Chapter 18

- **Testing Conventional Applications**

Testability

- 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

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

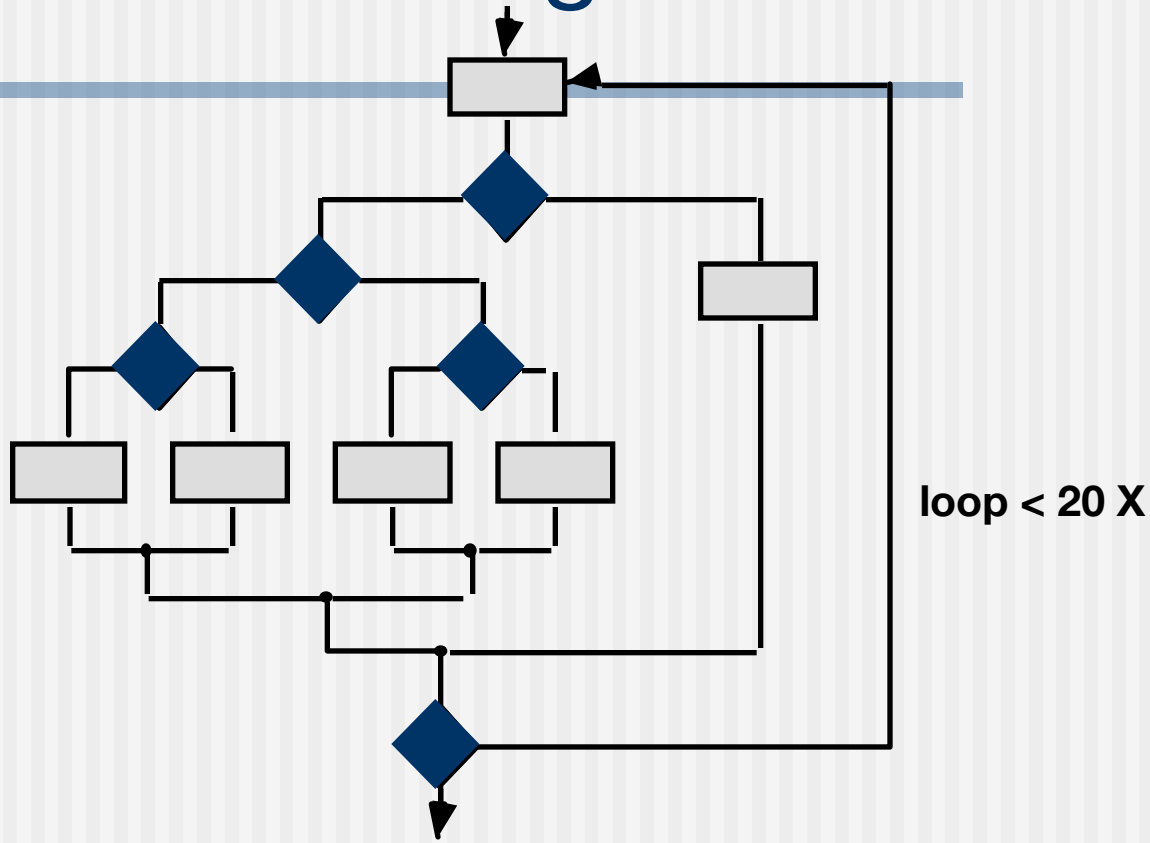
Internal and External Views

- Any engineered product (and most other things) can be tested in one of two ways:
  - Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;
  - Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised.

