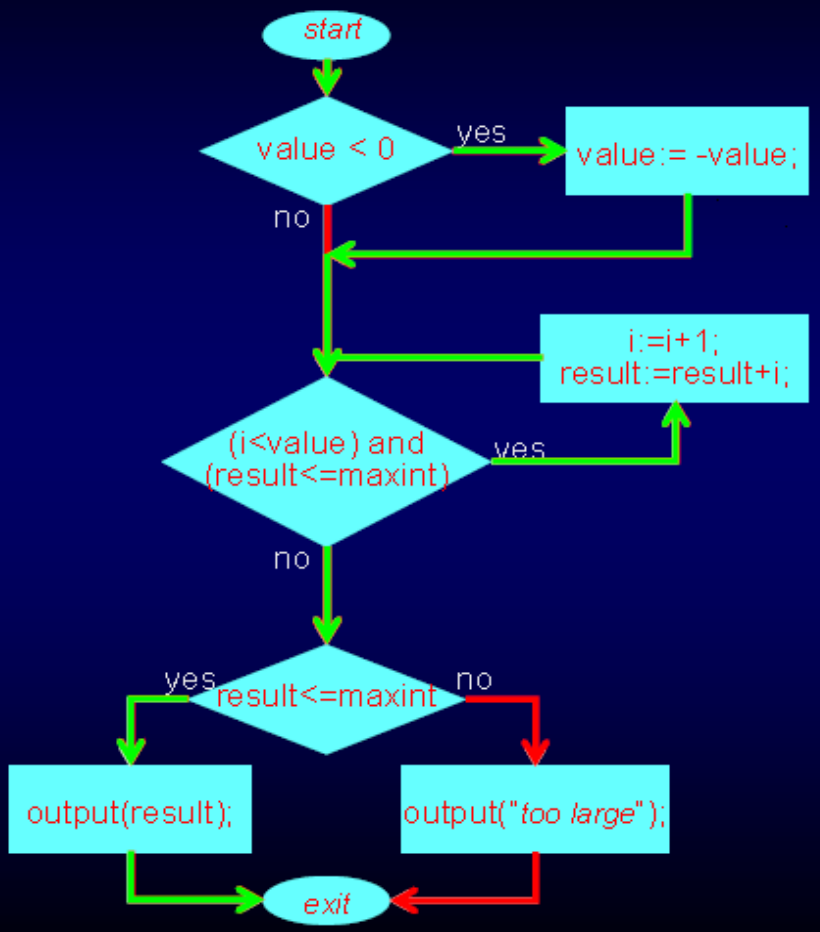
is not sufficient for branch coverage;

Take:

maxint	*value*
10	3
0	-1

for complete branch coverage

Example : Branch Testing



maxint	*value*
-1	*-1*
10	*3*

But:

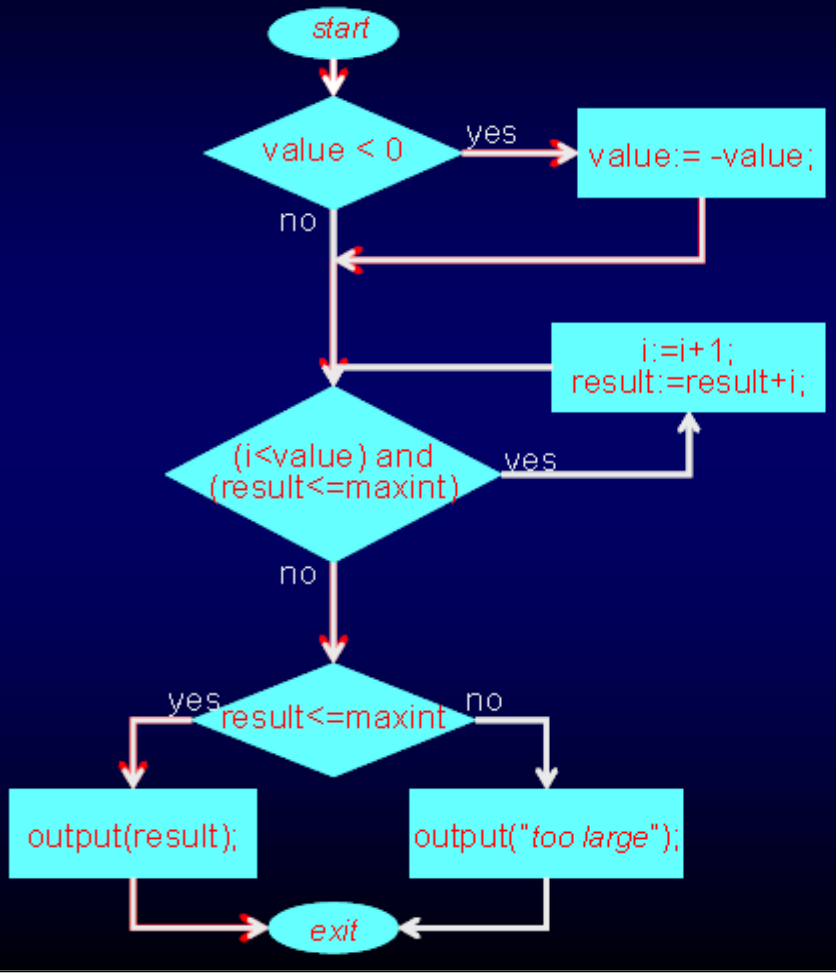
No green path !

