**Software Testing:**

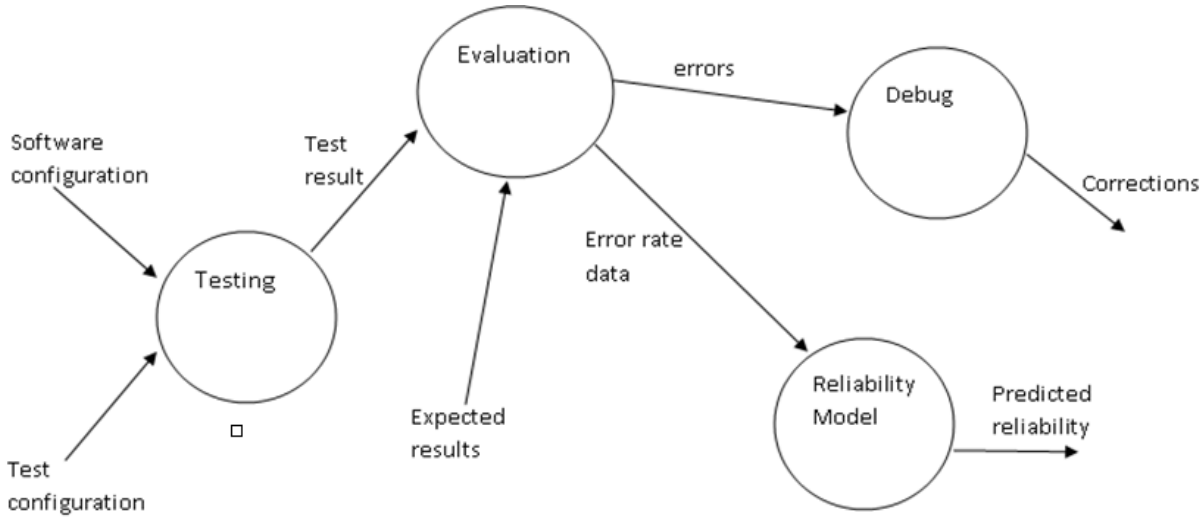
Checking correctness of software by executing the software on some inputs (test cases)

***Objectives:***

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.

**What is a "Good" Test?**

- A good test has a high probability of finding an error
- A good test is not redundant.
- A good test should be "best of breed"
- A good test should be neither too simple nor too complex

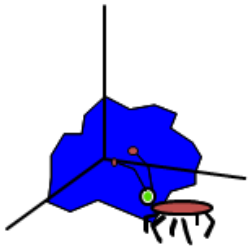



**Test information flow**

**Test Case Design**

"Bugs lurk in corners and congregate at boundaries ..."

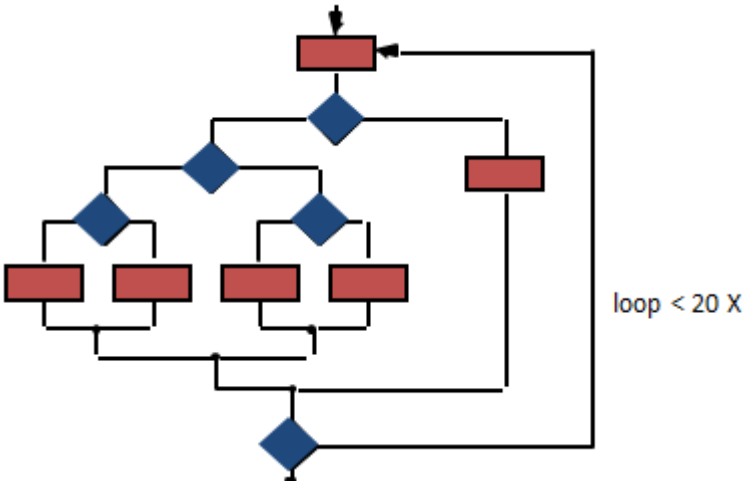
*Boris Beizer*



| | |
|---|---|
| *OBJECTIVE* | to uncover errors |
| *CRITERIA* | in a complete manner |
| *CONSTRAINT* | with a minimum of effort and time |

**Exhaustive Testing:**

Exhaustively testing which all possible input/output combinations is too expensive. It is impossible to test sw exhaustively, for each and every possible value. The number of test cases increases exponentially with the number of input/output variables. e.g., a program with 3 input integers with 16 bits, need $2^{16} * 2^{16} * 2^{16} = 2^{48}$ = 281.474.976.710.656 test cases.




There are $5^{20} = 10^{14}$ possible paths, if we execute one test per ms, it would take 3,170 years to test this program.

**Software Testing Strategies:**

1. White-box Testing: the goal is to ensure that all statements and conditions have been executed at least once.
2. Black box Testing: focus on the functional requirements of the software.

