Software Engineering

Lecture 8

Lectures	Topics
1	Introduction to Software Engineering
2	Software Development Process (SDLC Activities) - SDLC Activity: Specification or Requirement Engineering - SDLC Activity: System Modeling/Design - SDLC Activity: Implementation - SDLC Activity: Testing - SDLC Activity: Evolution (-) - SDLC Activity: Deployment/Installation - SDLC Activity: Maintenance (-)
3	SDLC Activity: Requirement Engineering - Requirement Elicitation - Requirement Analysis and Management - Requirement Validation
4, 5, 6,7	SDLC Activity: System Modeling/Design
8	SDLC Activity: Testing
9,10	
11	
12	

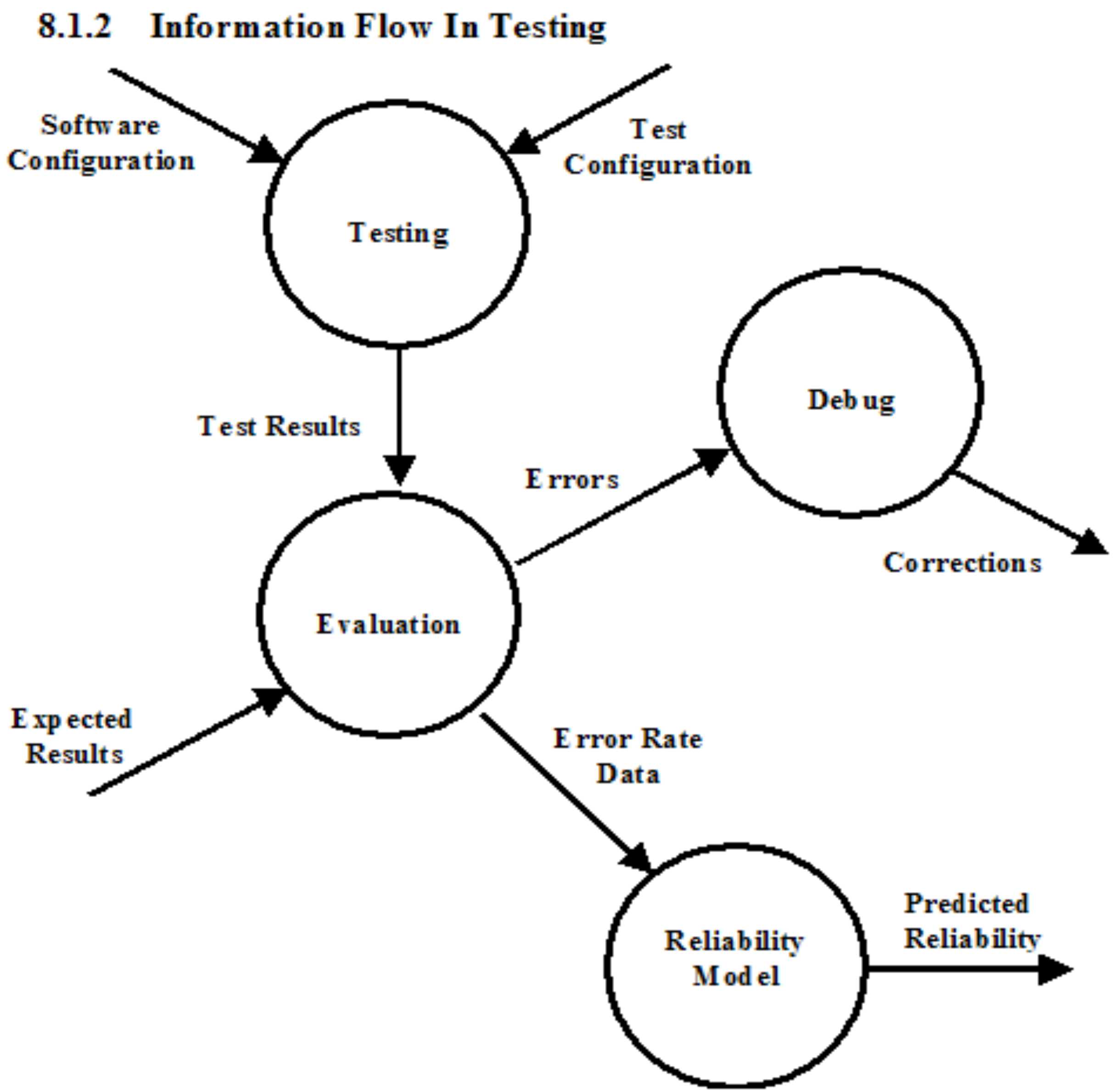
7.1 Software Testing Fundamentals

- Software testing
  - Software quality assurance
  - Ultimate review of specification, design and coding.
- Concerned with the actively identifying errors in software.

