Introduction to Dynamic Analysis

Static Analysis versus Dynamic Analysis

- **Static Analysis** -- the static examination of a product or a representation of the product for the purpose of inferring properties or characteristics
- **Dynamic Analysis** -- the "execution" of a product or representation of a product for the purpose of inferring properties or characteristics
- **Testing** -- the (systematic) selection and subsequent "execution" of sample inputs from a product's input space in order to infer information about the product's behavior.
  - usually trying to uncover failures
  - the most common form of dynamic analysis

Approaches

- Dynamic Analysis
  - Assertions
  - Error seeding, mutation testing
  - Coverage criteria
  - Fault-based testing
  - Specification-based testing
  - Object oriented testing
  - Regression testing
- Static Analysis
  - Inspections
  - Software metrics
  - Symbolic execution
  - Dependence Analysis
  - Data flow analysis
  - Software Verification

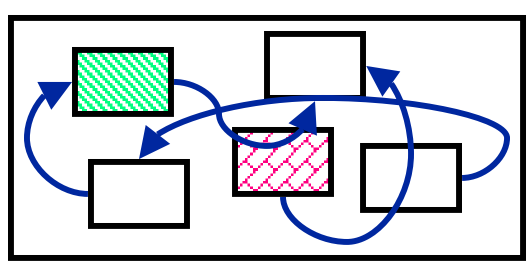
Types of Testing--what is tested

- Unit testing-exercise a single simple component
  - Procedure
  - Class
- Integration testing-exercise a collection of inter-dependent components
  - Focus on interfaces between components
- System testing-exercise a complete, stand-alone system
- Acceptance testing-customer's evaluation of a system
  - Usually a form of system testing
- Regression testing-exercise a changed system
  - Focus on modifications or their impact