Test Case Design

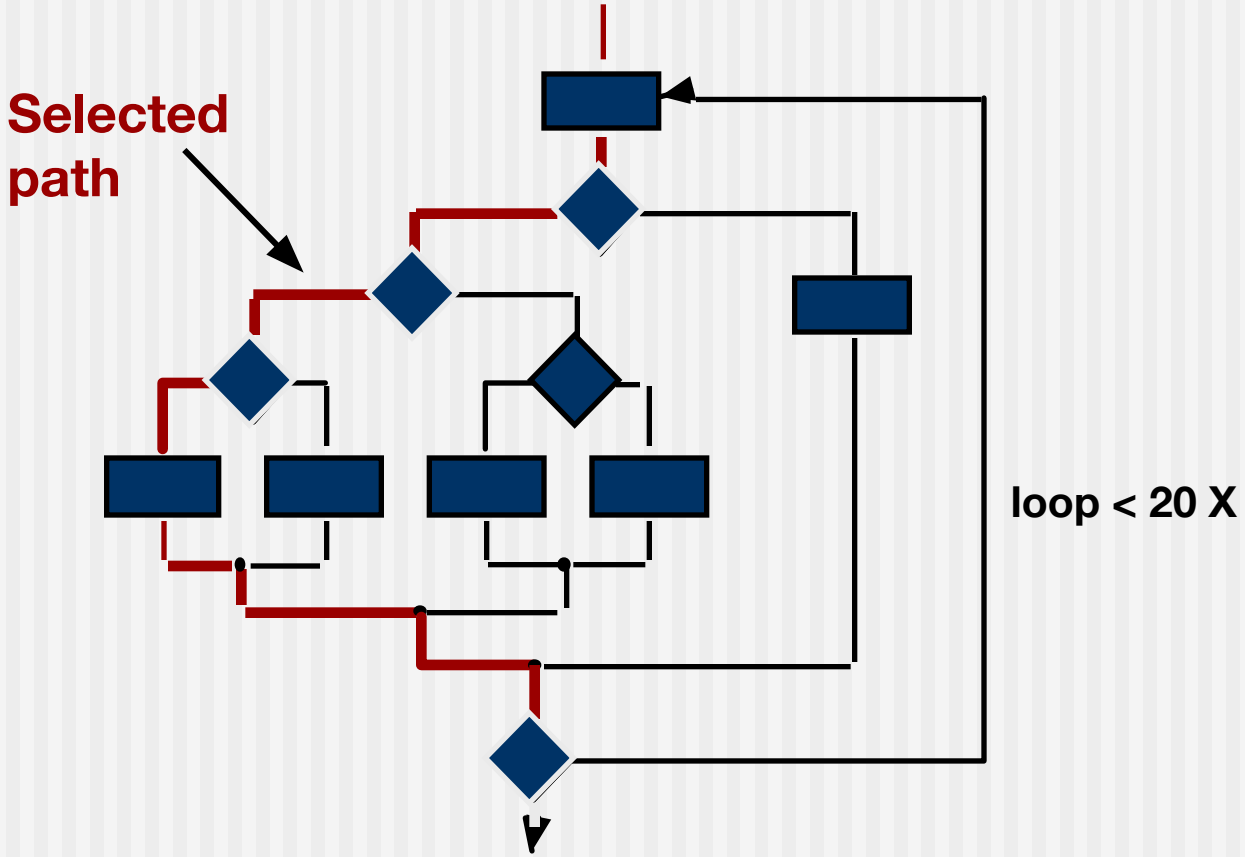
OBJECTIVE	**to uncover errors**
CRITERIA	**in a complete manner**
CONSTRAINT	**with a minimum of effort and time**