7.1.1 Testing Objectives

- Process of executing a program with the intent of finding an error

7.2 Information Flow in Testing



Information Flow in Testing

Two classes of input are provided to the test process, namely:

7.2.1

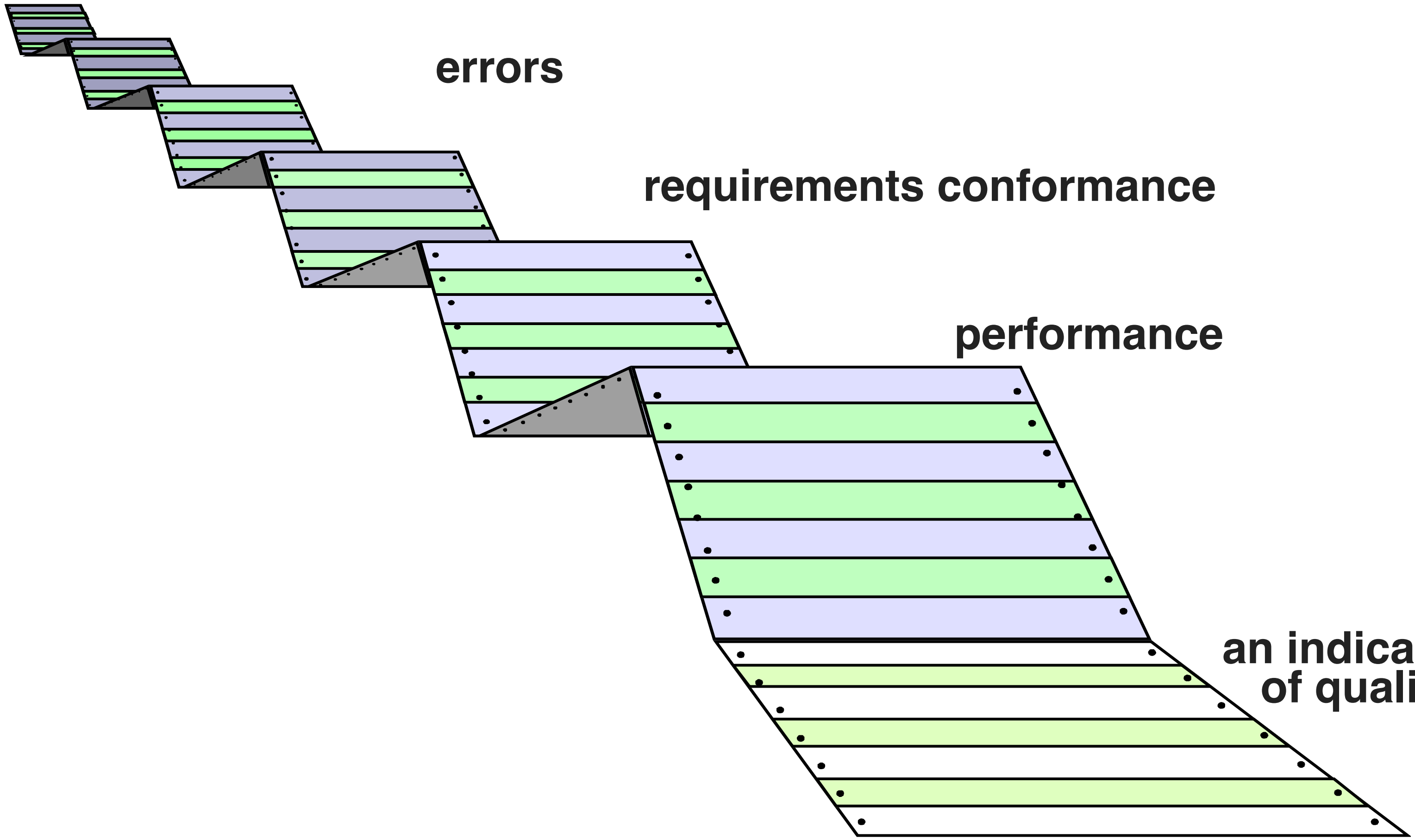
(1) A software configuration that includes a Software Requirements Specification, a Design Specification, and source code.