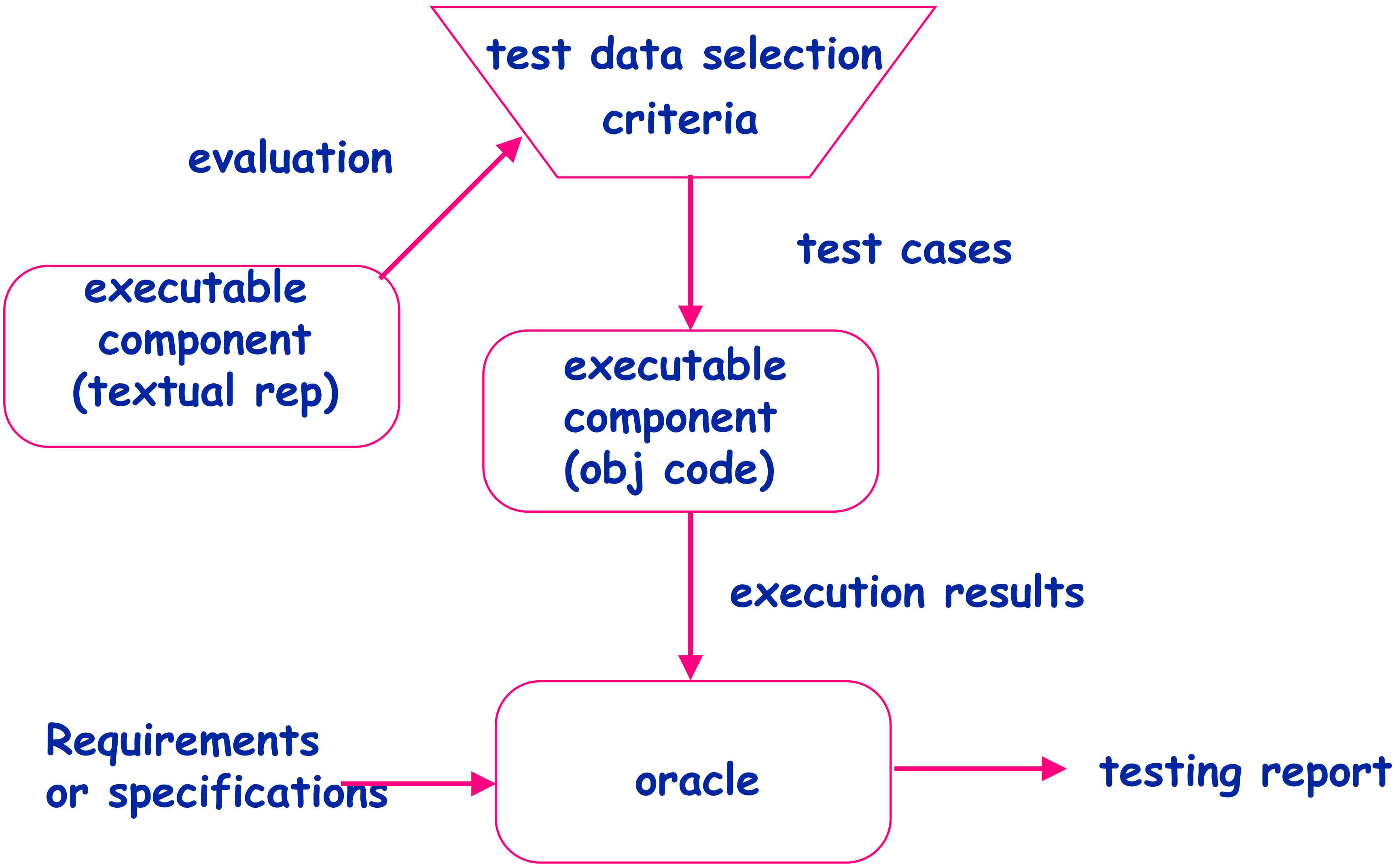
Approaches to testing

- **Black Box/Functional/Requirements based**

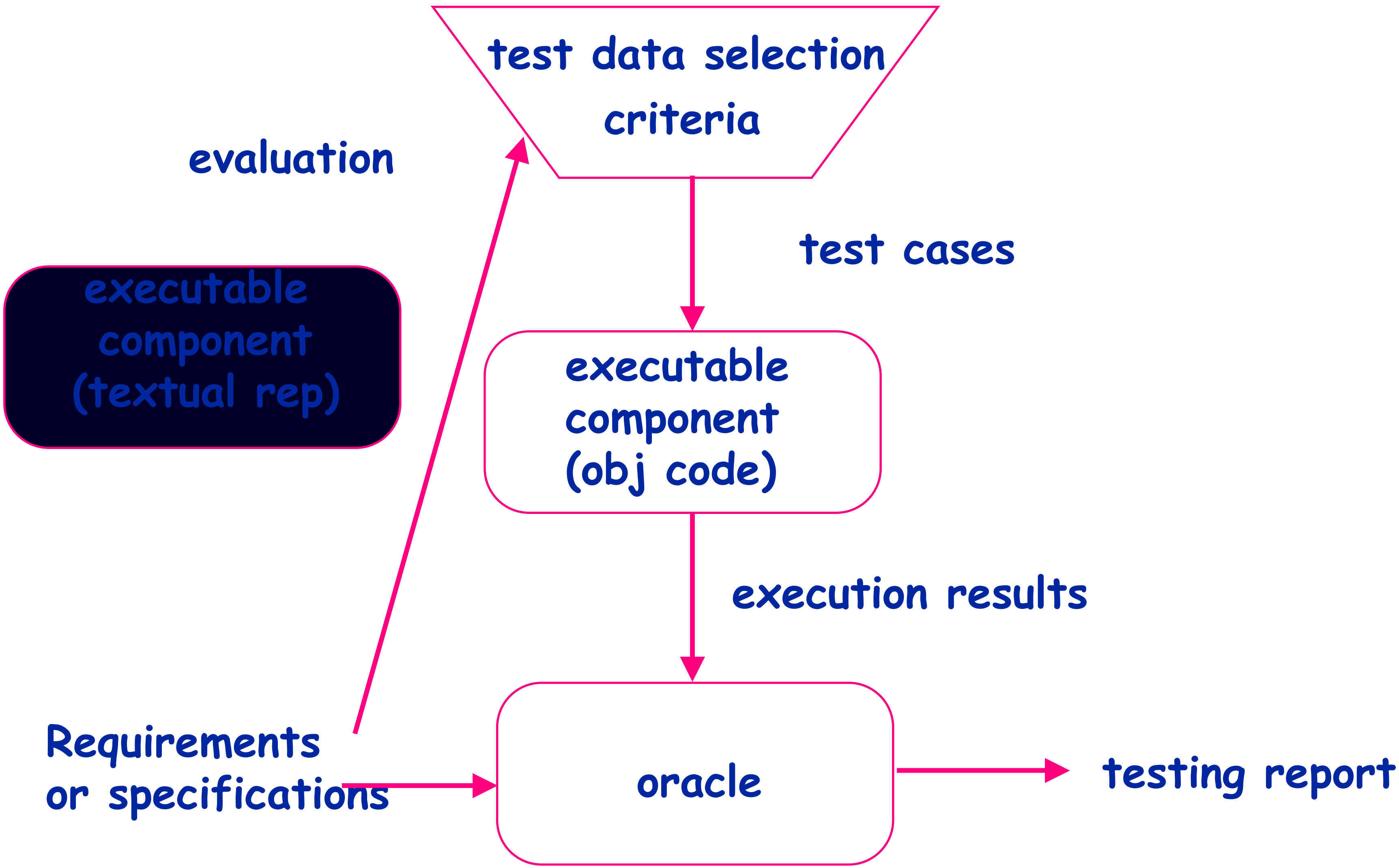
- **White Box/Structural/Implementation based**



White box testing process



Black box testing process



Why black AND white box?

