Chapter 20: Software Testing - White-Box, Coverage, Process

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

White-Box Testing Techniques

- Basis Path Testing
  - Flow Graph Notation
  - Cyclomatic Complexity
  - Deriving Test Cases
- Condition Testing
- Data Flow Testing
- Loop Testing
- Symbolic Execution

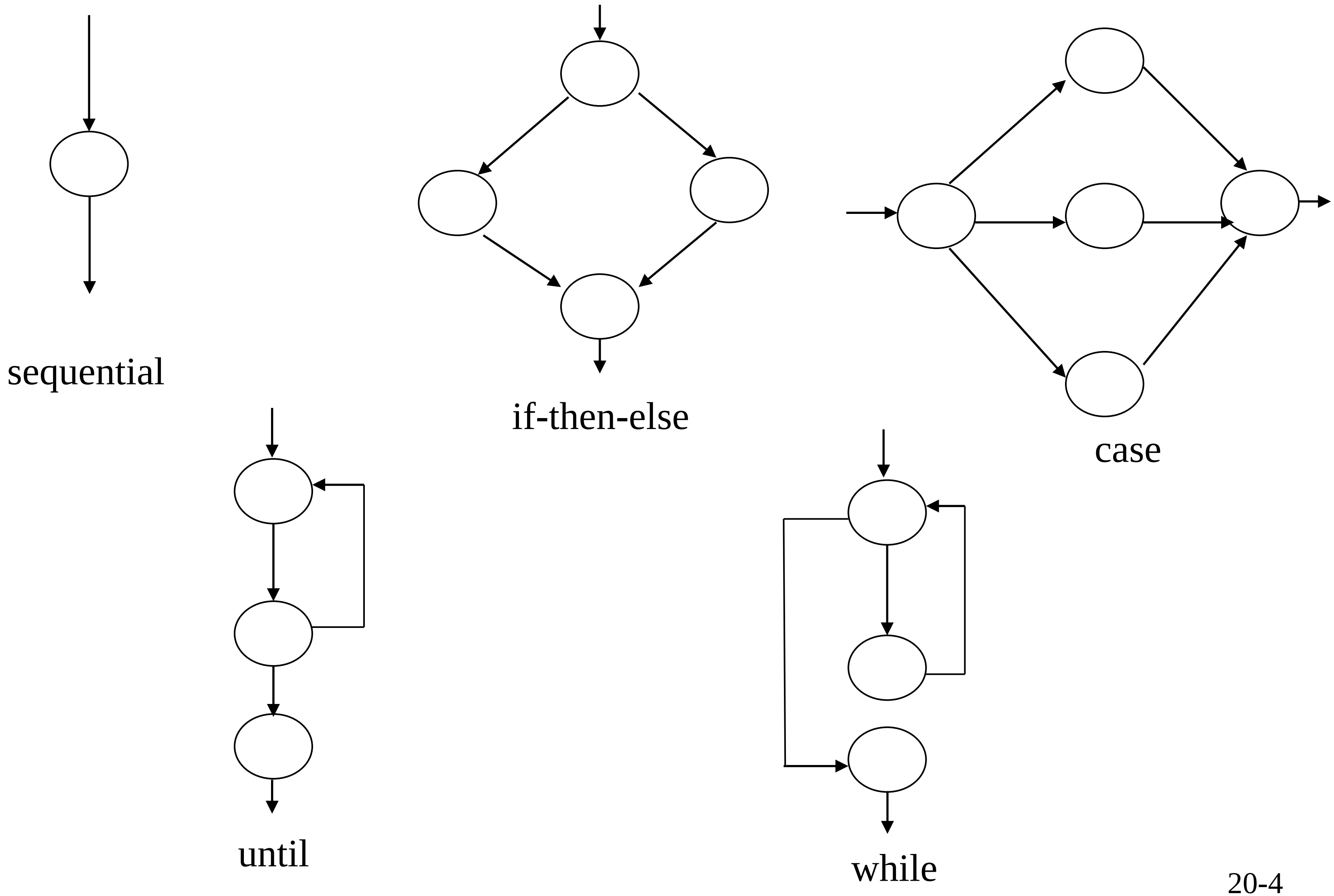
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Basis Path Testing Steps

- Construct a flow graph.
- Compute cyclomatic complexity.
- Determine basis paths.
- Check to ensure that the number of basis paths equals to the cyclomatic complexity.
- Derive test cases to exercise the basis paths according to the required coverage criteria.