7.2.2

(2) A test configuration that includes a Test Plan and Procedure, any testing tools that are to be used, and test cases and their expected results.

7.3

What Testing Shows



7.4

Who Tests the Software?

developer

independent tester

Understands the system

but, will test "gently"

and, is driven by "delivery"

Must learn about the system,

but, will attempt to break it

and, is driven by quality

7.5 Strategic approach to software testing

7.5.1 V & V

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

7.6 Testing Strategy

- 7.6.1: Unit Testing
- 7.6.2: Integration Testing

7.7

‘Principles’ of Testing

- The intention of testing - to find errors!
- It is impossible to completely test any nontrivial program.
- Testing requires creativity and hard work.
- Testing is best done by independent people/testers.

7.7.1 Attributes of a “good test”

A good test has a high probability of finding an error.

- The tester must understand the software and attempt to develop a mental picture of how the software might fail.

A good test is not redundant.

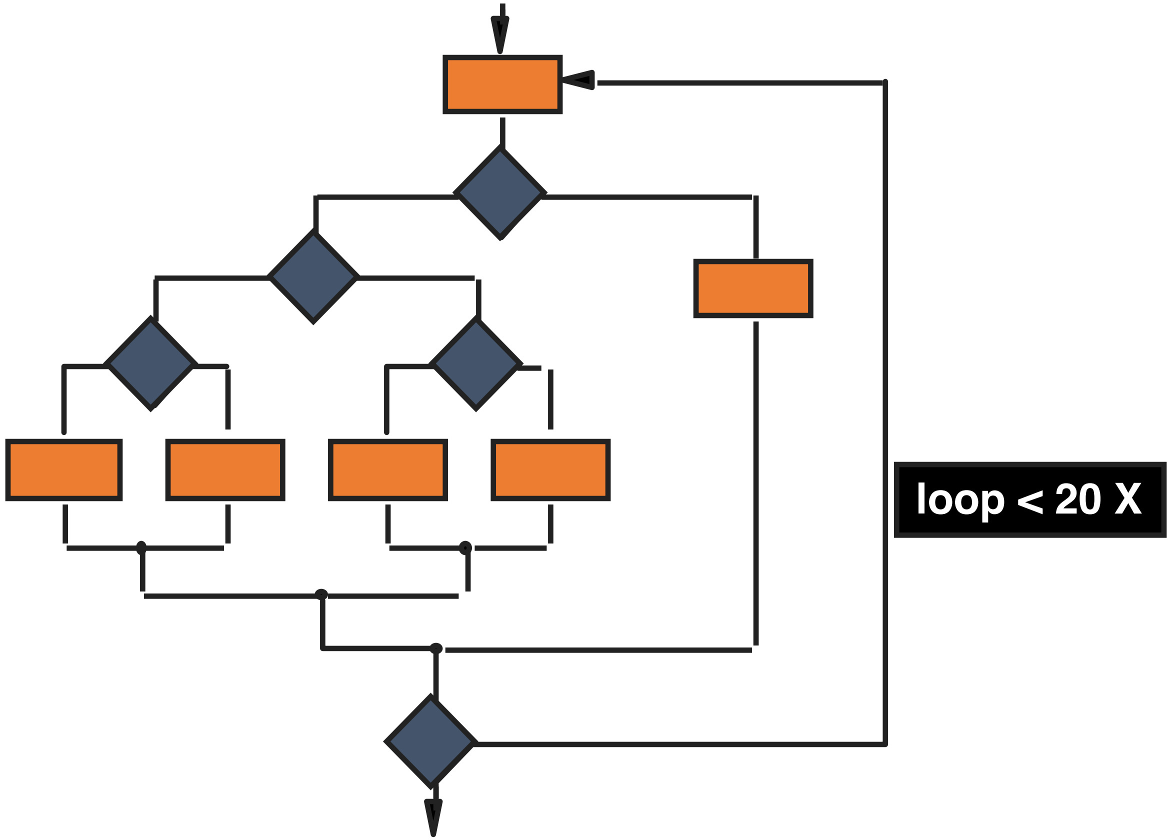
- Testing time and resources are limited.
- There is no point in conducting a test that has the same purpose as another test.

7.7.2 The Impossibility of Complete Testing

- The domain of possible inputs is too large.
- There are too many combinations of data to test.
- There are too many possible paths through the program to test.