- Black box
  - May not have access to the source code
  - Often do not care how s/w is implemented, only how it performs
- White box
  - Want to take advantage of all the information
  - Looking inside indicates structure=> helps determine weaknesses

Test Selection Criteria

- How do we determine what are good test cases?
- How do we know when to stop testing?

Test Adequacy

Test Selection Criteria

- A test set T is a finite set of inputs (test cases) to an executable component
- Let D(S) be the domain of execution for program/component/system S
- Let S(T) be the results of executing S on T
- A test selection criterion C(T,S) is a predicate that specifies whether a test set T satisfies some selection criterion for an executable component S.
- Thus, the test set T that satisfies the Criterion C is defined as:

$$\{\ t \in T \mid T \subseteq D(S) \text{ and } C(T, S)\ \}$$

Ideal Test Criterion

- A test criterion is ideal if for any executable system S and every $T \subseteq D(S)$ such that $C(T, S)$, if S (T) is correct, then S is correct
  - of course we want $T \ll D(S)$
  - In general, $T = D(S)$ is the only test criterion that satisfies ideal

In general, there is no ideal test criterion

"Testing shows the presence, not the absence of bugs"
E. Dijkstra

- Dijkstra was arguing that verification was better than testing
- But verification has similar problems
  - can't prove an arbitrary program is correct
    - can't solve the halting problem
  - can't determine if the specification is complete
- Need to use dynamic and static techniques that compliment each other

Effectiveness a more reasonable goal

- A test criterion *C* is *effective* if for any executable system S and every T $\subseteq$ D (S) such that C(T, S),

  $\Rightarrow$if S (T) is correct, then S is highly reliable

  OR

  $\Rightarrow$ if S (T) is correct, then S is guaranteed (or is highly likely) not to contain any faults of a particular type

- Currently can not do either of these very well
  - Some techniques (static and dynamic) can provide some guarantees

Two Uses for Testing Criteria

- **Stopping rule**--when has a system been tested enough
- **Test data evaluation rule**--evaluates the quality of the selected test data
  - May use more than one criterion
  - May use different criteria for different types of testing
    - regression testing versus acceptance testing

Black Box/Functional Test Data Selection

- Typical cases
- Boundary conditions/values
- Exceptional conditions
- Illegal conditions (if robust)
- Fault-revealing cases
  - based on intuition about what is likely to break the system
- Other special cases

Functional Test Data Selection

- Stress testing
  - large amounts of data
  - worse case operating conditions
- Performance testing
- Combinations of events
  - select those cases that appear to be more error-prone
  - Select 1 way, 2 way, … n way combinations