Exhaustive Testing

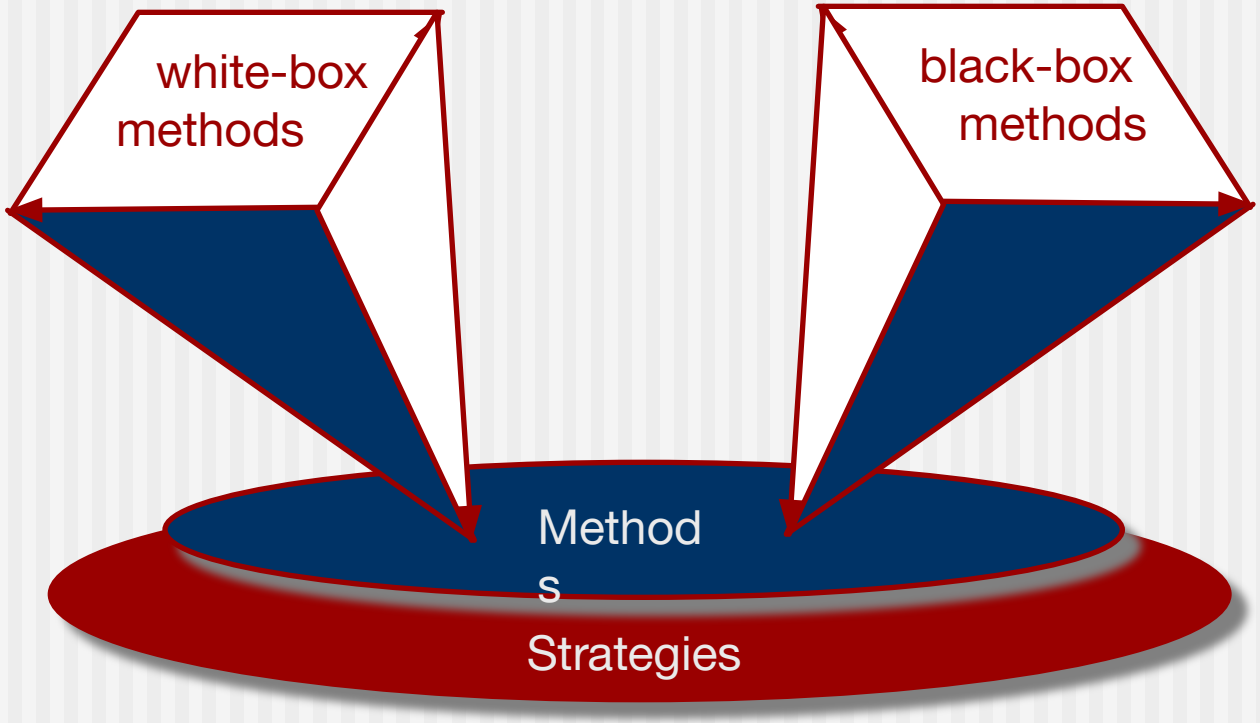


There are 10^{14} possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

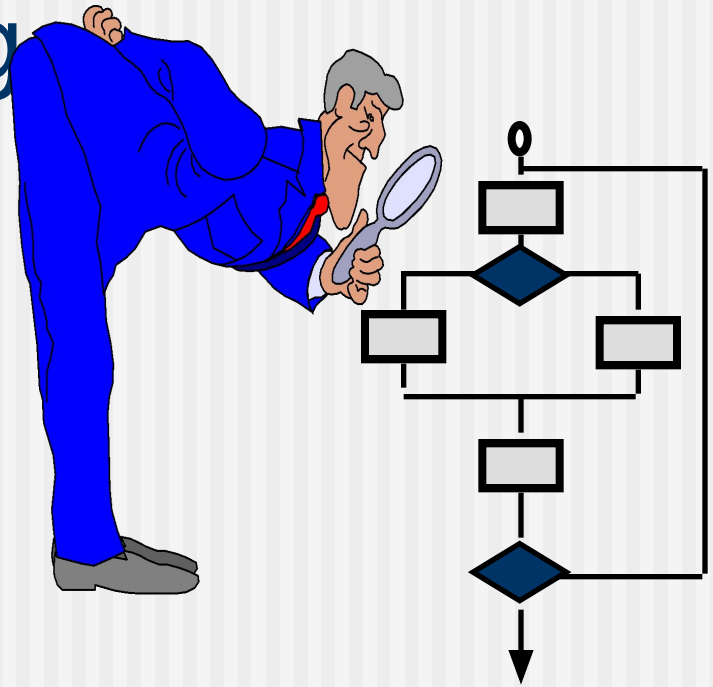
Selective Testing



Software Testing



White-Box Testing

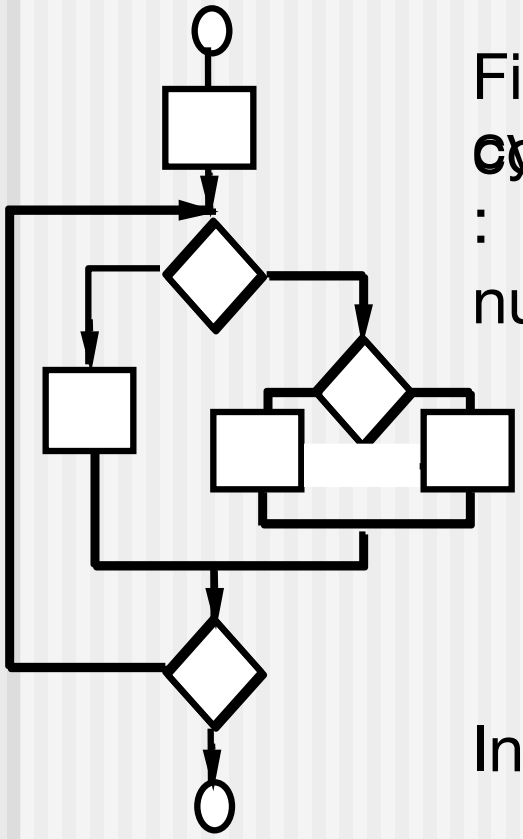


... our goal is to ensure that all statements and conditions have been executed at least once

Why Cover?

- **logic errors and incorrect assumptions are inversely proportional to a path's execution probability**
- **we often believe that a path is not likely to be executed; in fact, reality is often counter intuitive**
- **typographical errors are random; it's likely that untested paths will contain some**

Basis Path Testing



First, we compute the cyclomatic complexity:

number of simple decisions + 1

or

number of enclosed areas + 1

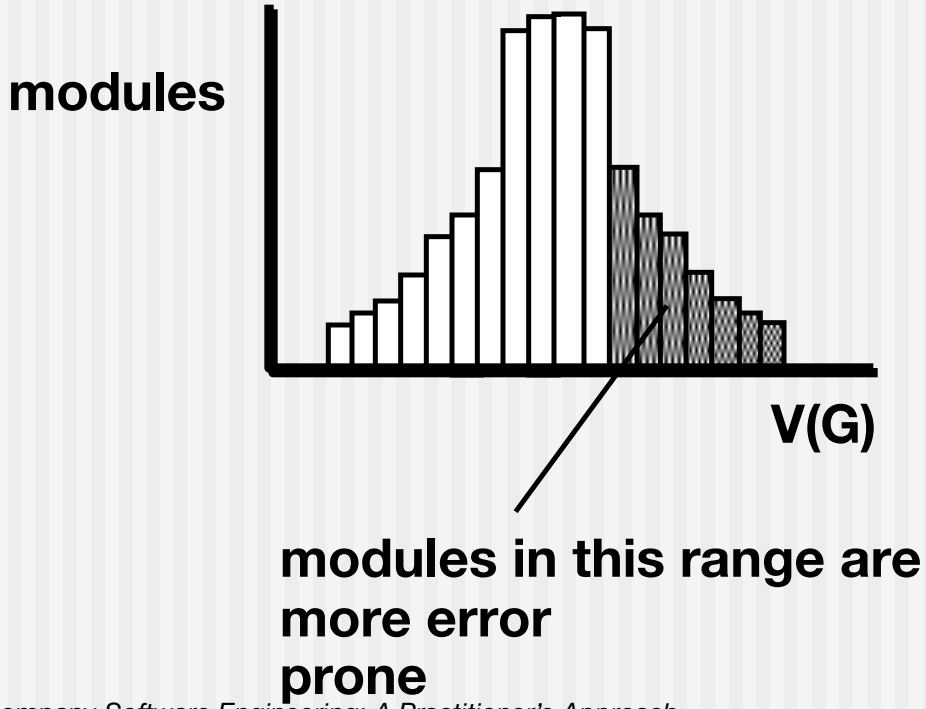
or

No. of Edges - No. of Node + 2

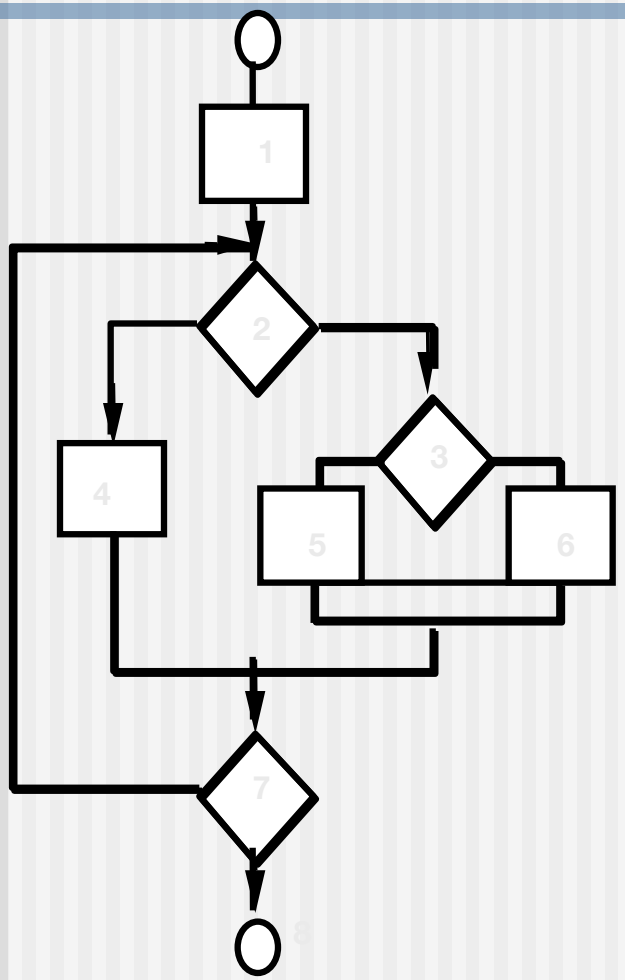
In this case, V(G) = 4

Cyclomatic Complexity

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



Basis Path Testing



Next, we derive the independent paths:

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

Path 1: 1,2,3,6,7,8

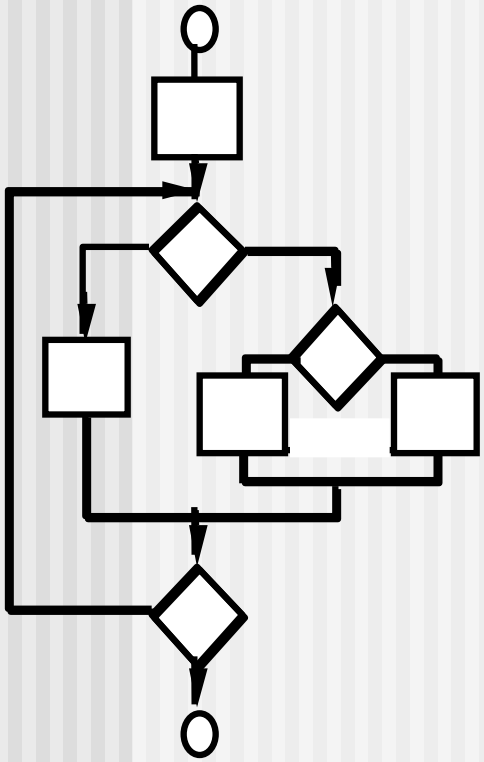
Path 2: 1,2,3,5,7,8

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
- basis path testing should be applied to critical modules

Deriving Test Cases

- *Summarizing:*
  - 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.

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

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

Data Flow Testing

- **Data Flow Testing** is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams. It is concerned with:
- Statements where variables receive values,
- Statements where these values are used or referenced.
  - For a statement with S as its statement number
    - DEF(S) = {X | statement S contains a definition of X}
    - USE(S) = {X | statement S contains a use of X}

Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program.
Reference or define anomalies in the flow of the data are detected at the time of associations between values and variables.