**White box testing:**

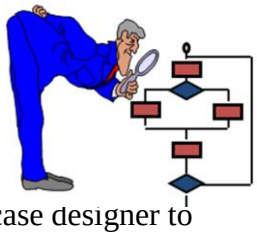
It is a test case design method that uses the control structure of the procedural design to derive test cases. Using white box testing methods, the software engineer can derive test cases that:

(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. And

(4) Exercise internal data structures to ensure their validity.



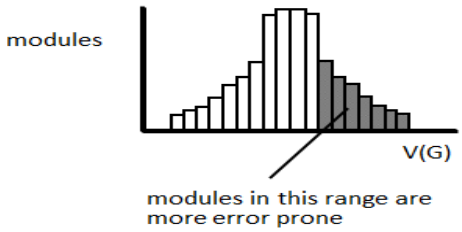
**Basis path testing**:

It is a white box testing technique. 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.

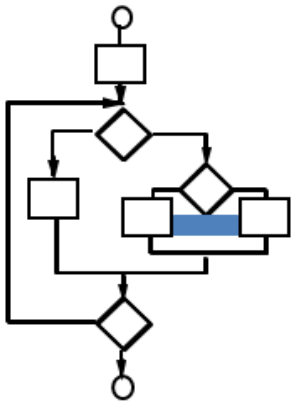
Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

Cyclomatic complexity V(G) is a software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of ***independent paths*** *in the* ***basis set*** of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

A number of industry studies have indicated that the higher V(G), the higher the probability of errors.




The Cyclomatic complexity V(G) could be calculated using the following:



**First**, we compute the cyclomatic complexity:

(1) number of simple decisions + 1

or

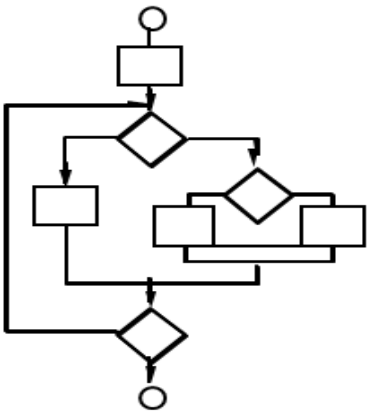
(2) number of enclosed areas + 1

In this case, V(G) = 4

or

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

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



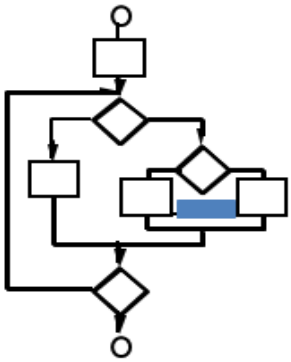
**Next**, we derive the independent paths:

Since V(G) = 4,
there are four paths

Path 1: 1,2,3,6,7,8
Path 2: 1,2,3,5,7,8
Path 3: 1,2,4,7,8
Path 4: 1,2,4,7,2,4,...7,8

Finally, we derive test cases to exercise these paths.

**Basis Path Testing Notes**



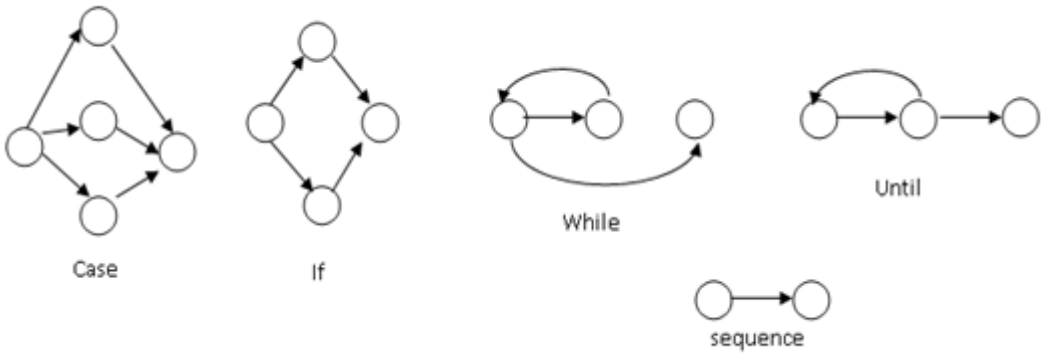
- you don't need a flow chart, but the picture will help when you trace program paths
- count each simple logical test, compound tests count as 2 or more
- basis path testing should be applied to critical modules

**Basis path method**

Before the basis path method can be introduced, a simple notation for the representation of control flow, called a **flow graph** (or **program graph**) must be introduced.

- The flow graph depicts logical control flow using the notation illustrated in next slide.
- Each structured construct has a corresponding flow graph symbol.