7.7.3

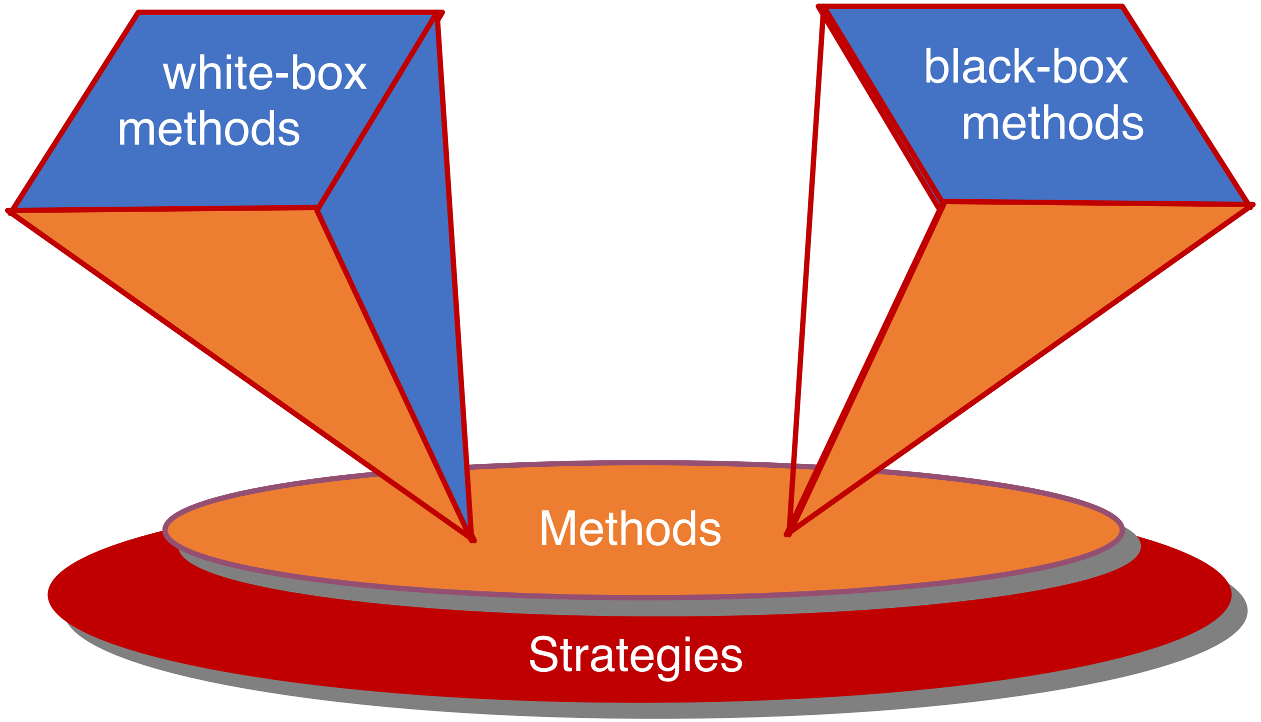
Exhaustive Testing



It would take 3,170 years to test this program!!