Continue…

These anomalies are:

- A variable is defined but not used or referenced,
- A variable is used but never defined,
- A variable is defined twice before it is used

Advantages of Data Flow Testing:

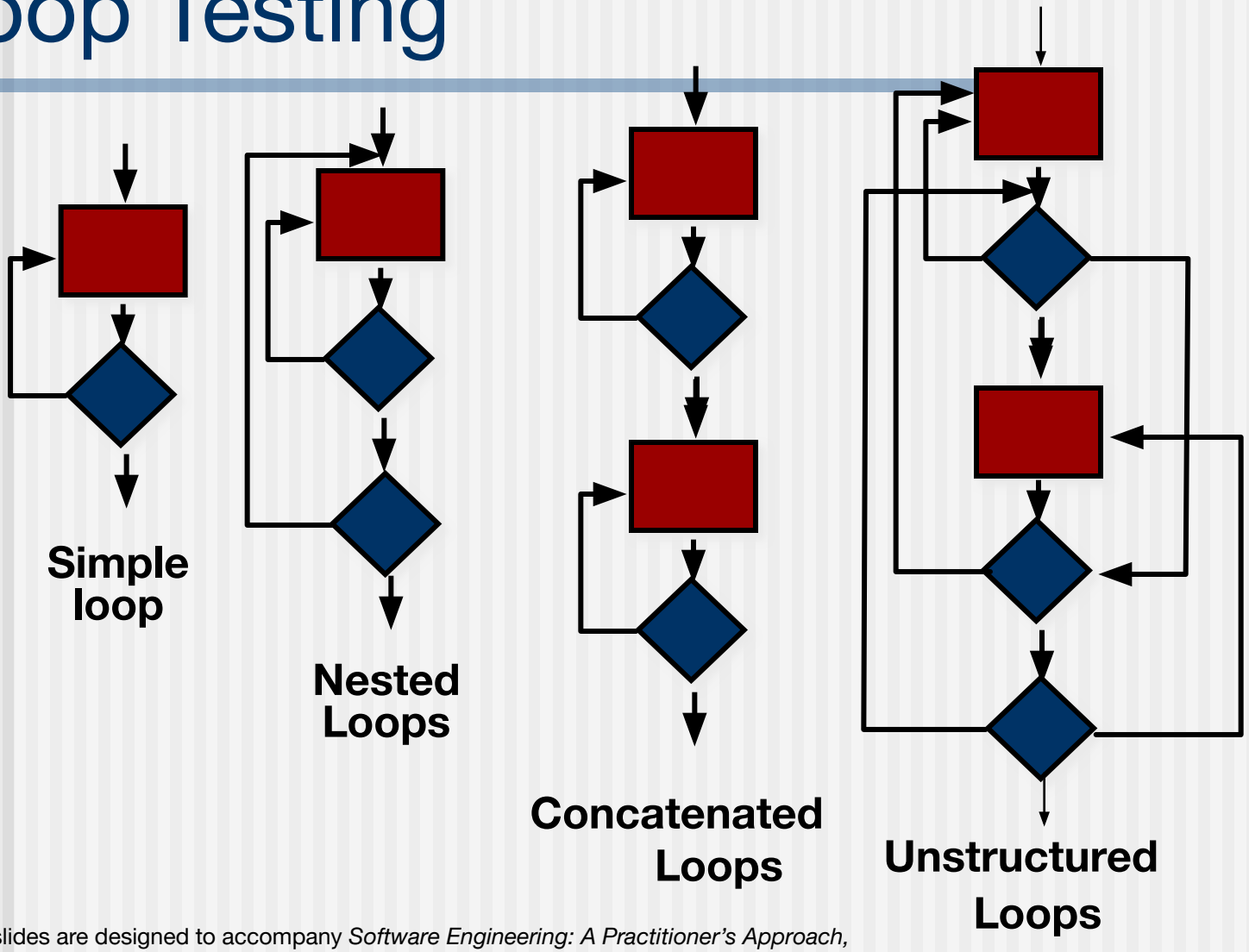
Data Flow Testing is used to find the following issues-

- To find a variable that is used but never defined,
- To find a variable that is defined but never used,
- To find a variable that is defined multiple times before it is use,
- De allocating a variable before it is used.

Disadvantages of Data Flow Testing

- Time consuming and costly process
- Requires knowledge of programming languages
- Ref. example…flow graph testing_ example.docx

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.

Continue….