Needed :

Combination of decisions

10 -3

Example : Branch Testing



Sometimes there are infeasible paths (infeasible combinations of conditions)

Condition Testing For Software

What is condition testing software?

Design test cases to exercise each logic condition in a program.

Test focus is on: incorrect logic in Boolean expressions of branch nodes.

- **Boolean variable errors**
- **Boolean parenthesis errors**
- **Boolean operator errors**
- **Relational operator errors**
- **Arithmetic expression errors**

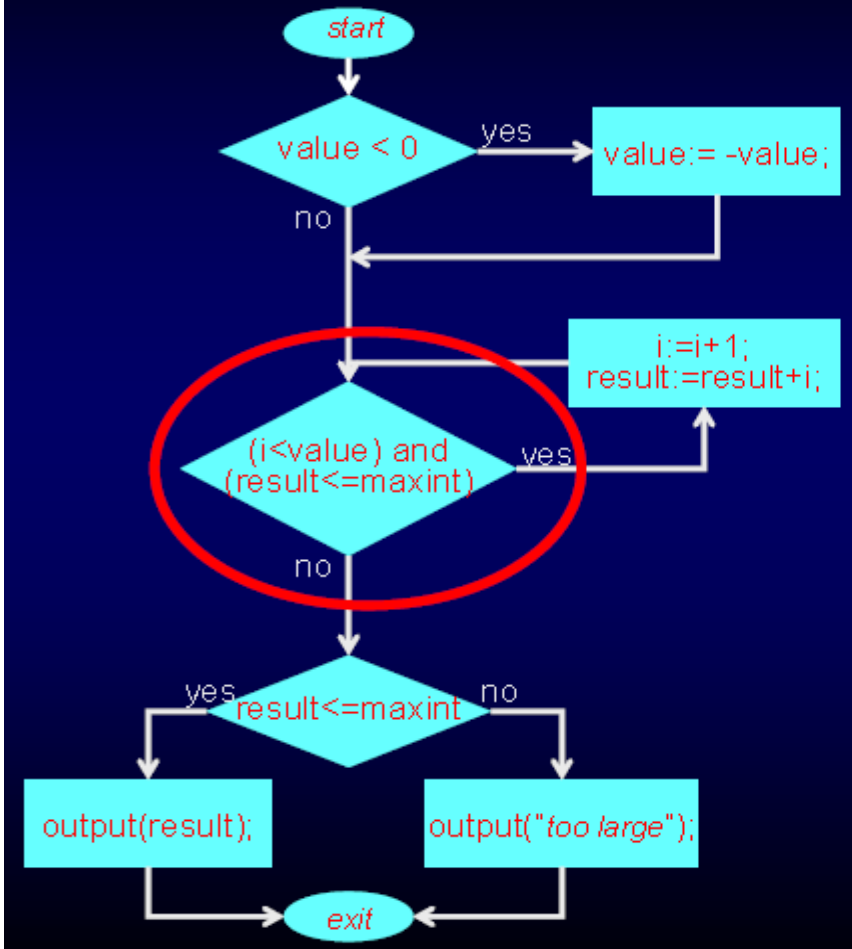
Test coverage:

For a compound condition C, the True and False branches of C and every simple condition in C need to be executed at least once.