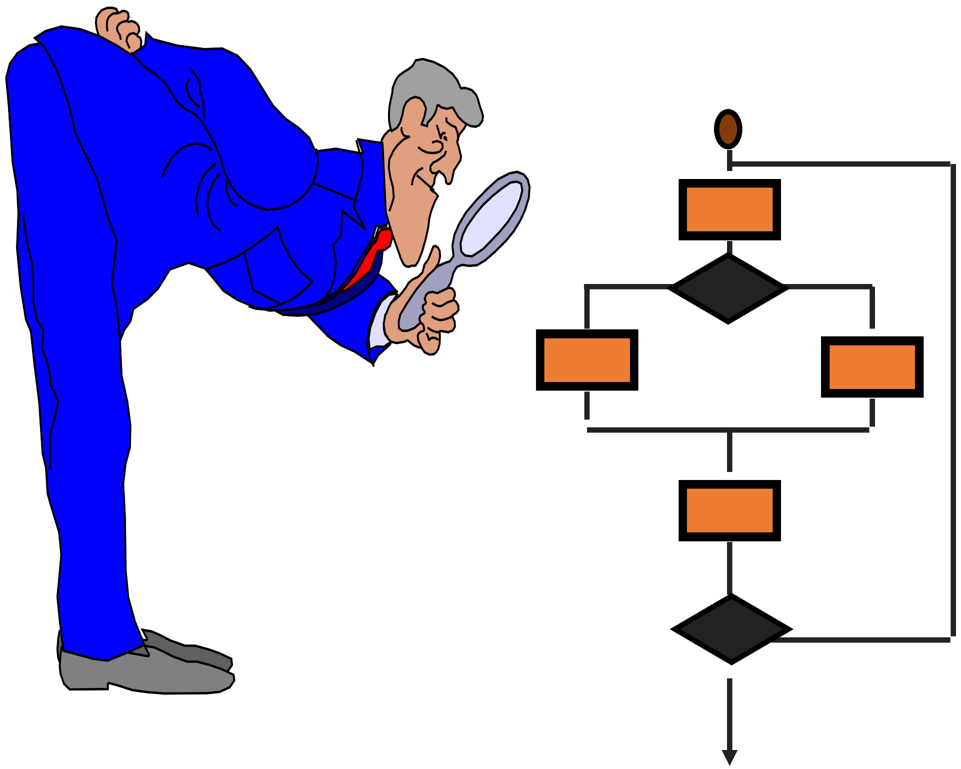
7.8

Software Testing



7.8.1

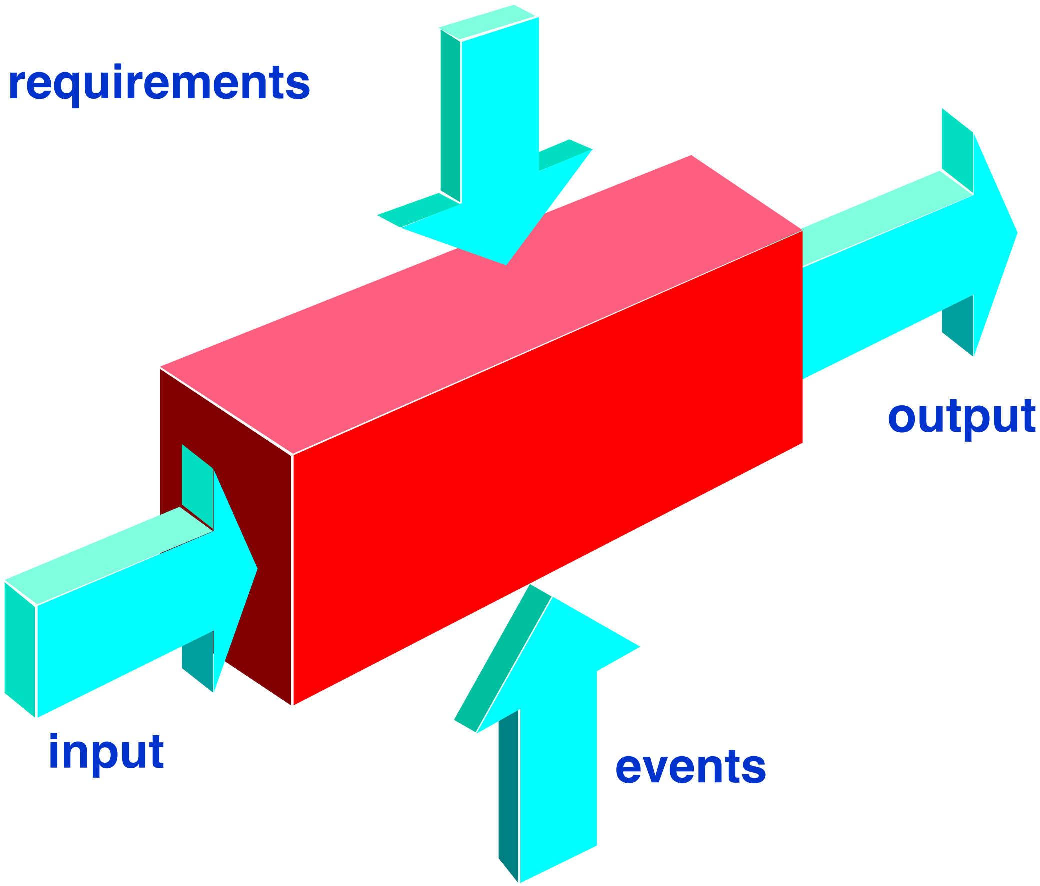
White-Box Testing



... our goal is to ensure that all statements and conditions have been executed at least once ...