**Flow graph notation**

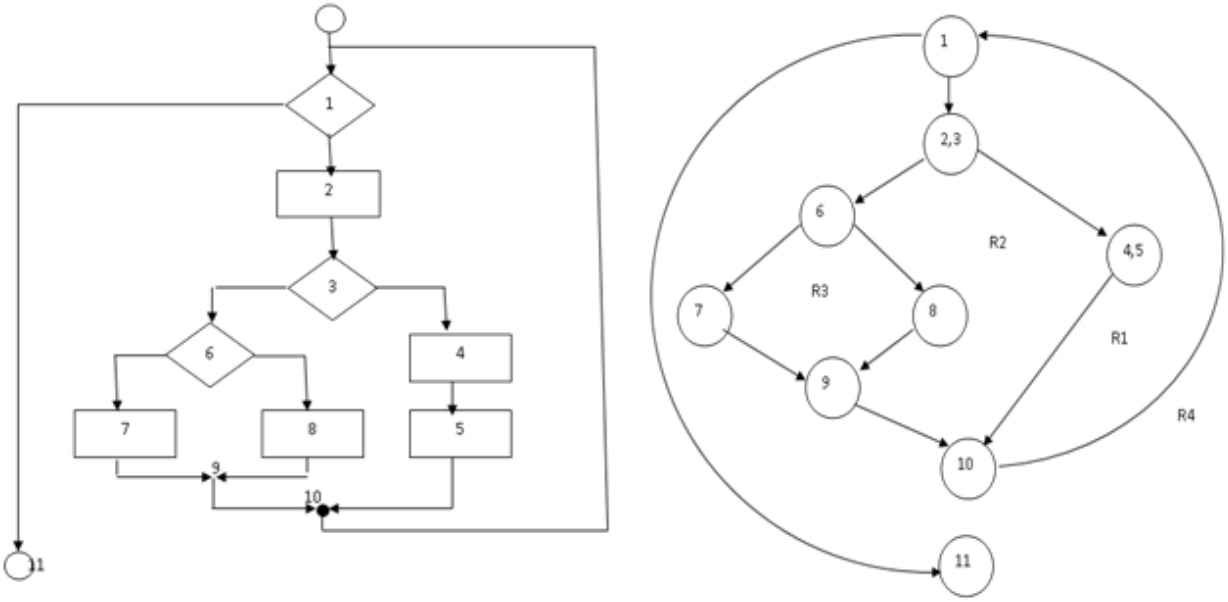



**Flow Graph notes**

- Each circle, called a flow graph node, represents one or more procedural statements.
- A sequence of process boxes and a decision diamond can map into a single node.
- The arrows on the flow graph, called edges, represent flow of control and are analogous to flow chart arrows.

- An edge must terminate at a node, even if the node does not represent any procedural statements.
- Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph and count it as a region.

**Example 1 : Flowchart to Flow graph**




The set of independent paths for example 1 is:

- 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

Note that each new path introduces a new edge. The path

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges.

Paths 1, 2, 3, and 4 defined above comprise a ***basis set*** for the flow graph. That is, if tests can be designed to force execution of these paths (a basis set), every statement in the program will have been guaranteed to be executed at least one time and every condition will have been executed on its true and false side.

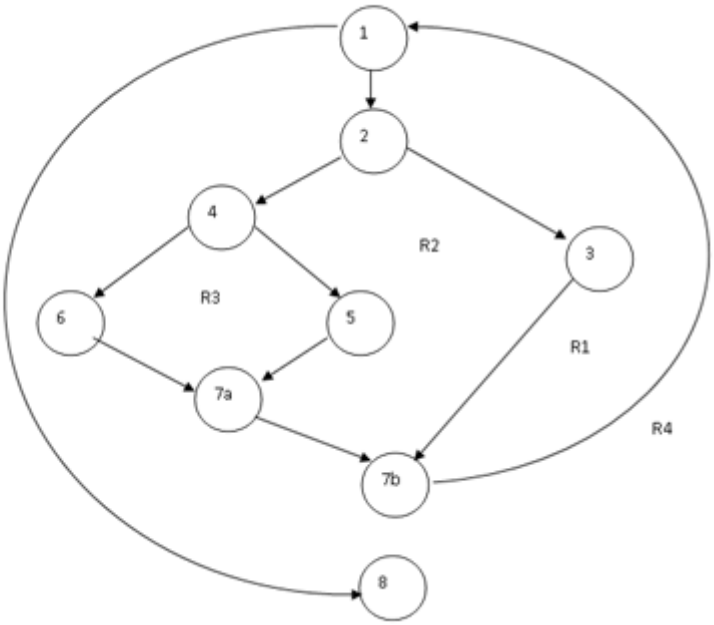
**PDL to Flow graph**

Any procedural design representation can be translated into a flow graph. A program design language (PDL) segment and its corresponding flow graph are shown. Note that the PDL statements have been numbered and a corresponding numbering is used for the flow graph. When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated.