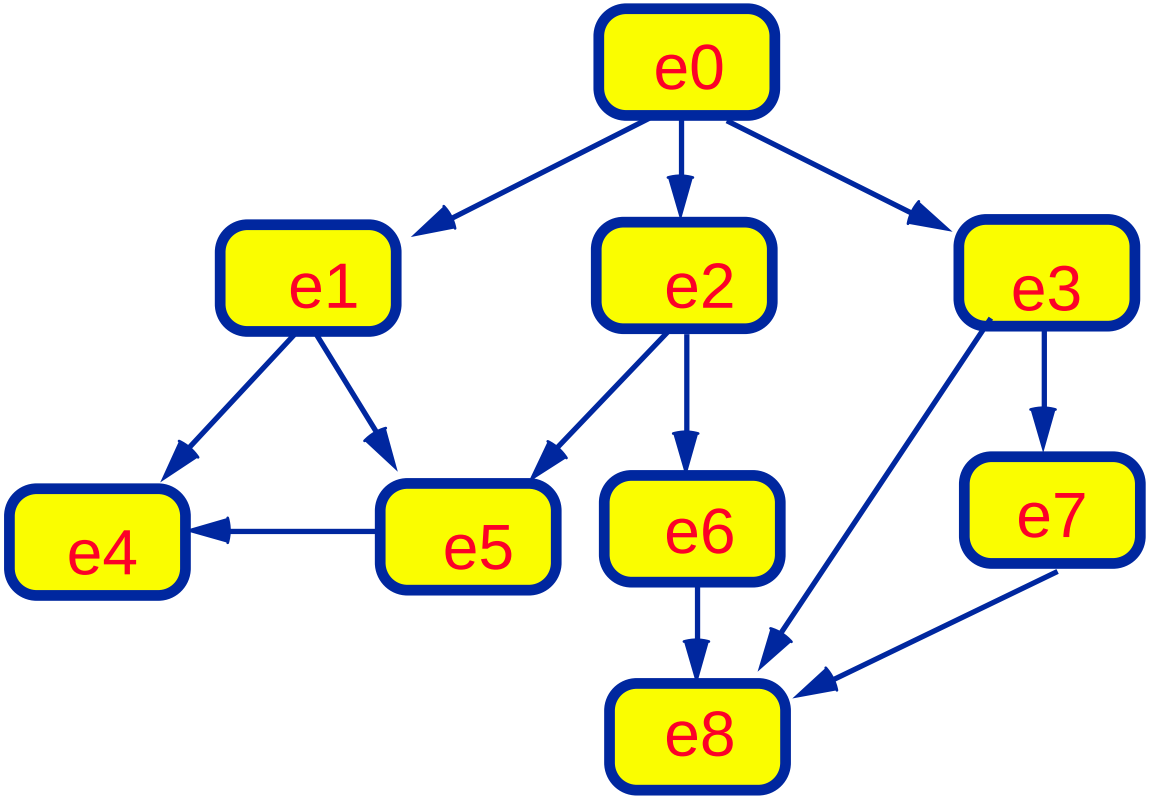
Sequences of events

- Common representations for selecting sequences of events
  - Decision tables
  - Cause and effect graphs
  - Usage scenarios

Decision Table

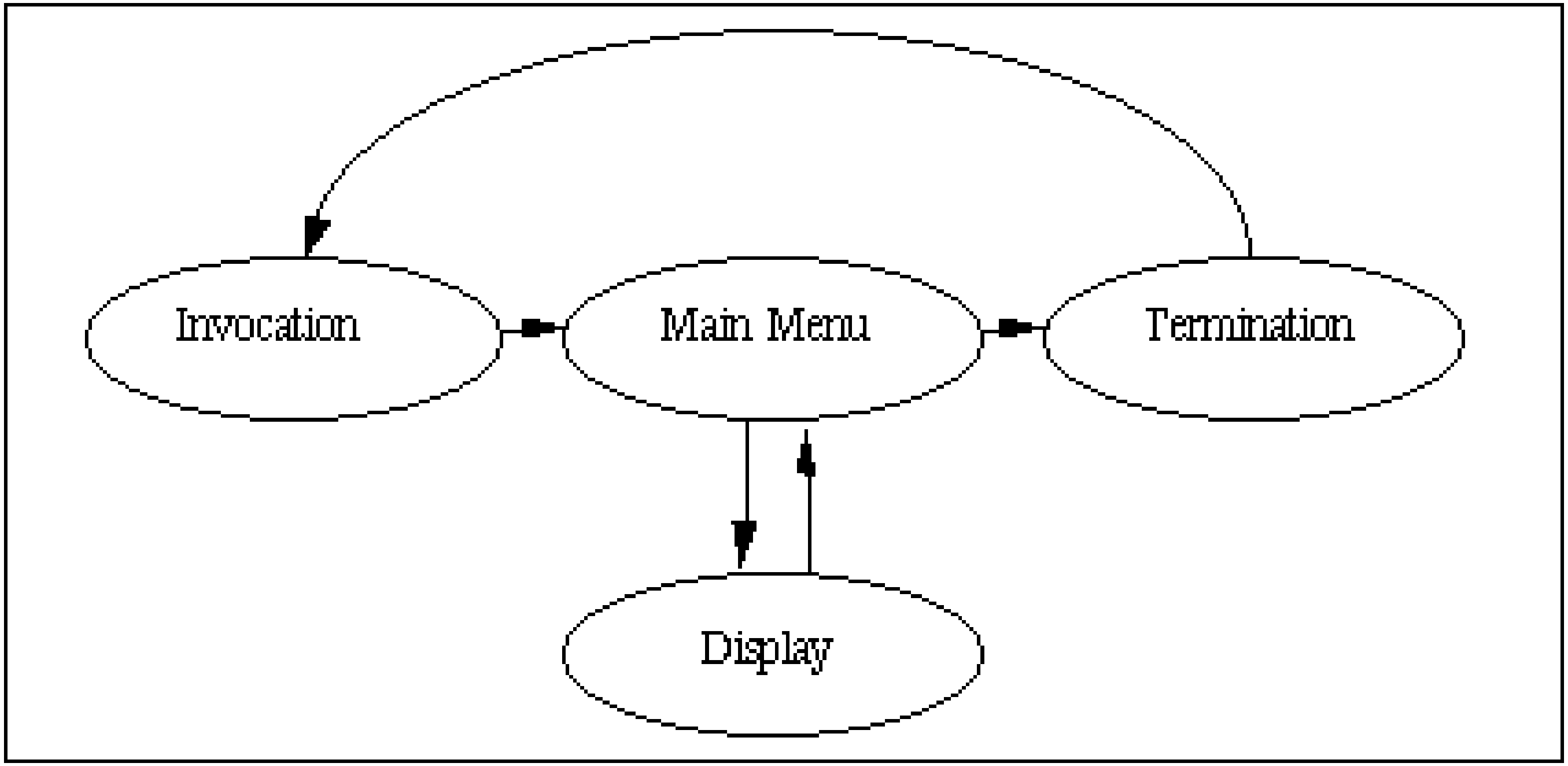
events	t1	t2	t3	t5	t6	t7	...
e1	x	x	x		-		
e2		x	x	x	x		x
e3	x			x		x	
e4	-		x		x		x
...		x			x	x	-