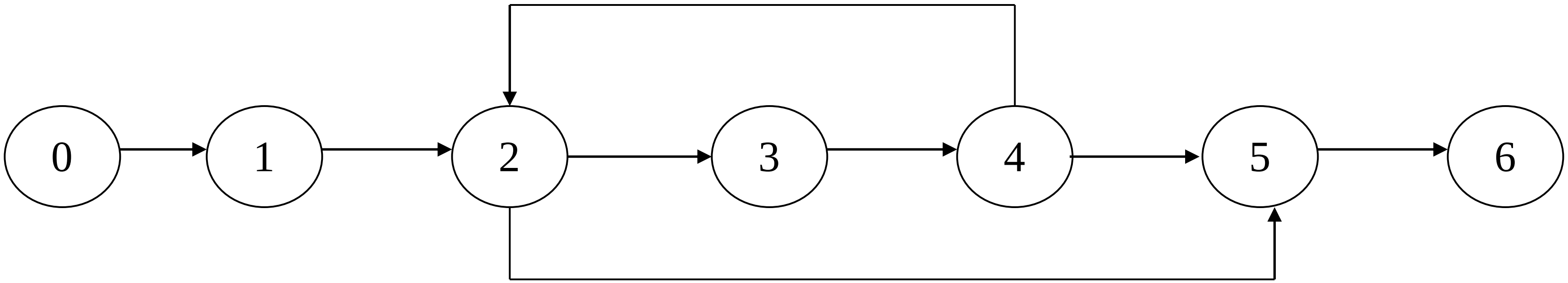
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Flow Graph Notations



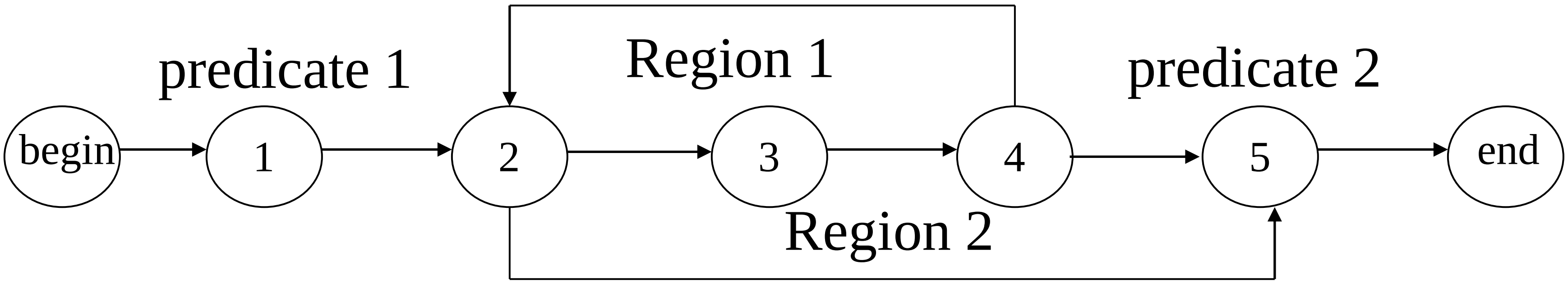
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

An Example Flow Graph



Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Cyclomatic Complexity



Three ways to compute cyclomatic complexity:

- Number of closed regions plus 1
- Number of atomic binary predicate + 1
- Number of edges - Number of Nodes + 2

The cyclomatic complexity is 2+1=3.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