7.8.2

Black-Box Testing



Difference between black box and white box testing

- Black Box/(functional) Testing
  - How well the module meets its specification
  - The module is looked at from **outside.**
  - The focus is on inputs to and outputs from, and **the functions** that are to be carried out by the module.
- White Box/(glass box, structural) Testing
  - Testing a program using a knowledge of the internal workings of the program is known as “white box testing”.

7.8.1

White Box Testing

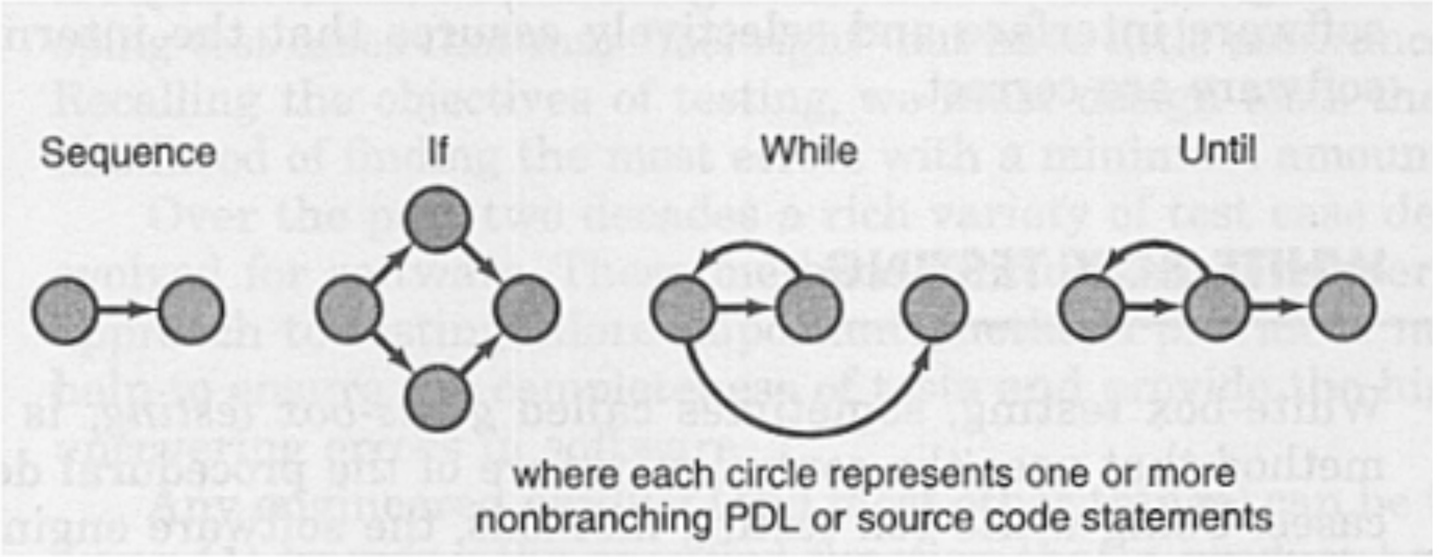
- **White box testing is a test case design method that uses the control structure of the procedural design to derive test cases.**

- 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 or false sides;
3. Execute all loops at their boundaries and within their operational bounds ; and

7.8.1.1 Basis Path Testing Methods

- This testing technique makes use of simple flow graph notation, which is illustrated here:



LAB

7.8.1.1

Basis Path Testing

- Basis Path Testing is a white box testing techniques proposed by Tom McCabe.
- Enables that test case designer to derive a **logical complexity measure.**
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

LAB

7.8.1.1

Basis Path Testing

Cyclomatic Complexity V(G)

- Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.

Complexity can be computed in one of several ways:

1. V(G) = Number of regions of the flow graph

2. V(G) = E-N+2; where E is the number of flow graph edges and N is the number of flow graph nodes.

3. V(G) =P+1 where P is the number of predicate nodes contained in the flow graph G.
   - Predicate notes are characterized by two or more edges emanating from it.