White-Box : Condition Testing

- Design test cases such that each possible outcome of each condition in each decision occurs at least once
- Example:
  - decision (i < value) AND (result <= maxint)
    consists of two conditions : (i < value) AND (result <= maxint)
    test cases should be designed such that each gets value true and false at least once

Example : Condition Testing



(i = result = 0) :

maxint value	i<value	result<=maxint
-1	1	true	false
1	0	false	true

gives condition coverage for all conditions

But it does not preserve decision coverage

⇓

always take care that condition coverage preserves decision coverage :

decision / condition coverage

White-Box : Multiple Condition Testing

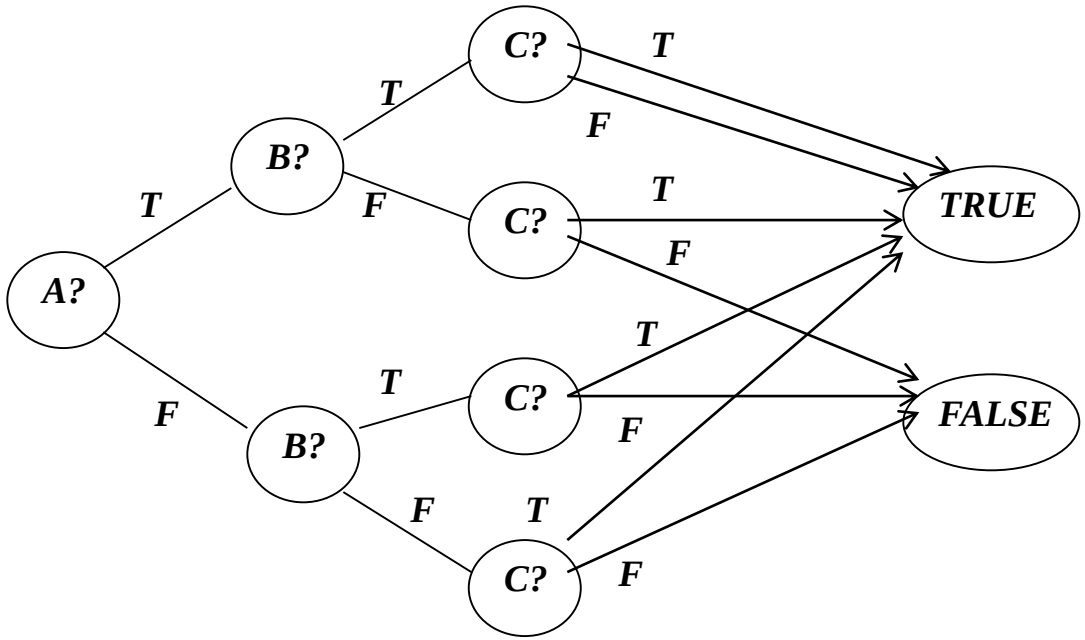
- Design test cases for each combination of conditions
- Example:
  - (i < value) (result <= maxint)

(i < value)	(result <= maxint)
false	false
false	true
true	false
true	true

- Implies decision-, condition-, decision/condition coverage
- But : exponential blow-up
- Again : some combinations may be infeasible

Condition Testing for Software

Condition: (A AND B Or C)



Condition Testing For Software

A	*B*	*C*	*A&B or C*
TRUE	*TRUE*	*TRUE*	*TRUE*
TRUE	*TRUE*	*FALSE*	*TRUE*
TRUE	*FALSE*	*TRUE*	*TRUE*
TRUE	*FALSE*	*FALSE*	*FALSE*
FALSE	*TRUE*	*TRUE*	*TRUE*
FALSE	*TRUE*	*FALSE*	*FALSE*
FALSE	*FALSE*	*TRUE*	*TRUE*
FALSE	*FALSE*	*FALSE*	*FALSE*

Loop Testing For Software

Simple Loops, where *n is the maximum number of allowable passes through the loop.*

- Skip loop entirely
- Only one pass through loop
- Two passes through loop
- m passes through loop where m<n.
- (n-1), n, and (n+1) passes through the loop.

Nested Loops

- 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.

