Lecture: WHITE-BOX TESTING

PURPOSE: To ensure that all parts of software have been executed during the testing process.

CONCEPT: Test the STRUCTURE of the software. Make sure every part works correctly.

Example #1: function Employee_Pay

Synopsis: Compute pay for employee. Overtime is paid at 1.5 times the hourly pay rate for hourly employees (who make less than $20 per hour). Salaried employees (who make $20 per hour or more) are paid for exactly 40 hours, even when more or fewer hours are worked.

[Node] Code
1  float Employee_Pay(Rate, Hours)
2     begin
3       if (Rate < 20.00)
4          then (* c1 *)
5           if (Hrs > 40)
6              then (* c1 & c2 *)
7                  Pay = 40 * Rate +
8                      1.5 * Rate * (Hrs-40)
9              else (* c1 & !c2 *)
10                 Pay = Rate * Hours
11           else (* !c1 *)
12              Pay = 40 * Rate;
13     end;

WHAT IS MEANT BY STRUCTURE OF THIS PROGRAM?

- Control flow constructs in programming languages
  - sequence
  - selection (if, switch)
  - iteration (pretest/posttest loops)
  - gotos
IDENTIFICATION OF PROGRAM BLOCKS/EDGES:

0. Create blocks in the top-to-bottom flow of the source code. Let p denote the previous block.
1. The unit header/prototype is not assigned a block number.
2. The open brace for the body of the unit is block 1.
3. Each control structure (if, switch, for, while) is a new block y. Create edge (p,y).
4. Each case of a switch begins a new block y. Create edge (x,y), where x is the node for switch.
5. The consequence of an if begins a new block y.
   a) in C/C++/Java: the block begins AFTER the right parenthesis containing the if condition. Create edge (x,y), where x is the node for if.
   b) in Pascal/VB: the "then" keyword begins the new block y. Create edge (x,y), where x is the node for if.
6. Each else begins a new block y. Create edge (x,y), where x is the node for if.

MERGING BLOCKS:

Two consecutive (adjacent) blocks can be merged when the first (smaller number) block always flows into the second block AND there is no other edge flowing into the second block (i.e., "b2 iff b1").

DEPICTING STRUCTURE: Control Flow Graph for Employee_Pay

```
+---------------->--------------+
   |                               |
!c1 |       +--------->-----+       |
   |       | c1 & !c2      |       |
   |       +-------------->|       |
   |                       |
+---------------->------+
```

Edge List: { (1,2), (1,5), (2,3), (2,4), (3,6), (4,6), (5,6), (6,7) }
A. Control Flow Graph: Nodes + Edges

1. Four kinds of nodes:
   a) block = sequence of statements in the code is represented as a single node
   b) decision node = a decision point in program at which flow deviates from sequential flow. The following constructs require a decision node: if statement, switch, loop. (See nodes 1, 2).
   c) terminal node = a node that does not have a successor.

2. Edges show flow of control
   a) All edges are DIRECTED (have arrows)
   b) Decision node has OUTDEGREE > 1, i.e., two or more edges leave the decision node. Each edge leaving a decision node is called a BRANCH.

3. Path: Sequence of nodes from initial node 1 to terminal node.
   a) There are multiple paths from entry (b) to exit (e).
   b) A sequential unit has exactly 1 path from entry to exit.
   c) Each decision increases the number of paths by 1.
   d) A loop may make the number of paths infinite.
   e) Concept of basis path applies to the branches in a loop:
      - Loop TERMINATION branch + loop CONTINUATION branch.
      - Basis testing condition = loop is tested when each branch has been executed during testing.

4. Making it Visual - COLORING THE GRAPH

   White box testing can be measured by CHECKING OFF each node and edge in the control flow graph as it is traversed, and coloring each node as it is visited for the first time. When all edges have been checked off, BASIS PATH Coverage has been achieved. When all nodes have been visited, statement coverage has been achieved.

5. Measures of Testing Progress - COVERAGE

   1) statement coverage -- all statements in a unit are executed during testing:
      => every node in control flow graph is reached.