A compound condition occurs when one or more Boolean operators (logical OR. AND, NAND, NOR) are present in a conditional statement. Note that a separate node is created for each of the conditions (a and b).

**Example 2 : PDL to Flow graph**

```
Procedure sort
1. Do while records remain
2. Read record
   If record field 1=0
3.  Then process record
    Store in buffer
    Increment counter
   4.  Else if record field 2=0
   5.    Then reset counter
    6.     Else process record
    Store in file
    7a.  Endif
 Endif
7b.Enddo
8. End
```

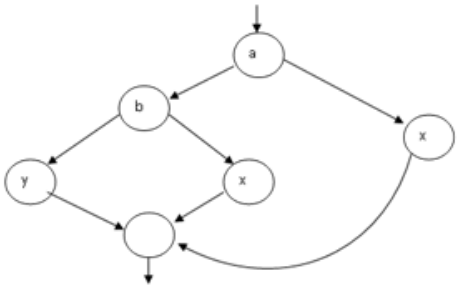



The basis set of example 2 is:

- Path 1: 1-8
- Path 2: 1-2-3-7b-1-8
- Path 3: 1-2-4-5-7a-7b-1-8
- Path 4: 1-2-4-6-7a-7b-1-8

Example 3:

```
If a OR b
   Then procedure x
    Else procedure y
Endif
```

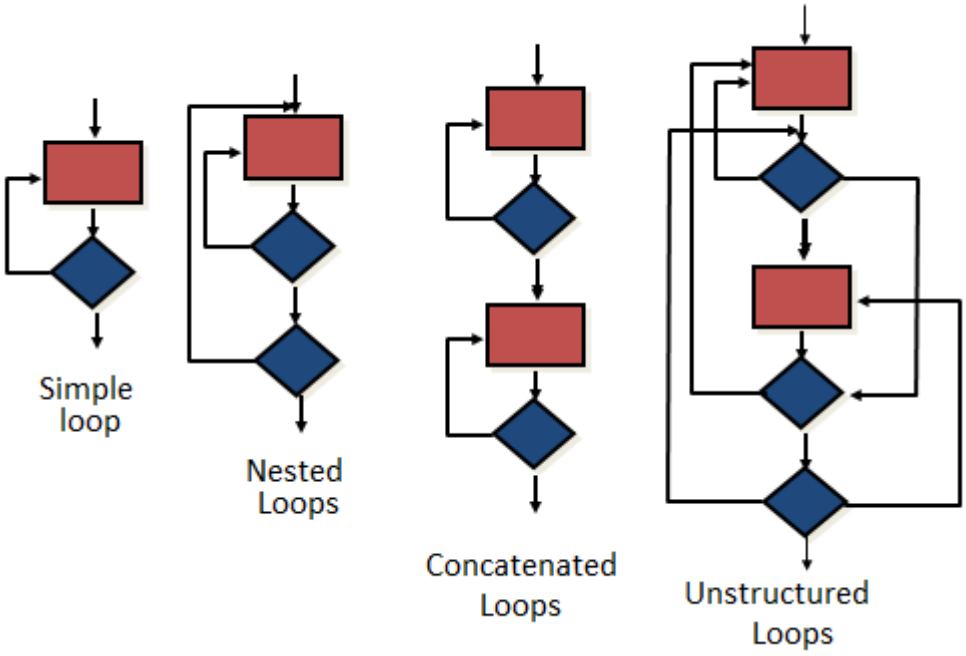



Note that a separate node is created for the statement ( IF a Or b). Each node that contains a condition is called a **predicate node**, and is characterized by two or more edges coming from it.

**Other techniques of white box testing:**

1. **Condition testing** — a test case design method that exercises the logical conditions contained in a program module.
2. **Data flow testing** — selects test paths of a program according to the locations of definitions and uses of variables in the program.
3. **Loop testing** — exercises all loops in the program.

**Loop Testing**



**Loop Testing: Simple Loops**

1. skip the loop entirely
2. only one pass through the loop
3. two passes through the loop
4. m passes through the loop m < n
5. (n-1), n, and (n+1) passes through the loop

where n is the maximum number of allowable passes

**Loop Testing: Nested Loops**

- Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.
- Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.
- Move out one loop and set it up as in step 2, holding all other loops at typical values. Continue this step until the outermost loop has been tested.

**Loop Testing: Concatenated Loops**

- If the loops are independent of one another then treat each as a simple loop
- else* treat as nested loops
- *for example, the final loop counter value of loop 1 is used to initialize loop 2.*