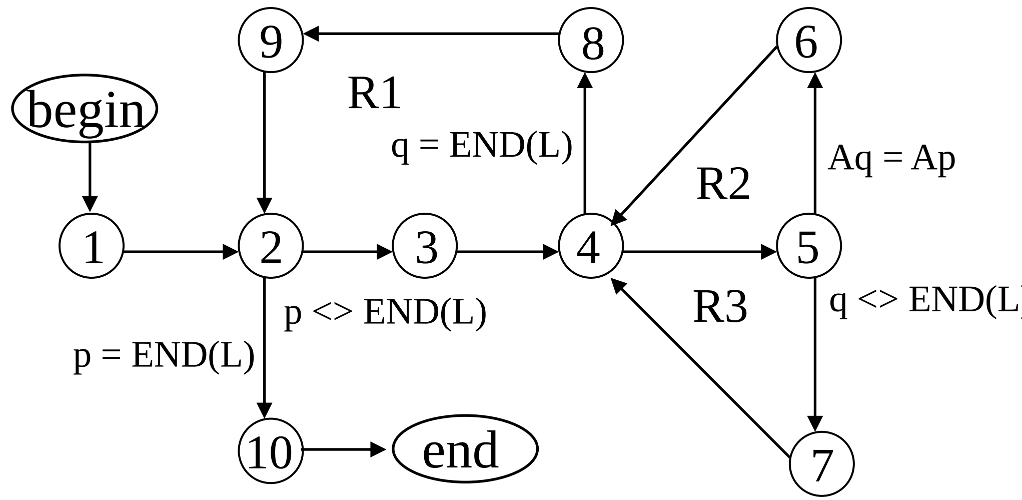
Determining Basis Paths

- The total number of test cases needed to exercise all basis paths equals to the cyclomatic complexity.
- A basis path is
  - a path from the begin node to the end node AND
  - traverses a cycle either zero times or exactly once.
- The basis paths are:
  - begin, 1, 2, 3, 4, 2, 5, end
  - begin, 1, 2, 3, 4, 5, end
  - begin, 1, 2, 5, end

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Example

```
Procedure purge (var L:list)
       var p: …
(1)     begin p:= FIRST(L);
(2)     while P <> END(L) do
(3)     begin q:= next(p,L);
(4)        while q <> END(L) do
(5)            if Aq = Ap then
(6)               delete (Aq, L)
(7)            else q:= next(q,L);
(8)        end
(9)        p := next(p,L)
(10)    end;
```



