Unit IV: Software Testing

CHAPTER 19: QUALITY CONCEPTS

CHAPTER 22 SOFTWARE TESTING STRATEGIES

CHAPTER 23 TESTING CONVENTIONAL APPLICATIONS

Preamble

Discuss techniques for software **test-case design** for conventional applications.

23.1 SOFTWARE TESTING FUNDAMENTALS

a)Following characters lead to testable software

- Operability—it operates cleanly
- Observability—the results of each test case are readily observed
- Controllability—the degree to which testing can be automated and optimized
- Decomposability—testing can be targeted
- Simplicity—reduce complex architecture and logic to simplify tests
- Stability—few changes are requested during testing
- Understandability—of the design

b)Attributes of 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

23.3 WHITE-BOX TESTING

- ***Glass-box testing or structural testing,*** *is a* test-case design philosophy that uses the control structure described as part of component-level design to derive test cases.

- Using white-box testing methods, 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.

White box testing Vs black box Testing

White box

1. Driven by the program's implementation.
2. Allows tester to be sure every statement has been tested.
3. Difficult to discover missing functionality.
4. Structural testing: Internal program logic is tested
5. Ex

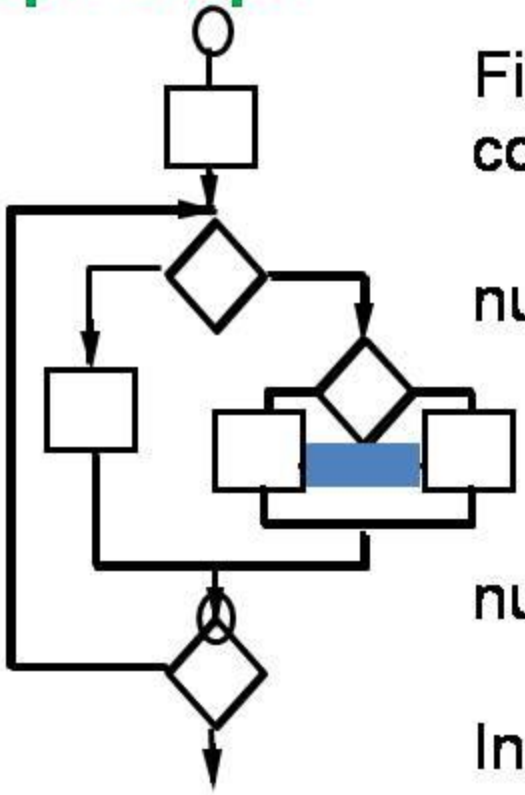
Black box

1. Driven by the program's specification.
2. Checks that the product conforms to specifications.
3. Cannot determine how much code has been tested.
4. Functional testing: Software requirements are tested
5. Ex

23.4 Basis Path Testing

It is a white-box testing technique 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.

a) Flowgraph Graph



First, we compute the cyclomatic complexity:

number of simple decisions + 1

or

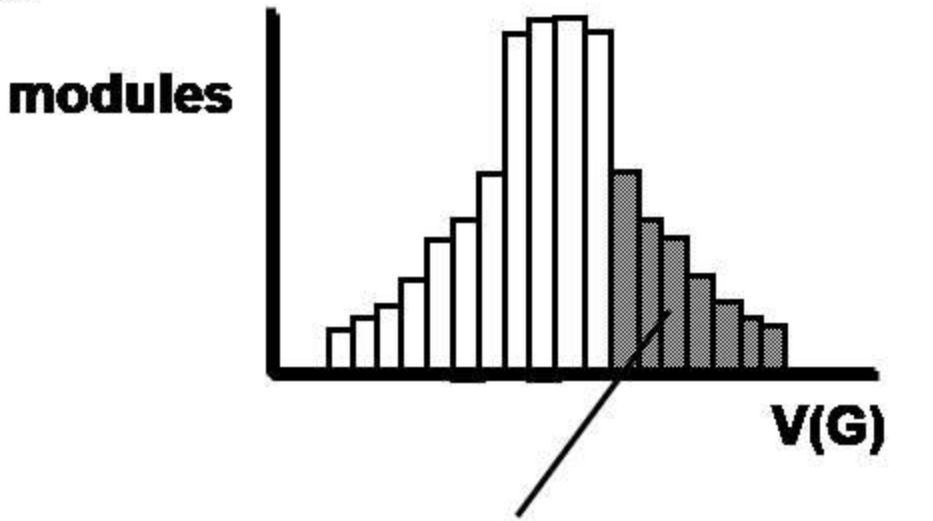
number of enclosed areas + 1

In this case, V(G) = 4

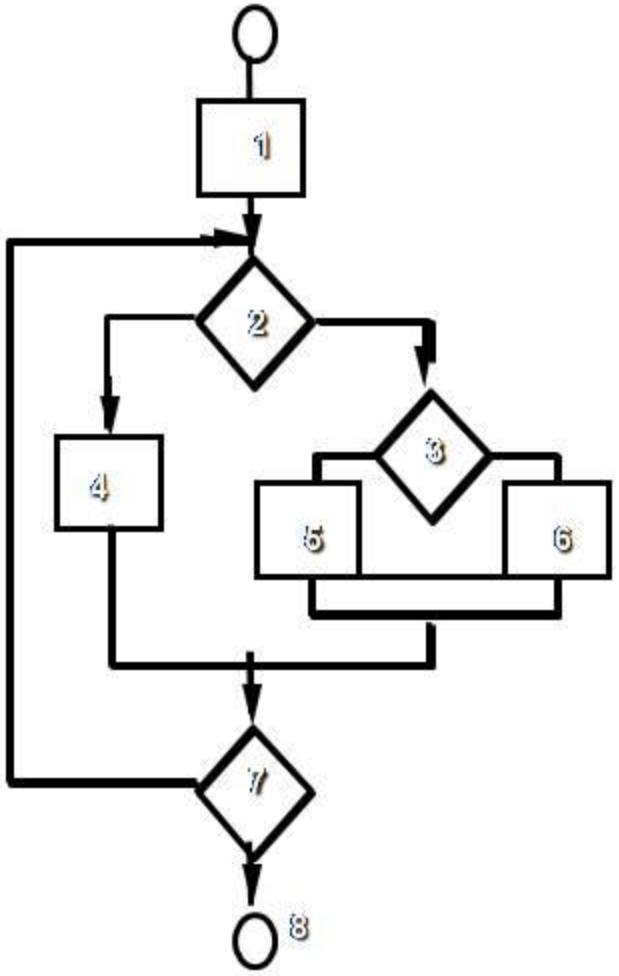
a) Flowgraph Graph

Cyclomatic Complexity

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



b) Independent Program Paths



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.

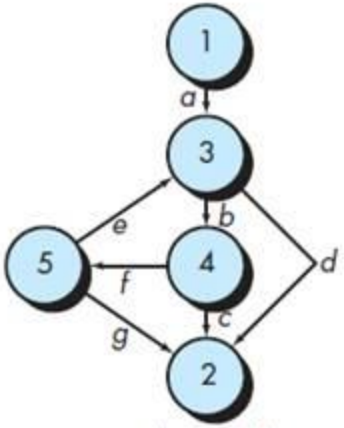
c) Deriving Test Cases

- Using the design or code as a foundation, draw a corresponding **flow graph**.
- Determine the **cyclomatic complexity** of the resultant flow graph.
- Determine a **basis set of linearly independent paths.**
- **Prepare test cases** that will force execution of each path in the basis set.

d) Graph Matrices

- A graph matrix is a square matrix whose size (i.e., number of rows and columns) is equal to the number of nodes on a flow graph
- Each **row and column corresponds** to an identified node, and matrix entries correspond to connections (an edge) between nodes.
- By adding a *link weight* to each matrix entry, the graph matrix can become a **powerful tool for evaluating program control structure** during testing