   2) edge coverage -- each edge is traversed at least once during testing.
      => each decision branch is traversed.

   NOTE: Edge.cov => Statement.cov
3) branch coverage -- each branch condition outcome (true/false) is executed at least once during testing.

   => each decision branch is traversed.

   NOTE: Branch.cov => edge.cov

4) condition coverage -- at each decision (possibly representing a compound (complex) condition, all FEASIBLE combinations of condition outcomes are executed at least once during testing:

   => analysis of logical conditions required

   NOTE: condition.cov => branch.cov

   NOTE: this step forces simplification of complex conditions.

5) path coverage -- each path through the from the start node (b) to the end node (e) is traversed.

   NOTE: Path.cov => Branch.cov

   NOTE: with loops, the number of paths is infinite, since each iteration can increase #paths exponentially.

6) basis path coverage (feasible) -- each edge of a the unit's control flow graph has been traversed at least once during testing:

   => practical way of testing, based on control flow graph.

--------------------------------------------------------

COVERAGE ANALYSIS: Employee_Pay, set of two test cases.
--------------------------------------------------------

<table>
<thead>
<tr>
<th></th>
<th>Coverage</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>node coverage</td>
<td>5/7</td>
</tr>
<tr>
<td></td>
<td>edge coverage</td>
<td>4/8</td>
</tr>
<tr>
<td></td>
<td>branch coverage</td>
<td>0/2</td>
</tr>
<tr>
<td></td>
<td>condition coverage</td>
<td>0/2</td>
</tr>
<tr>
<td></td>
<td>path coverage</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Case 1: Rate=15, Hours = 30: Path = [1,2,4,6,7]

<table>
<thead>
<tr>
<th></th>
<th>Coverage</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>node coverage</td>
<td>5/7</td>
</tr>
<tr>
<td></td>
<td>edge coverage</td>
<td>3/8</td>
</tr>
<tr>
<td></td>
<td>branch coverage</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>condition coverage</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>path coverage</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Case 2: Rate=15, Hours = 50: Path = [1,2,3,6,7]
COLORING THE CONTROL FLOW GRAPH:

Traversed Edges List: { (1,2), (2,3), (2,4), (3,6), (4,6), (6,7) }

EXAMPLE #2: function Make_Up

Consider the specification: Write a program segment which, given an initial exam score $E_1$, and a make-up score $E_2$, will assign an exam score as follows:

1. use the initial score if it was 90 or above;
2. use 93 if the make-up score exceeds 95;
3. use the make-up score when it is no greater than 95 and the initial score was less than 90.

Code:

```plaintext
[Node] Code
1  function Make_Up(E1, E2) : integer;
[1] 2     begin
[2] 3     if (E1 < 90)  (* c1 *)
[3] 4     then (* c1 & !c2 *)
[4] 5     if (E2 > 95)  (* c1 & c2 *)
[5] 6     then (* c1 & !c2 *)
[6] 7     else (* c1 & !c2 *)
[7] 8     else // !c1  Score := 93
[8] 9     Score := E2;
[9] 10    else // !c1  Score := E1;
[10] 11  end; (* Make_Up *)
```
A. CONTROL FLOW GRAPH: Make_Up

+---------------->--------------+
|       | c1 & !c2      |       |
|       |               v       v
----> 1 ----> 2 ----> 3       4       5 ----> 6
| c1       c1   |       |               |
| & c2   |       |               |
|       +-------------->^|
|                       |
| +-------->--------------+

Edge List: { (1,2),(1,5),(2,3),(2,4),(3,6),(4,6),(5,6) }

B. Decision Analysis – Build Decision Table from Code Logic

1) Decision Analysis - look at the decisions made, and the combination of logical outcomes. For each combination, list the actions taken by the algorithm (program). Create inputs to force each decision to be made.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Logical Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 &lt; 90</td>
<td>Y</td>
</tr>
<tr>
<td>E2 &gt; 95</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score := E1</td>
</tr>
<tr>
<td>Score := E2</td>
</tr>
<tr>
<td>Score := 93</td>
</tr>
</tbody>
</table>