Path 1 (R1): 1-2-3-4-8-9-2-10

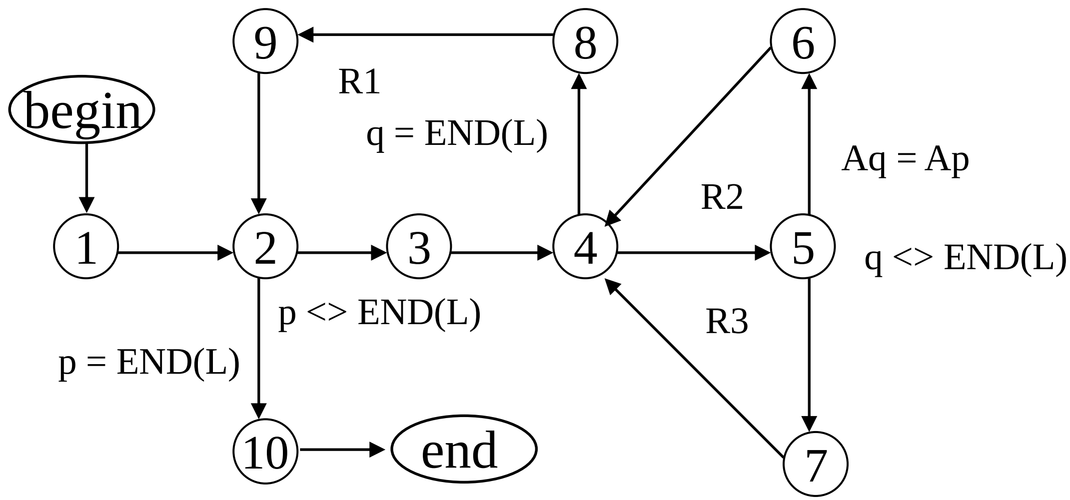
Path 2 (R2): 1-2-3-4-5-6-4-8-9-2-10

Path 3 (R3): 1-2-3-4-5-7-4-8-9-2-10

Path 4: 1-2-10

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Example



Path 1 (R1):	1-2-3-4-8-9-2-10	L=(Ap)	
Path 2 (R2):	1-2-3-4-5-6-4-8-9-2-10	L=(Ap Aq)	Ap = Aq
Path 3 (R3):	1-2-3-4-5-7-4-8-9-2-10	L=(Ap Aq)	Ap <> Aq
Path 4:	1-2-10	L=( )	

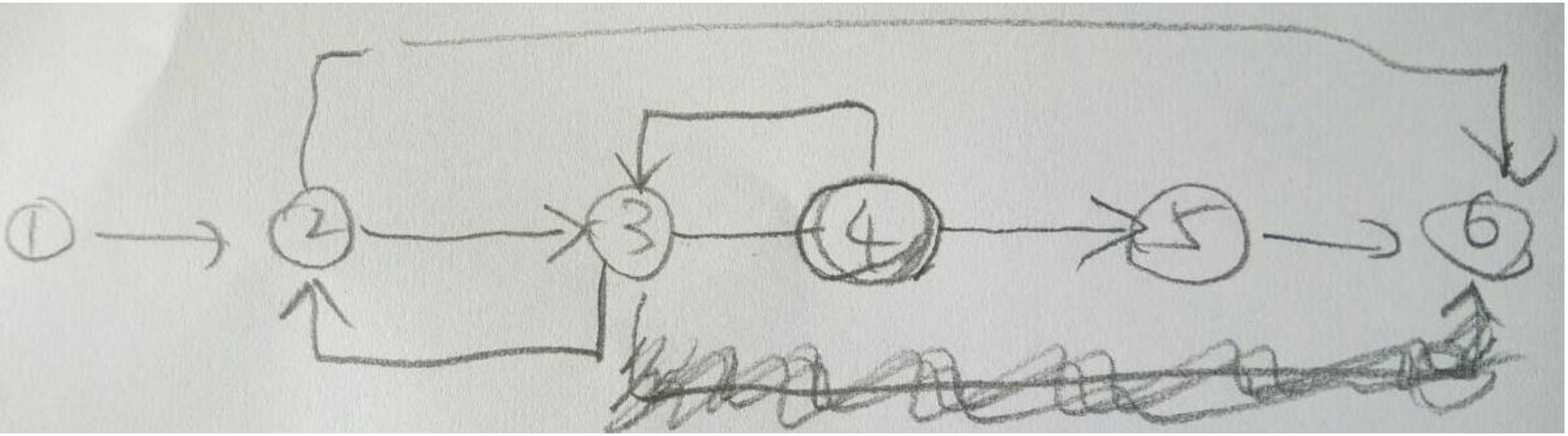
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Class Exercise: Basis Path Testing

```
(1) public void bubbleSort(int n)
(2) { for(int j=1; j<n; j++)
(3)       for(int i=n-1; i>=j; i--)
(4)         if (a[i]>a[i+1])
(5)            swap (a[i], a[i+1]);
(6) }
```

- Draw flow graph
- Compute cyclomatic complexity
- Derive basis paths
- Derive test cases

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.



Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Condition Testing

- Focus on testing the logical conditions in the program.
- Errors detected:
  - incorrect, missing, or extra boolean operators
  - boolean variable error
  - parenthesis error
  - relational operator (==, !=, >, <, >=, <=) error
  - arithmetic expression error

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Condition Testing Techniques

- Branch testing: exercise the true and false branches of a compound condition C and every simple condition in C at least once.
- Domain testing:
  - if condition is e1 <rel_op> e2 then test cases are val(e1) > val(e2), val(e1) == val(e2), val(e1) < val(e2)
  - if condition is b1 & b2 then test cases are (t,t), (t,f), (f,t)
  - if condition is b1 or b2 then test cases are (f,f), (f,t), (t,f)

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Condition Testing Techniques: 為何(t,f) (f,t)都要執行? 測試程式內部的正確性!

```
public void putSeed (int seed)
{
    if (seed >= 0 && seed < c.length)
        c [seed] = null;
}
```

Test case (t,t): seed >= 0 && seed < c.length

Test case (t,f): seed >= 0 && seed >= c.length

Test case (f,t): seed < 0 && seed < c.length

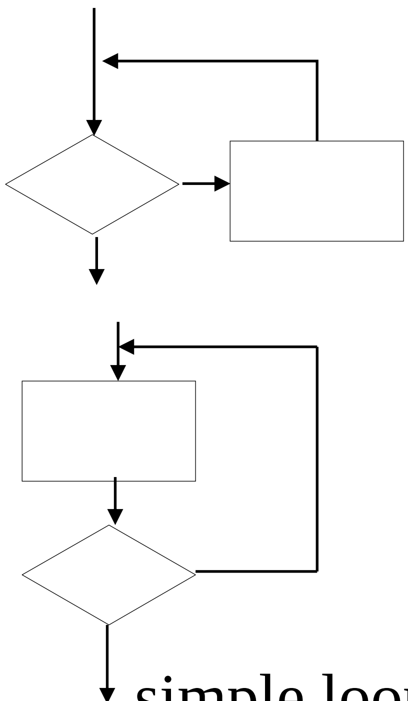
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Loop Testing

- focuses on the validity of loop constructs.
- four different classes of loops:
  - simple loop: a single loop
  - nested loops: a loop within another loop
  - concatenated loops: one loop after another loop
  - unstructured loops: complex nested and concatenated loops.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Testing Simple Loops

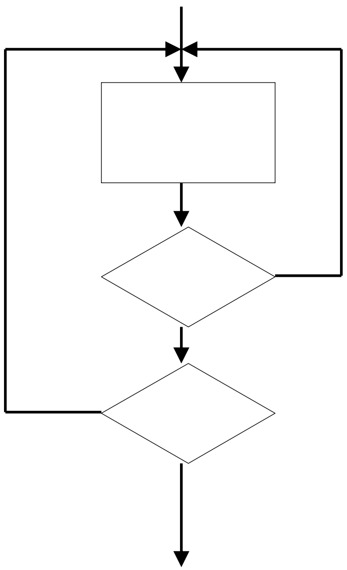


simple loops

1. Skip the loop entirely.
2. Only one pass through the loop.
3. Two passes through the loop.
4. m passes through the loop where m < n.
5. n -1, n, n + 1 passes through the loop.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Testing Nested Loops



nested loops

Beizer's approach reduces the number of tests:

1. Start at the innermost loop. Set all other loops to minimum values.
2. Conduct simple loop tests for the innermost loop:
   - holding the outer loops at their minimum iteration parameter.
   - add other tests for out-of-range or excluded values.

(to be continued)

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

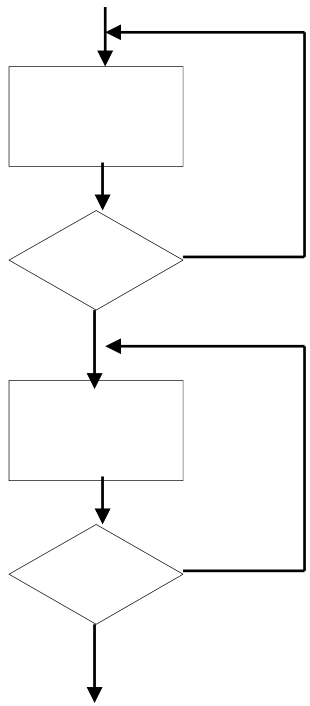
Testing Nested Loops

3. Work outward, conducting tests for the next loop:
   - keeping all other outer loops at minimum values and
   - other nested loops to "typical" values.
4. Continue until all loops have been tested.