LAB

7.9.1.1

Basis Path Testing

The basis path testing can be applied as a series of steps:

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

2. Determine the cyclomatic complexity of the resultant flow graph.

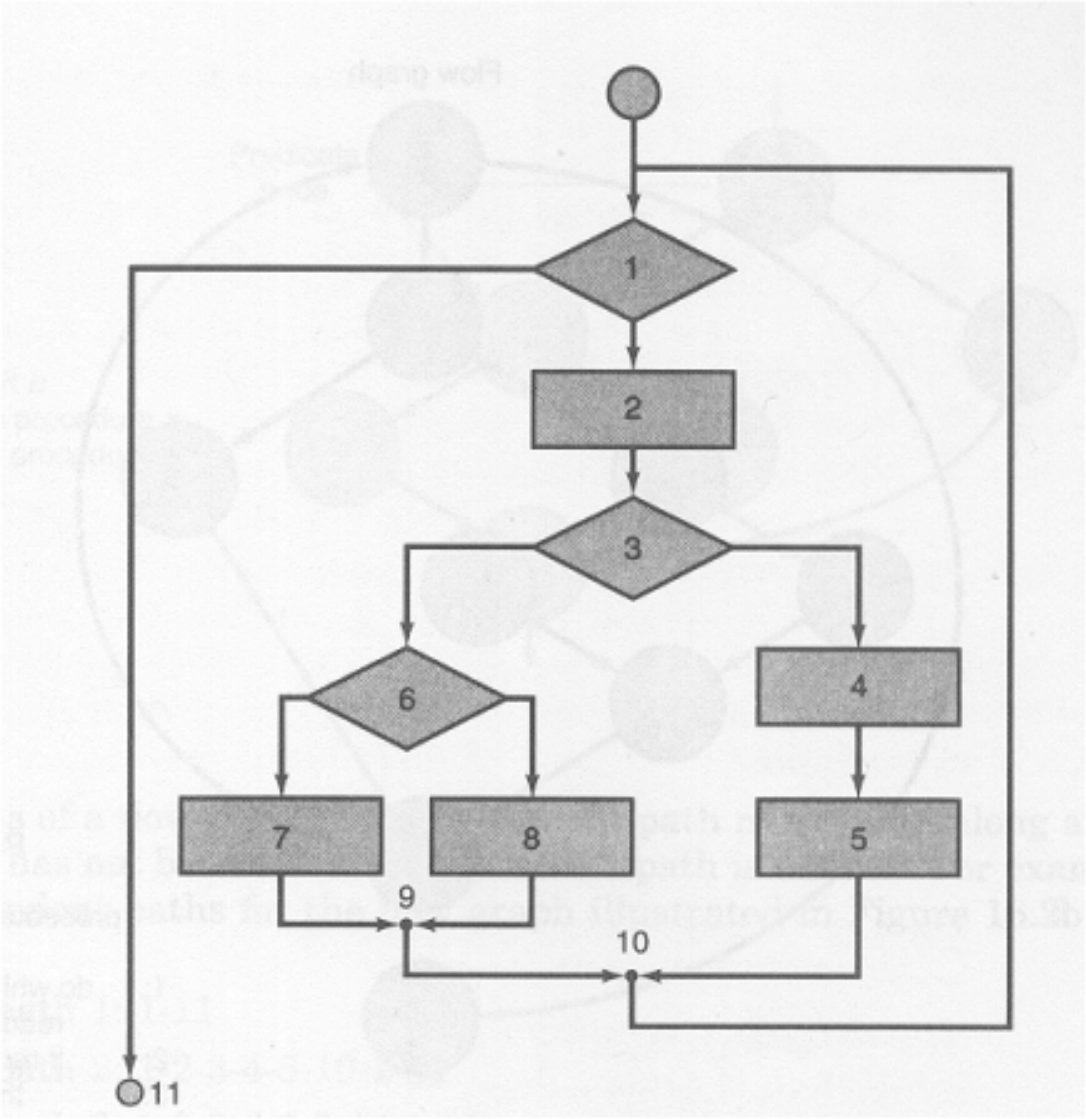
3. Determine a basis set of linearly independent paths.