C. COVERAGE ANALYSIS for Make_Up: 2 test cases

Cumulative

Case 1: E1=80, E2 = 97: Path = [1,2,3,6]

<table>
<thead>
<tr>
<th>Coverage</th>
<th>Node coverage</th>
<th>Edge coverage</th>
<th>Branch coverage</th>
<th>Condition coverage</th>
<th>Path coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5/5</td>
<td>3/7</td>
<td>0/2</td>
<td>0/2</td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>4/6</td>
<td>3/7</td>
<td>0/2</td>
<td>0/2</td>
<td>1/3</td>
</tr>
</tbody>
</table>
Case 2: E1=90, E2 = 50: Path = [1,5,6]

node coverage = 3/6  5/6
edge coverage = 2/7  5/7
branch coverage = 1/2  1/2
condition coverage = 1/2  1/2
path coverage = 1/3  2/3

COLORING THE CONTROL FLOW GRAPH:

[n] ==> node n was reached
* ==> edge traversed

Traversed Edge List: { (1,2),(1,5),(2,3),(3,6),(5,6) }
Loop testing focuses on reasonable circumstances related to entering, exiting, and repeating the loop, including the following:

(1) Zero iterations, i.e., the loop is never entered.
(2) Exactly one iteration - this establishes that the loop can be entered and terminated.
(3) One fewer than the maximum number of iterations (e.g., FOR loops).
(4) The maximum number of iterations (not always practical).

Any control structures contained inside the loop are tested as previously described.

A. SOURCE CODE

```c++
[Node] Code
1  void AddScores(ifstream & inF, ofstream & outF)
2  {
3    int Sum, Score;
4    inF >> ID;
5    while (ID != 0)
6      {
7        Sum = 0;
8        inF >> Score;
9        while (Score >= 0)
10           {
11          Sum = Sum + Score;
12          inF >> Score;
13      }//while Score
14      outF << ID << Sum << endl;
15     inF >> ID;
16  }//while ID
17  }//AddScores
```

B. CONTROL FLOW GRAPH: AddScores

(a) Graphic
C. TEST CASE DESIGN CRITERIA

Strategy:

1) Outer Loop Iterations: [OT1] 0; [OT2] 1; [OT3] 2.
2) Inner Loop Iterations: [IT1] 0; [IT2] 1; [IT3] >2.

Combination of Criteria:

<table>
<thead>
<tr>
<th>OT1</th>
<th>OT2</th>
<th>OT3</th>
<th>IT1</th>
<th>IT2</th>
<th>IT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

D. TEST CASE DESIGN

Test Sets:

TS1 = {Data[ID=0, Score={}], Exp={{}]]
TS2 = {Data[ID=2740, Score={3, -1}], Exp=[ID=2740, Sum=3]} 
TS3 = {
    (Data[ID=2741, Score={-1}], Exp=[ID=2741, Sum=0>]),
    (Data[ID=542, Score={5 4 2 -1}], Exp=[ID=542, Sum=11])
}

E. COVERAGE ANALYSIS for AddScores

1. Test Set #1:

Case 1: {Data[ID=0, Score={}}}

Path = [1,2,7]

<table>
<thead>
<tr>
<th>Coverage Type</th>
<th>Test Case</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>node coverage</td>
<td>3/7</td>
<td>3/7</td>
</tr>
<tr>
<td>edge coverage</td>
<td>2/8</td>
<td>2/8</td>
</tr>
<tr>
<td>branch coverage</td>
<td>0/2</td>
<td>0/2</td>
</tr>
<tr>
<td>condition coverage</td>
<td>0/2</td>
<td>0/2</td>
</tr>
<tr>
<td>path coverage</td>
<td>1/3</td>
<td>1/3</td>
</tr>
</tbody>
</table>
COLORING THE CONTROL FLOW GRAPH:

[n] ==> node n was reached
* ==> edge traversed

(b) Traversed Edges List = { (1,2), (2,7) }