先讓最內部的loop複雜化，但單純化外面的。
再用不同test case逐步往外複雜化

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Testing Concatenated Loops



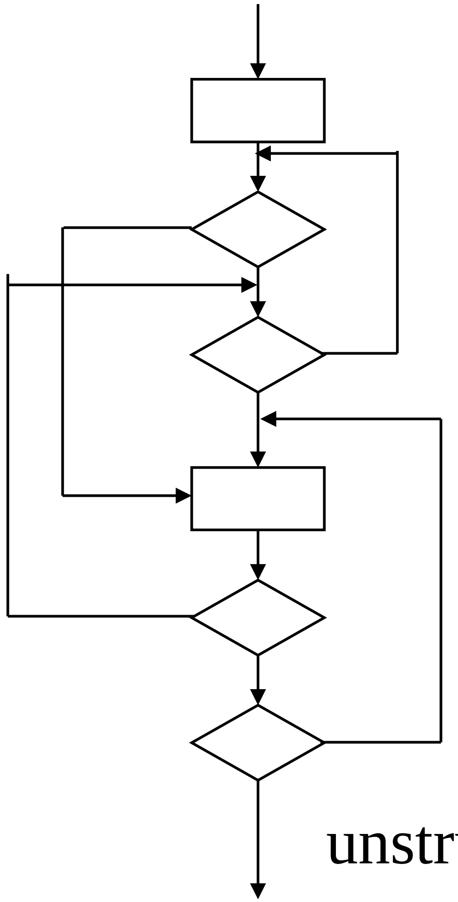
concatenated loops

• if each of the loops is independent of the other then test them as simple loops.

• if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then test them as nested loops.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Testing Unstructured Loops



unstructured loop

Redesign the algorithm to remove loops.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Symbolic Evaluation (看看就好，太複雜了，這是數學家在看測試!!)

- Execute a program using symbolic values rather than numeric values to validate the program path by path.
- The result of symbolic execution is a set of logical expressions representing the paths through the program.
- Each logical expression corresponds to a test case.
- A constraint solver can be used to derive the test data – values that satisfy the path condition.
- There exists symbolic evaluator software, also called a symbolic executor.

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Symbolic Evaluation

```
Procedure purge (var L:list)
    var p: …
    begin p:= FIRST(L);
    while p <> END(L) do
    begin q:= next(p,L);
      while q <> END(L) do
        if Aq = Ap then
          delete (Aq, L)
        else q:= next(q,L);
      end
      p := next(p,L)
    end;
```

Symbolic execution result:

$FIRST(L)=END(L) \Rightarrow L=L$

$FIRST(L) <> END(L) \wedge$

$NEXT(FIRST(L),L)=END(L) \Longrightarrow L=L$

$FIRST(L) <> END(L) \wedge$
$NEXT(FIRST(L),L)<>END(L) \wedge$

$FIRST(L)=NEXT(FIRST(L),L) \Longrightarrow$
$L=DELETE(NEXT(FIRST(L)))$

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Symbolic Execution & Correctness Proof

```
x <-- a; // a and be are
y <--b; // symbolic values
if (x > = y) then
   max <-- x
else
   max <-- y
```

a > = b ==> max = a

a < b ==> max = b

Symbolic execution

```
pre: TRUE
if (x >= y) then
   max <--- x
else
   max <--- y
post:
 (max >= x) & (max >= y)
```

Prove: if (pre) and after executing program, then post is true.