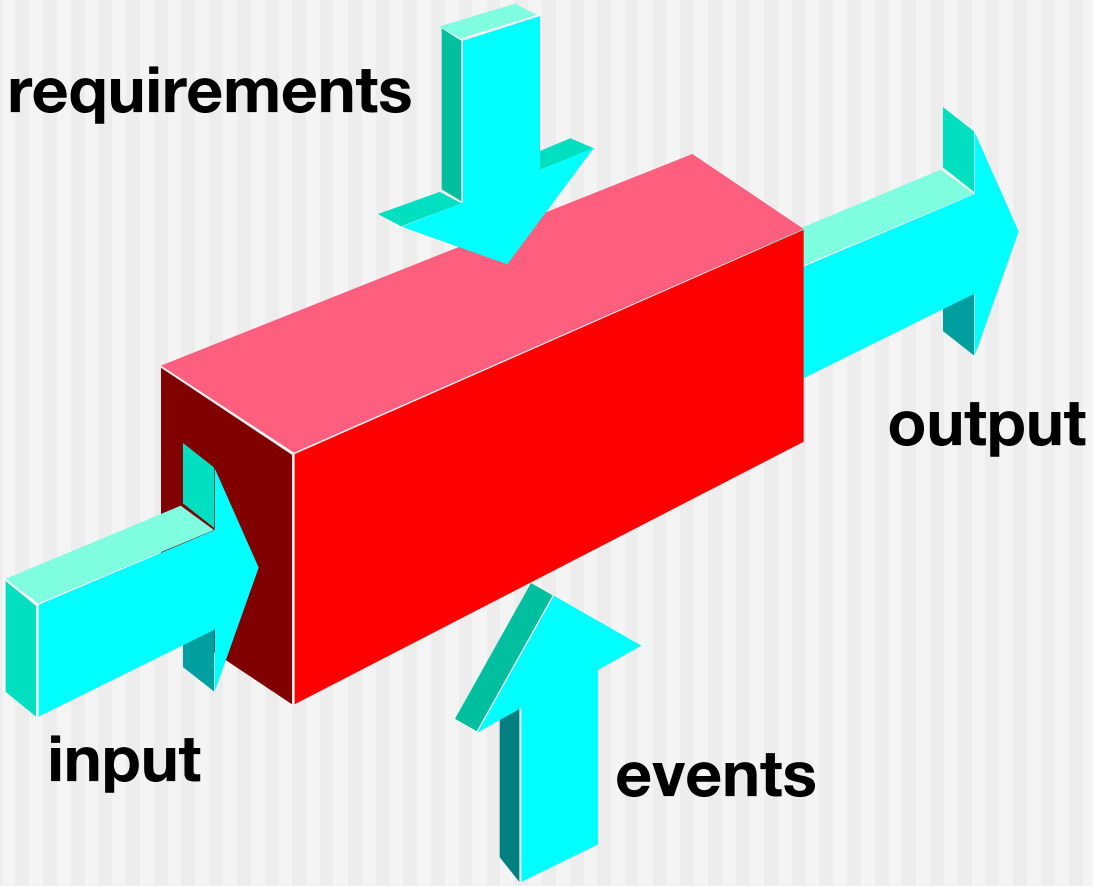
Limitation in Loop testing

- Loop bugs show up mostly in low-level software
- The bugs identified during loop testing are not very noticeable
- Many of the bugs might be detected by the operating system as such they will cause memory boundary violations, detectable pointer errors, etc.

Summary

- In Software Engineering, Loop testing is a White Box Testing. This technique is used to test loops in the program.
- Loops testing can reveal performance/capacity bottlenecks
- Loop bugs show up mostly in low-level software

Black-Box Testing



Black-Box Testing

- How is functional validity tested?
- How is system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

Graph-Based Methods

This technique of Black box testing involves a graph drawing that depicts the link between the causes (inputs) and the effects (output), which trigger the effects.

This testing utilizes different combinations of output and inputs. It is a helpful technique to understand the software's functional performance, as it visualizes the flow of inputs and outputs in a lively fashion.

Equivalence Partitioning

- It is a software testing technique or black-box testing that divides input domain into classes of data, and with the help of these classes of data, test cases can be derived.
- In equivalence partitioning, equivalence classes are evaluated for given input conditions. Whenever any input is given, then type of input condition is checked, then for this input conditions, Equivalence class represents or describes set of valid or invalid states.