Concatenated Loops

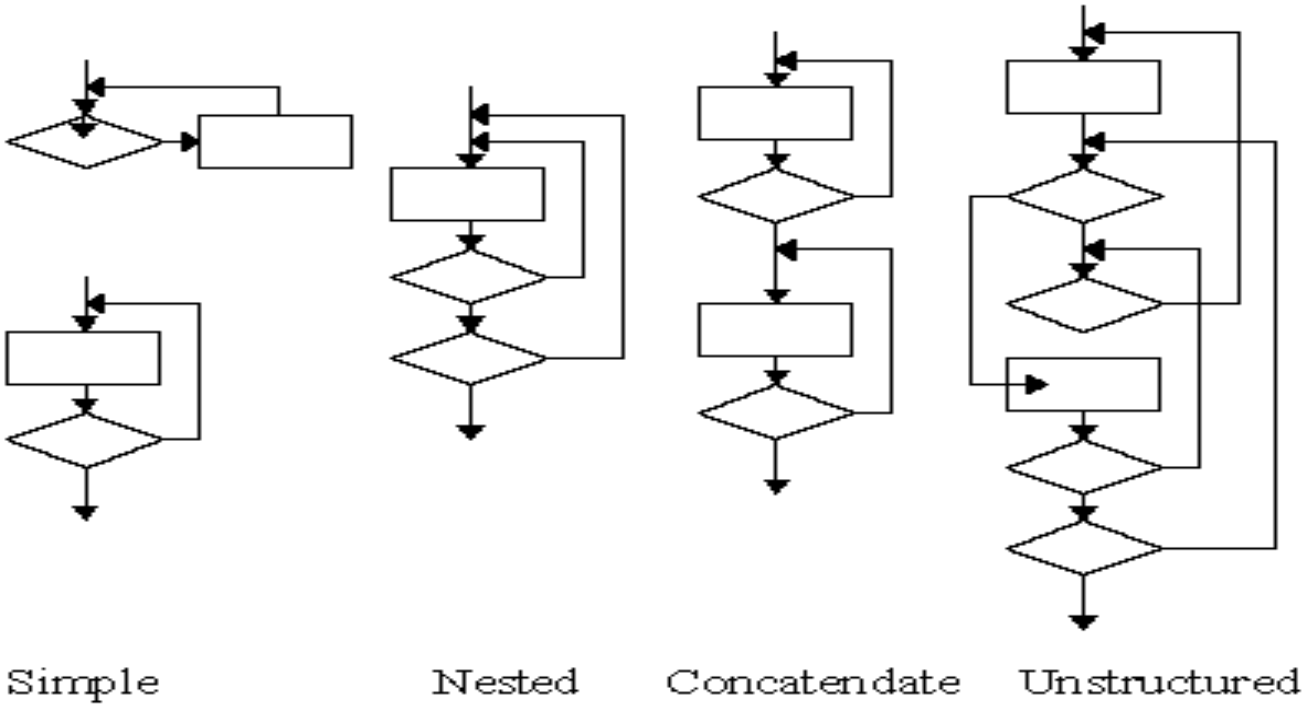
- If independent loops, use simple loop testing.
- If dependent, treat as nested loops.

Unstructured loops

- Don't test - redesign.

Loop Testing For Software

Examples:



Syntax-Based Software Testing

What is syntax testing?

Syntax testing is a powerful software testing technique for testing command-driven software and similar applications. There is a number of commercial tools.

Test model:

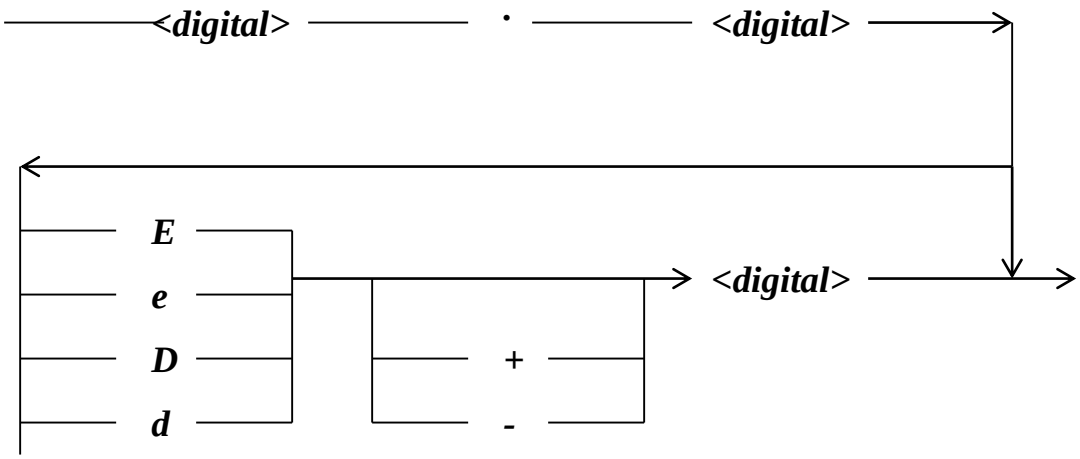
BNF-based graph is used as a test model to generate tests.