Cause and Effect Graph



Usage Scenarios

Graphical Usage Model of a Simple System



Overview of Dynamic Analysis Techniques

- Testing Processes
  - Unit, Integration, System, Acceptance, Regression, Stress
- Testing Approaches
  - Black Box versus White Box
- Black Box Strategies
  - Test case selection criteria
  - Representations for considering combinations of events/states

White Box/Structural Test Data Selection

- Coverage based
- Fault-based
  - e.g., mutation testing, RELAY
- Failure-based
  - domain and computation based
  - use representations created by symbolic execution

Coverage Criteria

- control-flow adequacy criteria
- G = (N, E, s, f) where
  - the nodes N represent executable instructions (statement or statement fragment)
  - the edges E represent the potential transfer of control
  - s ϵ N is a designated start node
  - f ϵ N is a designated final node
  - E = { (n_i, n_j) | syntactically, the execution of n_j follows the execution of n_i}

Control-Flow-Graph-Based Coverage Criteria

- Statement Coverage
- Branch Coverage
- Path Coverage
- Hidden Paths
- Loop Guidelines
  - General
  - Boundary - Interior

Statement Coverage

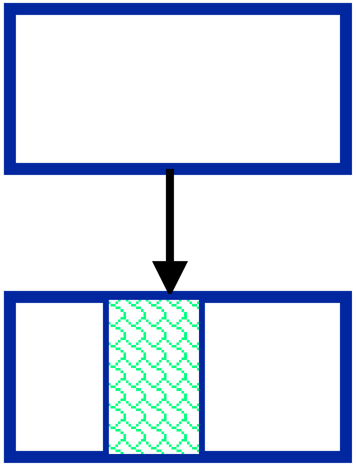
- requires that each statement in a program be executed at least once
- formally:
  - a set P of paths in the CFG satisfies the statement coverage criterion iff for each $n_i \in N$, $\exists\ p \in P$ such that n_i is on path p
    - defined in terms of paths

Statement Coverage

- only about 1/3 of NASA statements were executed before software was released (Stucki 1973)
- usually can achieve 85% coverage easily, but why not 100%?
  - unreachable code
  - complex sequence (should be tested!)
- Microsoft reports 80-90% code coverage

How does OO affect coverage?

- Often only parts of a reused component are actually executed by a system
  - Would expect good coverage for unit testing
  - More restricted coverage for integration testing



Coincidental Correctness

- Executing a statement does not guarantee that a fault on that path will be revealed

- Example:

  Y := X * 2

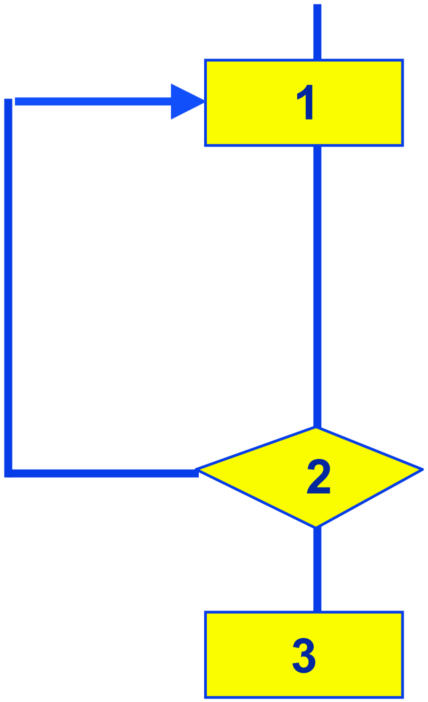
  Y := X ** 2

  If x = 2 then the fault is not exposed

Branch Coverage

- Requires that each branch in a program (each edge in a control flow graph) be executed at least once
  - e.g., Each predicate must evaluate to each of its possible outcomes
- Branch coverage is stronger than statement coverage

Branch Coverage



STATEMENT COVERAGE: PATH 1, 2, 3

BRANCH COVERAGE: PATH 1, 2, 1, 2, 3

Hidden Path (branch) Coverage

- Requires that each condition in a compound predicate be tested

  Example:

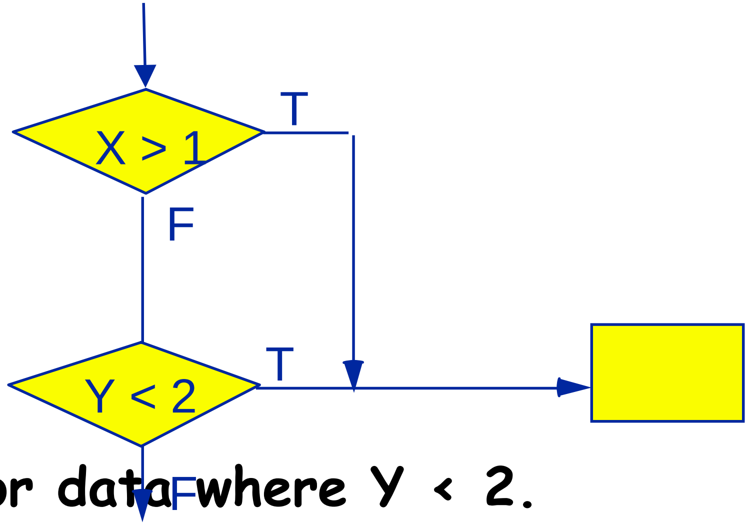
  $(X > 1) \vee (Y < 2)$

  Test Data:

  X = 2, Y = 5 ->T

  X = 1, Y = 5 ->F

  but, true branch is never tested for data where Y < 2.



(X > 1)	(Y < 2)
T	F
F	T
T	T
F	F

Path Coverage

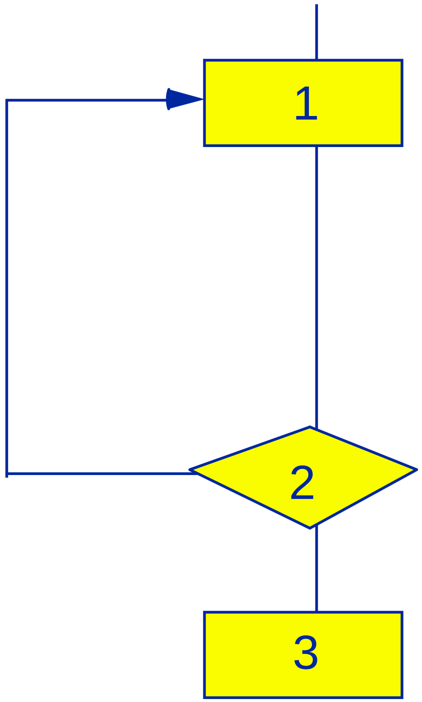
- Requires that every executable path in the program be executed at least once
- In most programs, path coverage is impossible
  - Example:

```
read N;
SUM :=  0;
for I = 1 to N do
    read X;
    SUM := SUM + X;
endfor
```

- How do we choose a set of paths?

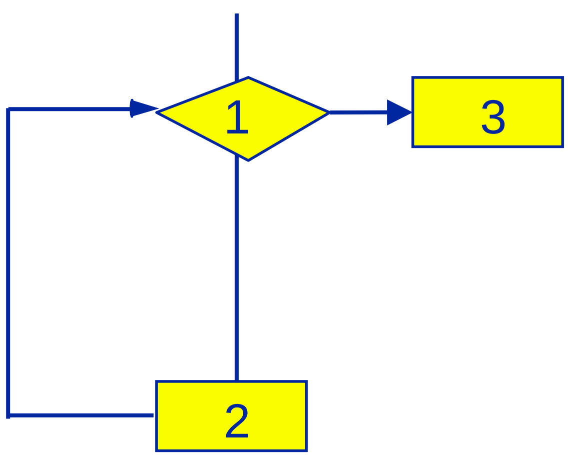
Loop Coverage

- Path 1, 2, 1, 2, 3 executes all branches (and all statements) but does not execute the loop well.



Typical Guidelines for loop coverage

- fall through case
- minimum number of iterations
- minimum +1 number of iterations
- maximum number of iterations