Guidelines for Equivalence Partitioning :

- If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.
- If a specific value is given as input, then one valid and two invalid equivalence classes are defined.
- If a member of set is given as an input, then one valid and one invalid equivalence class is defined.
- If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined. Equivalence partitioning testing Example.docx

Boundary Value Analysis

- Boundary Value Testing is one of the popular software testing mechanism, where testing of data is done based on boundary values or between two opposite ends , i.e. minimum and maximum value.
- The black box testing techniques are helpful for detecting any errors or threats that happened at the boundary values of valid or invalid partitions rather than focusing on the center of the input data.
- **Importance :** This is done when there is a huge number of test cases are available for testing purposes and for checking them individually, this testing is of great use.Test _boundary value.docx

Comparison Testing

- Used only in situations in which the reliability of software is absolutely critical (e.g., human-rated systems)
  - Separate software engineering teams develop independent versions of an application using the same specification
  - Each version can be tested with the same test data to ensure that all provide identical output
  - Then all versions are executed in parallel with real-time comparison of results to ensure consistency

Orthogonal Array Testing

- **Orthogonal Array Testing (OAT)** is software testing technique that uses orthogonal arrays to create test cases. It is statistical testing approach especially useful when system to be tested has huge data inputs. Orthogonal array testing helps to maximize test coverage by pairing and combining the inputs and testing the system with comparatively less number of test cases for time saving.
- For example, when a train ticket has to be verified, factors such as – the number of passengers, ticket number, seat numbers, and train numbers have to be tested. One by one testing of each factor/input is cumbersome. It is more efficient when the QA engineer combines more inputs together and does testing. In such cases, we can use the Orthogonal Array testing method.orthogonal test_example.doc

Model-Based Testing

- Analyze an existing behavioral model for the software or create one.
  - Recall that a *behavioral model* indicates how software will respond to external events or stimuli.
- Traverse the behavioral model and specify the inputs that will force the software to make the transition from state to state.
  - The inputs will trigger events that will cause the transition to occur.
- Review the behavioral model and note the expected outputs as the software makes the transition from state to state.
- Execute the test cases.
- Compare actual and expected results and take corrective action as required.

Software Testing Patterns

- Testing patterns are described in much the same way as design patterns (Chapter 12).
- *Example:*
  - *Pattern name:* **ScenarioTesting**
  - *Abstract:* Once unit and integration tests have been conducted, there is a need to determine whether the software will perform in a manner that satisfies users. The **ScenarioTesting** pattern describes a technique for exercising the software from the user's point of view. A failure at this level indicates that the software has failed to meet a user visible requirement. [Kan01]

Types of Black Box Testing

- **Functional Testing:**

This type of testing is useful for the testers in identifying the functional requirements of a software or system.

- **Regression Testing**:

This testing type is performed after the system maintenance procedure, upgrades or code fixes to know the impact of the new code over the earlier code.

- **Non-Functional Testing**:

This testing type is not connected with testing for any specific functionality but relates to non-functional parameters like usability, scalability and performance.

Difference between Black Box Testing and White Box Testing

Used to test software without knowing the internal structure of the software	Performed after knowing the internal structure of the software
Carried out by testers	Performed by developers
Does not require programming knowledge	Requires programming knowledge
Requires implementation knowledge	Does not require implementation knowledge
Higher level testing	Lower level testing
Consumes less time	Consumes a lot of time
Done in the trial and error method	Data domains and boundaries can be tested
Types of black box testing 1. Functional testing 2. Regression testing 3. Non-functional testing	**Types of white box testing** 1. Path testing 2. Loop testing 3. Condition testing
Not suitable for algorithm testing	Suitable for algorithm testing

Exercise

- What is “GOOD” test? What “Testability” includes
- Differentiate white box and black box testing
- Calculate cyclomatic complexity for a given flowgraph using all four methods and list out independent path
- Write down steps to derive test cases
- Explain dataflow testing with example also write down advantages and disadvantages
- Explain loop testing with figure
- What is Blackbox testing ? Explain equivalence partitioning with example
- Explain Functional , nonfunctional testing and regression testing
- Explain orthogonal Array testing with example
- Explain Boundary value analysis with example.