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

LAB

7.8.1.1

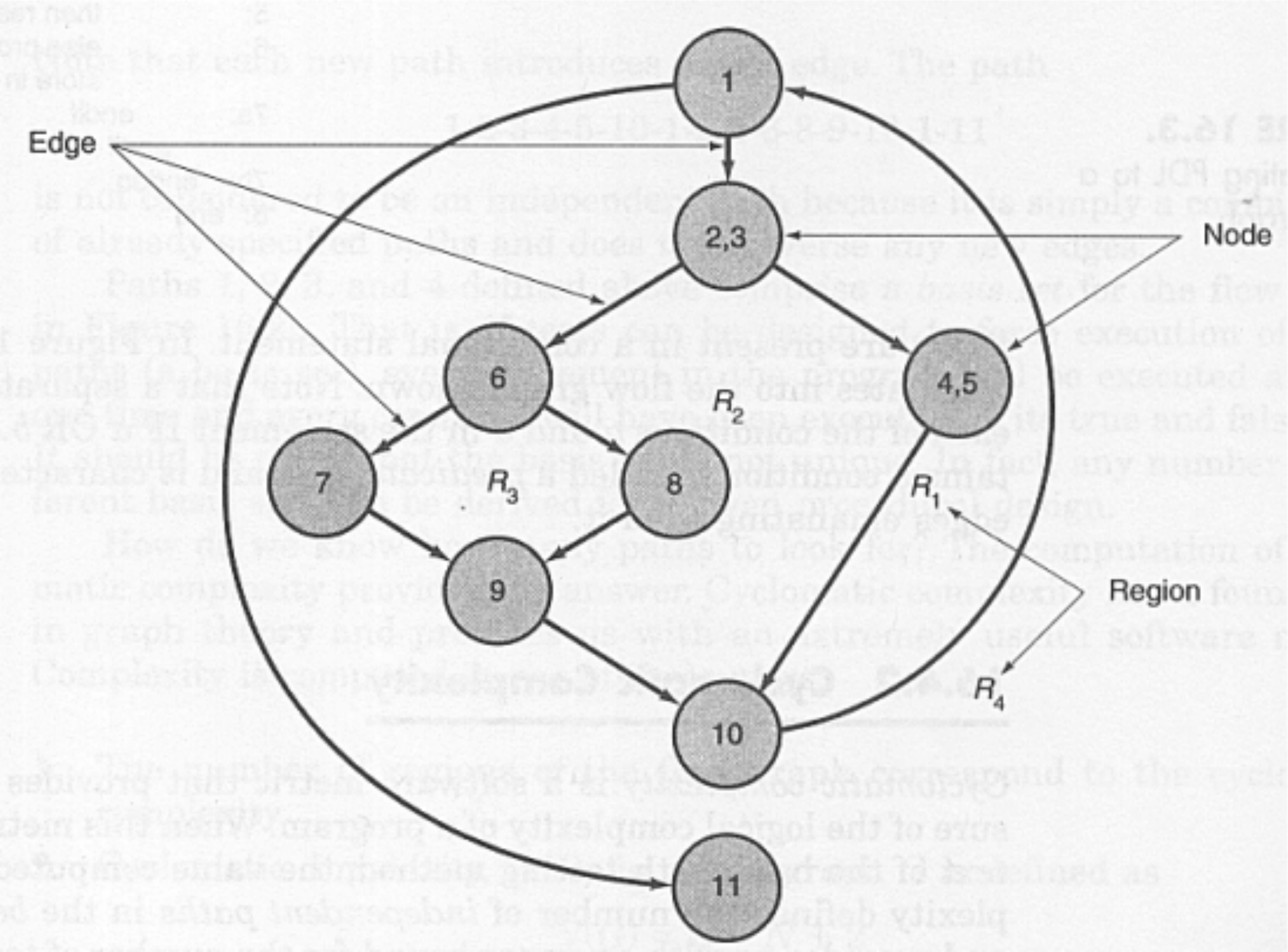
Basis Path Testing: An Example



LAB

7.8.1.1

Basis Path Testing: An Example



LAB

7.8.1.1

Basis Path Testing: An Example

1. The flow has 4 regions (marked as R1, R2, R3 and R4); hence V(G) = 4

2. V(G) = E - N + 2 = 11 edges - 9 nodes + 2 = 4

3. V(G) = P + 1 = 3 Predicate Notes + 1 = 4 (Predicate nodes are: node (2,3), node (4,5), and node (6).

Thus, the value of V(G) provides us with an upper bound for the number of independent paths ....(etc...)

The set of independent paths are:
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

LAB

Basis Path Testing: An Exercise

```
While (x>0) and (data <> 0) do
    if (x>5) then
        y = x + 3
    else
        y = 5
    endif
    x = x - 1
    data = data - 1
end While
If data = 0 then
    write ('Goodbye')
else
    write ('Error')
endif
write ('End of Program')
```

LAB

Basis Path Testing: An Exercise

1. Carefully consider the pseudo-code above, and draw the corresponding *Flow Graph*. In your answer, also indicate the pseudo-code *statement* that each *node* is representing.

2. Calculate the value of Cyclomatic Complexity V(G). Calculate this number using the Number of Regions method.

3. Using the Cyclomatic Complexity, design a set of *test cases* that will adequately test the entire program.

LAB

7.9

Testing Flow