1, 3

1,2,3

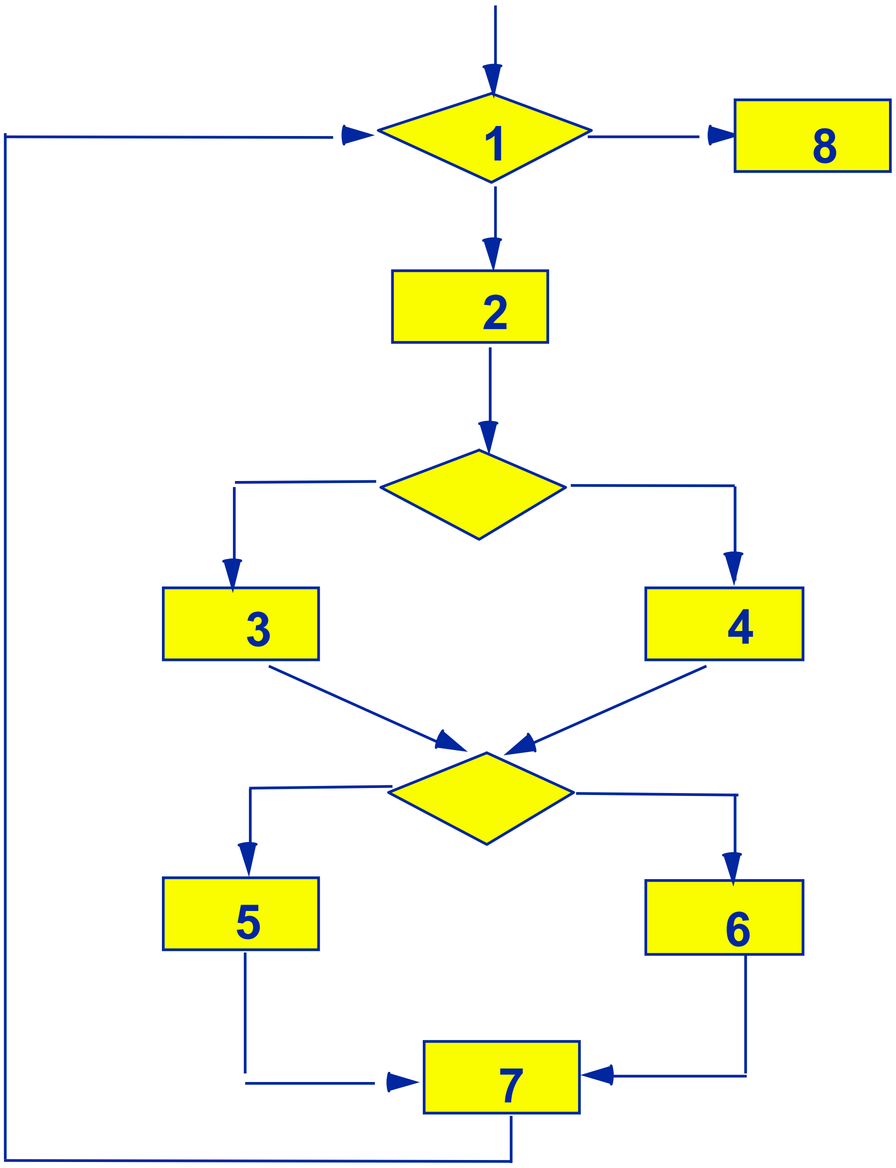
1,2,1,2,3

$(1, 2,)^n$ 3

Boundary - Interior Criteria

- boundary test of a loop causes the loop to be entered but not iterated
- interior test of a loop causes a loop to be entered and then iterated at least once
- both boundary and interior tests are to be selected for each unique path through the the loop