correctness proof

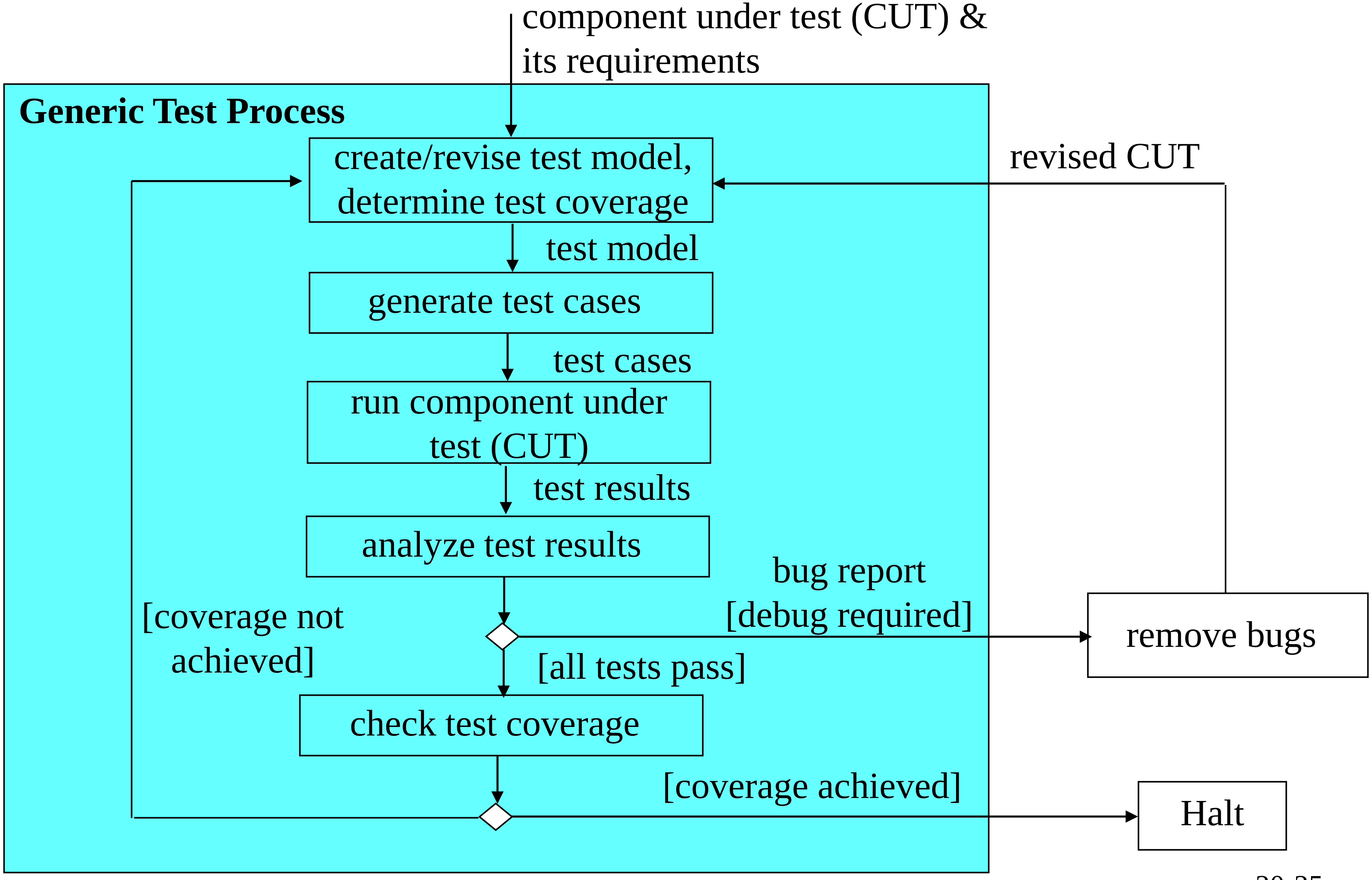
Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Test Coverage

- A test coverage is a quality goal to be accomplished, as well as a measurement of the accomplishment of the goal.
- Examples:
  - 100% branch coverage (to be accomplished)
  - 100% branch coverage (accomplished)
  - 100% requirements coverage (to be accomplished as well as actually accomplished)

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

A Generic Testing Process



Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.

Usefulness of Generic Process

- It facilitates understanding of test methods because test methods follow the generic process.
- It describes the steps for software testing.
- It serves as a guide for introducing and implementing software testing in an organization.
- Class exercise: Discuss why above are important?

Copyright {c} 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.