Test coverage:

- Node coverage - not very useful
- Link coverage - very important due to loops.

Syntax-Based Software Testing

Syntax Graph for <real_number>:



Syntax-Based Software Testing

Dirty Syntax Testing:

It includes twofold: ***a) to exercise a good sample of single syntactic errors in all commands in an attempt to break the software.***

b) to force every diagnostic message appropriate to that command to be executed.

Example test cases:

Try to come out test cases without loops:

.05, ***1.,*** ***1.1e***

Try to come out test cases with loops:

12345678901.1, ***1.1234567890123,*** ***1.1e1234***

State-Based Software Testing

What is state-based software testing?

A software testing method in which a well-defined state machine (or a state graph) is used as a test model to check the state-based behavior of a program.

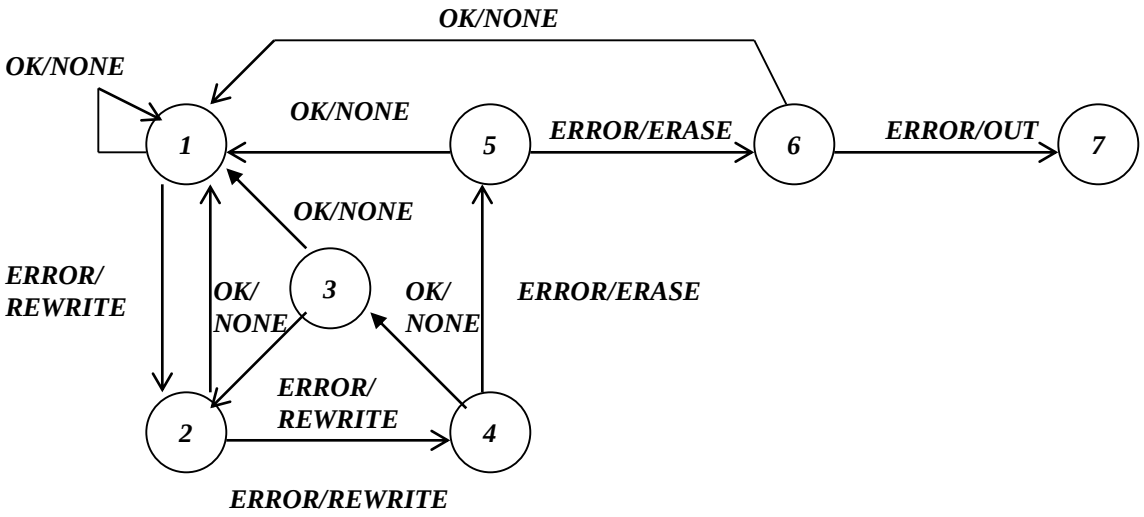
A state machine is used to define test cases and test criteria to uncover the incorrect state behavior of a program.

Coverage criteria:
- State node coverage
- Transition path coverage

Applications:
- Very useful to check protocol-based program communications.
- Good to check problems in state-driven or event-driven application programs.

State-Based Software Testing

A State Machine Example:



TAPE CONTROL RECOVERY ROUTINE STATE GRAPH

State-Based Software Testing

Making sure the correctness of your state graphs.

Checking the following problems in state machines:

- Dead lock state, a state can never be left.

- Non-reachable state, a state is not reachable from an initial state.

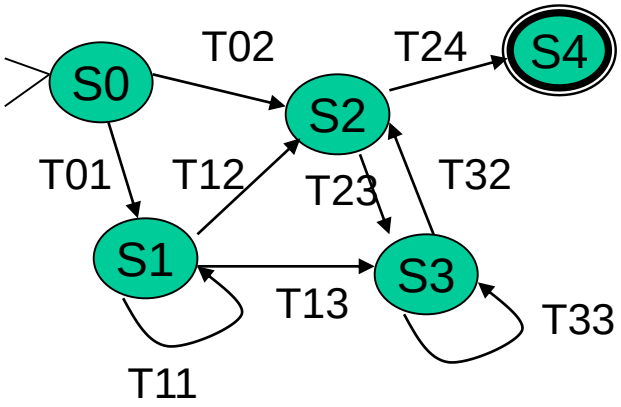
- Impossible and incorrect states.