Example



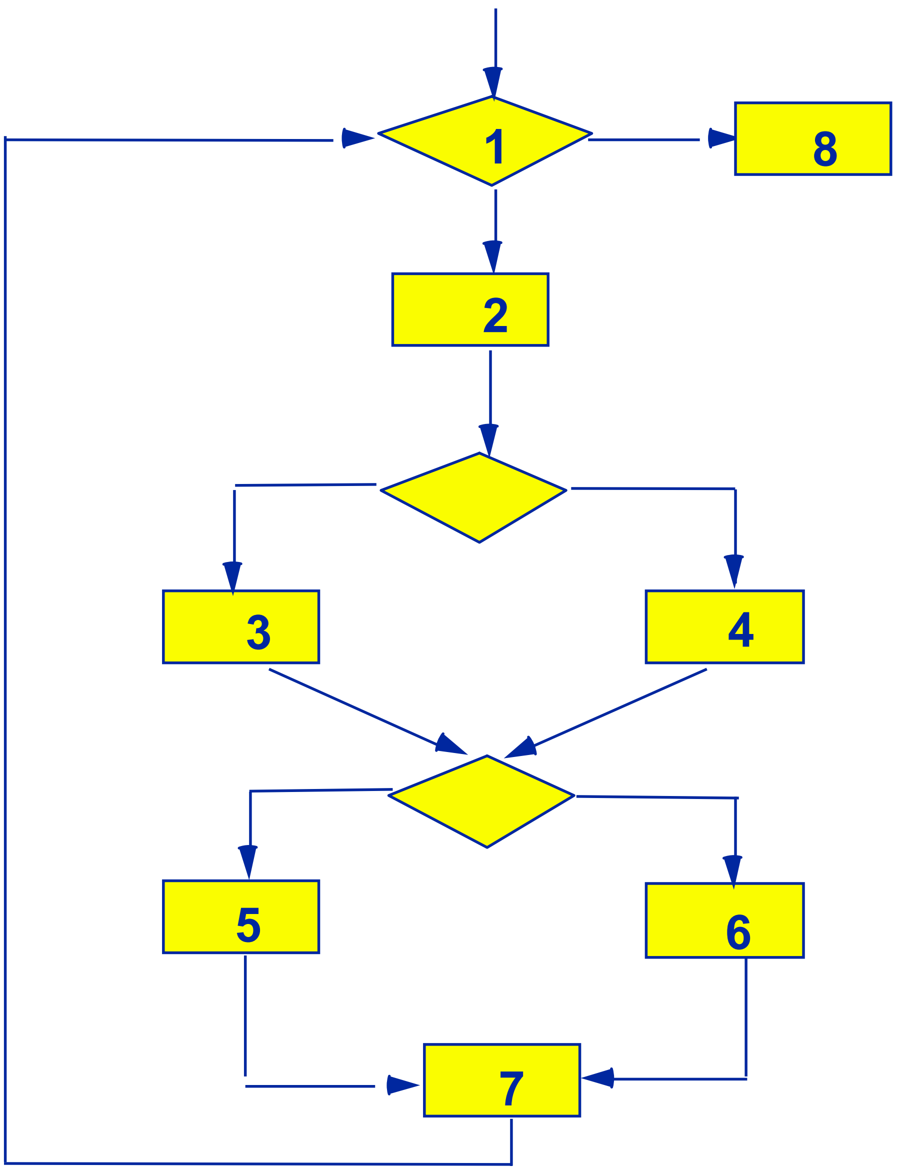
Paths for Example

Boundary paths

1,2,3,5,7	a
1,2,3,6,7	b
1,2,4,5,7	c
1,2,4,6,7	d

Interior paths
(for 2 executions of the loop)

a,a
a,b
a,c
a,d
b,a
b,b
...
x,y for x,y = a, b, c, d



Selecting paths that satisfy these criteria

- static selection
  - some of the associated paths may be infeasible
- dynamic selection
  - monitors coverage and displays areas that have not been satisfactorily covered

Problem with coverage criteria:

- Fault detection may depend upon
  - Specific combinations of statements, not just coverage of those statements
  - Astutely selected test data that reveals the fault, not just test data that executes the statement/branch/path
- Will look at semantically richer models
- First look at some axioms about testing criteria

Axiomatizing Software Test Data Adequacy

- Elaine Weyuker, Dec. 86, TSE
- Adequacy criteria for testing determines whether it is reasonable to stop testing
- Axioms are basic assumption that "well formed" criteria should satisfy
- A system that executes a test set T that satisfies an adequacy criterion is NOT necessarily correct
  - Correctness would be too strong
  - Only exhaustive testing would satisfy correctness

Weyuker's axioms

- for every system there exists an adequate test set [ADEQUACY]
  - Assuming that a system's domain is always finite, then the adequate test set is finite
- There is a system S and a test set T such that S is adequately tested by T, and T is not an exhaustive test set [NON-EXHAUSTIVE APPLICABILITY]
- If T is adequate for S and T is a subset of T', then T' is adequate for S [MONOTONICITY]

Weyuker's axioms

- the empty set is not adequate for any system [INADEQUATE EMPTY SET]
- let S be a renaming of Q, then T is adequate for S if and only if T is adequate for Q [RENAMING]
  - Superificial change does not change test cases

Weyuker's axioms

- if two systems compute the same function, a test set that is adequate for one is not necessarily adequate for the other [ANTI-EXTENSIONALITY]
  - Semantic equivalence does not preserve testing criteria
  - Implies that implementation must be taken into consideration
- if two systems are the same shape, a test set that is adequate for one is not necessarily adequate for the other [GENERAL MULTIPLE CHANGE]
  - Same shape means same CFG and same variables are referenced and defined at the nodes
  - Same values may not be computed

Weyuker's axioms

- for every n, there is a system S such that S is adequately tested by a set of size n, but not by any test set of size n-1 [COMPLEXITY]
  - Need at least n test cases
  - Any n test cases may not be adequate, however

Weyuker's axioms

- there exists a system S with a subcomponent Q such that T is adequate for S, T' is the set of vectors of values that variables can assume on entrance to Q and T' is not adequate for Q [ANTI-DECOMPOSITION]
  - S constrains the values that can be applied to Q and thus does not adequately test Q

Are these axioms?

- A principle that is accepted as true without proof as the basis for argument; a postulate (The American Heritage® Dictionary of the English Language, Third Edition copyright © 1992 by Houghton Mifflin Company. Electronic version licensed from InfoSoft International, Inc. All rights reserved)
- Want a set of axioms that are consistent and lead to theorems that provide insight
- Weyuker's "axioms" are not axioms, but desired properties
  - Showed that most testing criteria do NOT satisfy all these "axioms"

Stopping rule vs. Measurement

- C:(S,T) -> {*true, false*}   stopping rule
- C: (S,T) -> [0,1]   measurement

Zhu and Hall's Measurement Theory

- For all systems S and specifications R,
  - the adequacy of the empty test is 0
  - the adequacy of exhaustive testing is 1
  - If test set t1 is a subset of test set t2, then the adequacy of t1 is less than or equal to the adequacy t2 (monotonicity)