FIGURE 23.6
Graph matrix



Flow graph

Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

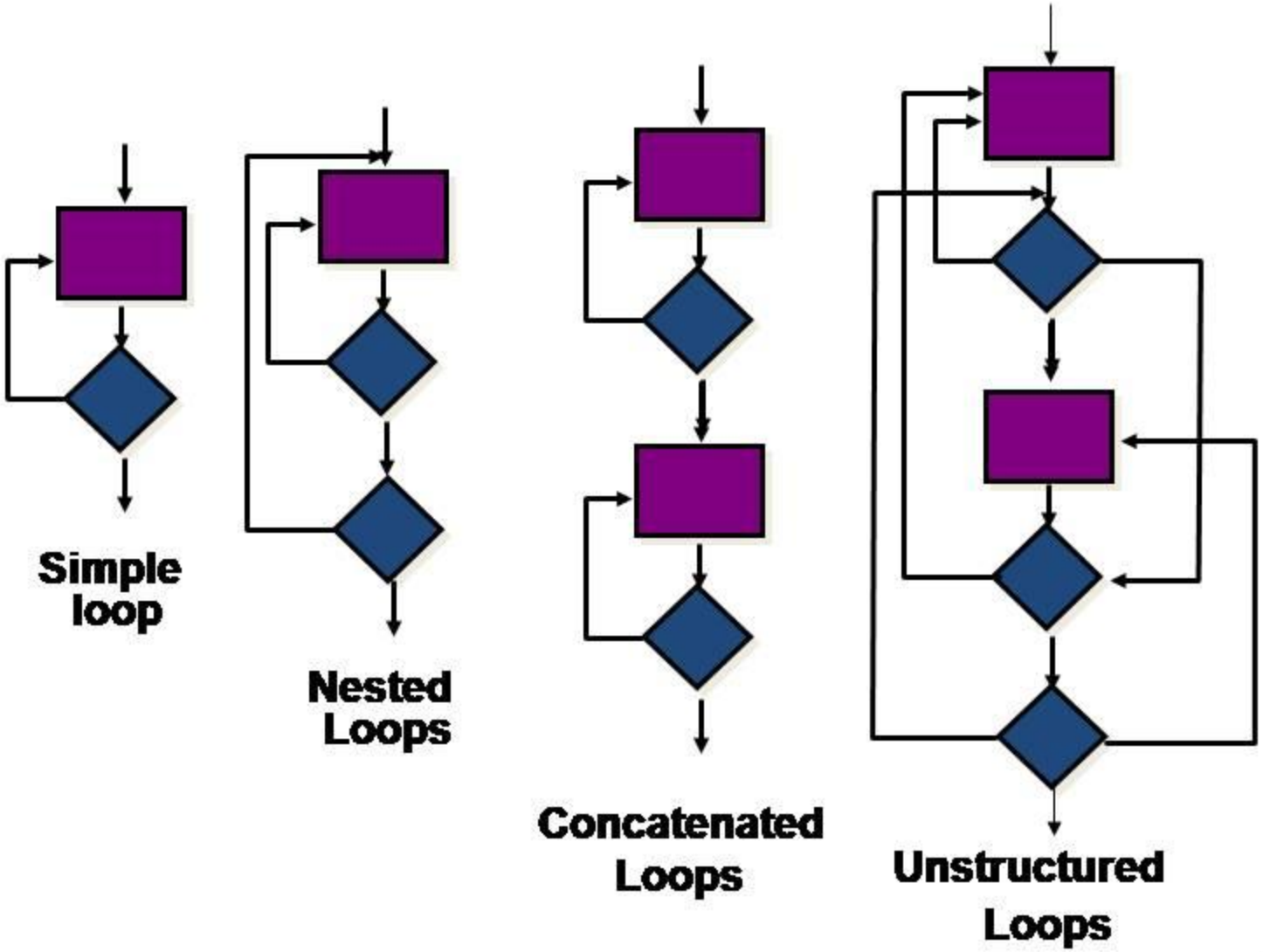
23.5 Control Structure Testing

- Condition testing — a test case design method that exercises the logical conditions contained in a program module.

- Data flow testing — selects test paths of a program according to the locations of definitions and uses of variables in the program.

- Loop testing - is a white-box testing technique that focuses exclusively on the validity of loop constructs.
  - Four different classes of loops can be defined: simple loops, concatenated loops, nested loops, and unstructured loops

Loop Testing



Loop Testing: Simple Loops

Minimum conditions—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

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.

Concatenated Loops

If the loops are independent of one another
then treat each as a simple loop
else* treat as nested loops
endif*

for example, the final loop counter value of loop 1 is used to initialize loop 2.

Concatenated Loops