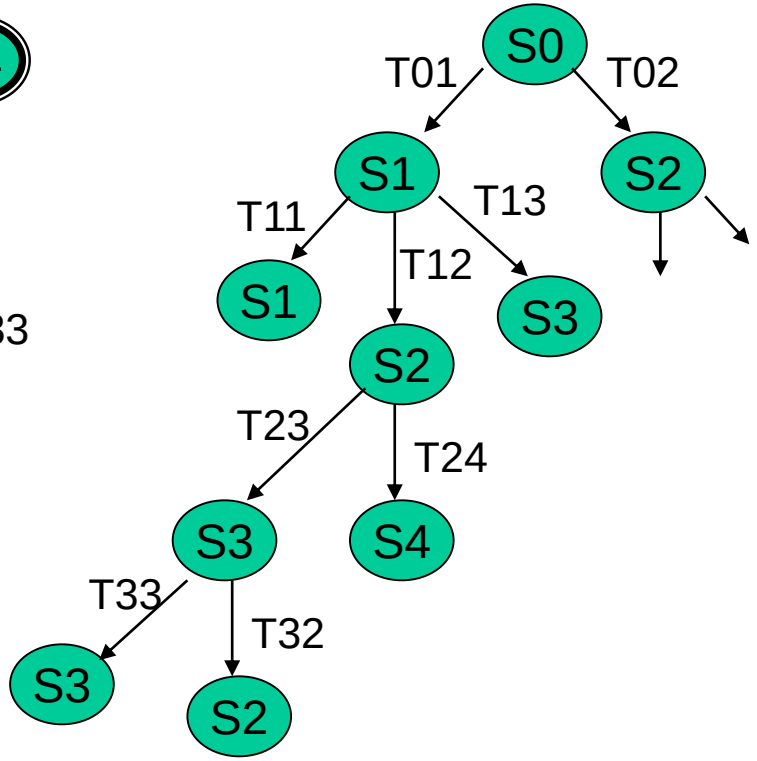
Some combinations of factors may appear to be impossible.

For example: (for a car)

GEAR	***R, N, 1, 2, 3, 4***	***= 6 factors***
DIRECTION	***FORWARD, REVERSE, STOPPED***	***= 3 factors***
ENGINE	***RUNNING, STOPPED***	***= 2 factors***
TRANSMISSION	***Okay, Broken***	***= 2 factors***
ENGINE	***Okay, Broken***	***= 2 factors***
Total		***= 144 states***

State-Based Test Generation





Test cases:

Level #1: S0->T01->S1
S0->T02>S2
Level #2: S0->T01->S1->T11->S1
S0->T01->S1->T12->S2
S0->T01->S1->T13->S3
…….

State-Based Software Testing

We can use the state-based testing method to uncover the following issues:

- ***Wrong number of states.***
- ***Wrong transition for a given state-input combination.***
- ***Wrong output for a given transition.***
- ***Pairs of states or sets of states that are inadvertently made equivalent (factor lost).***
- ***States or sets of states that are split to create inequivalent duplicates.***
- ***States or sets of states that have become dead.***
- ***States or sets of states that have become unreachable.***

State-Based Software Testing

Principles:

The starting point of state testing is:
• Define a set of covering input sequences that get back to the initial state when starting from the initial state.
•For each step in each input sequence, define the expected next state, the expected transition, and the expected output code.

A set of tests, then, consists of three sets of sequences:
1. Input sequences
2. Corresponding transitions of next state names.
3. Output sequences.

- ***Insist on a specification of transition and output for every combination of input and states.***
- ***Apply a minimum set of covering tests.***
- ***Instrument the transitions to capture the sequence of states and not just sequence of outputs.***
- ***Count the states.***

White-Box Software Testing Criteria

- Statement test coverage criteria
- Branch testing coverage criteria
- Condition test coverage criteria
- Basis-path test coverage criteria
- Loop test coverage criteria
- Dataflow testing coverage criteria
- Syntax testing coverage criteria
- State testing coverage criteria