**4. White Box Testing (implementation-oriented testing, developer testing)**

control structure based testing: control structure graphs for sequential programs, Petri nets for concurrent programs;   
coverage measures: node, branch, path coverage;   
several condition coverages: simple/multiple/minimal multiple condition coverage, MC/DC;  
code instrumentation for coverage measures;  
data flow based testing: defs/uses techniques;

4.1 control structure based testing: control structure graphs for sequential programs, Petri nets for concurrent programs;

Testing according to control structure: (Control flow?)

Note: it is not the control flow by the control structure.

Basis:

1. Graphical representation of the control structures

Programming flow charts.

Condition: b1 for **while** and b2 for **if**.

get

Strat

Count

stop

get

if b2

vocals ++

total ++

While b1

0 ( false)

1 ( true)

0 ( false)

1 ( true)

1. Control structure (flow) graph(CSG)
   1. Finite, Directed
   2. 1, entry noe, & 1 exit node. (no out go it, arc is a property)
   3. 1 statement per node
   4. Arcs = control flow

Note: we abstract the exact branching conditions;

1. Petri Nets (PN)

**Basic Assumption:**

Set of structural path is a good approximation for the set of path actually possible when considering all data dependency among branching conditions.

Example:

if b1

then a1

else a2

endif

if b2

then a3

else a4

endif

**Important point:**

**Assumption, b1 = b2 [for simplification]**

**Note: here, set of path actually possible 4, as data dependency.**

**2. CSG: (Control structure graph)**

**Strat**

get

while b1 0 **end**

1

total++

if b2

1 0

Empty Branch

vocals++

get

**Program Flow Graph (Control Flow Diagram)**

Flow graph notation defines the program using nodes connected by edges. The following shows some examples for flow graph:

|  |  |
| --- | --- |
| **Sequential statement block -** | **If else block –** |
| **Switch case block** | **If- then block** |
| **While block –** | **Loop** |
| **Do-loop –** | **!! Don’t forget:**  Empty branches:   1. empty else of if structure 2. empty else of switch structure 3. do while |

**3. Petri Net:**

**Strat**

get

exit while

while b1 0 **end**

1

total++

if b2

vocals++ 1 0

Empty Branch

skip

skip

get

**Generalization: several distribution of slaves measuring locally data,**

**+ Preprocessing**

- 1 master asking for update and computing total result based on local results

Slaves 2

Slaves 1

buffer

counter

Start signal 2

Start signal 1

Master

|  |  |
| --- | --- |
| **Counter** Message buffer;  **Signal** start signal 1, start signal 2  **Slave**:  Counter count = new counter (0,0);  Loop for ever  Wait for start signal  Count char (count);  Send count of buffer.  End loop | **Master**:  Counter local counter;  System count = new counter (0,0);  Create N slaves  Loop forever  …. Start N slaves  // send start signals  Loop N times do  Receive local count from buffer;  System count add (local count)  End loop  … … print results  End loop |

**Concurrency program: we need bid aromatic(Perti Net ) graphs**

**Start signal**

**Strat start signal 2**

**(receive)**

get

while b1 0 s**end**

1 or

total++ and buffer

repeat

if b2

1 0

Empty Branch

vocals++

get

4.2 coverage measures: node, branch, path coverage;

Logic coverage.

 **Statement:** each statement executed at least once.

 **Branch**: each branch traversed (and every entry point taken) at least once.

 **Condition:** each condition True at least once and False at least once.

 **Branch/Condition**: both branch and condition coverage achieved.

 **Compound Condition**: all combinations of condition values at every branch statement covered (and every entry point taken).

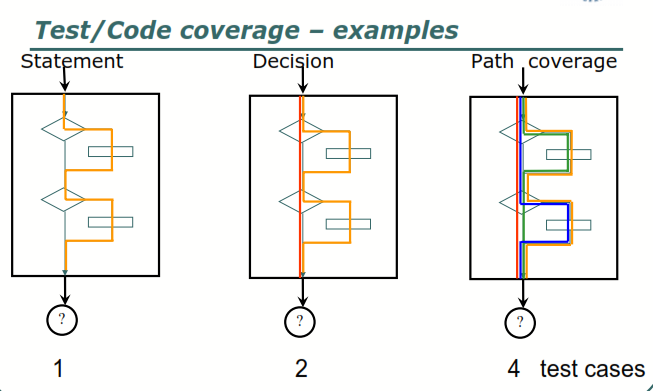
 **Path:** all program paths traversed at least once

**What is the difference to static analysis?**

Testing: Software is executed; graph is used for test case selection;

Analysis: Graph is used to decide certain properties;

|  |  |
| --- | --- |
| Example:  Int i; // not initialize  If true  Then i:=1  Else skip  Endif  Print (i); - - - > Error Message: - false positive “not initialize value i ” | if  i= 1 skip  print (i) |



**(C0) Statement Coverage test:**

|  |  |
| --- | --- |
| * *A set P of execution paths satisfies the statement coverage criterion if and only if for all nodes n in the flow graph, there is at least one path p in P such that p contains the node n.* * Each program sentence is executed at least once by some test case * Satisfies the criterion **→** complete (100%) statement coverage * Does not satisfy the criterion **→** partial (<100 %) statement coverage * Surprisingly difficult to achieve complete coverage in real life * Dead code * Error handling code and other rare conditions * Inadequate test cases * Conditional compiling | Percentage of executable statements exercised by a test suite   * Number of statements exercised/ total number of statements   Example:   * Program has 7 statements * Tests exercise 6 statements * Statement coverage = 85,7%   Typical ad hoc testing achieves 60 – 75% |

Example:

|  |  |
| --- | --- |
| 1 int foo (int a, int b, int c, int d, float e){  2 float e;  3 if (a == 0) {  4 return 0;  5 }  6 int x = 0;  7 if ((a==b) OR ((c == d) AND bug(a) )) {  8 x=1;  9 }  10 e = 1/x;  11 return e;  12 } | *Test Case 2:* the method call **foo (1, 1, 1, 1, 1.)**, expectedreturn value of 1. With this method call, we have achieved 100% statement coverage because we have now executed the program statements on lines 6-12. |

* All following strategies come with a coverage measure, C
  + C = (# of executed test cases) / (# of possible test cases )
  + Quantification of our testing efforts.
* Requires:
  + Measuring executed test cases : instrumentation,
  + Count: possible test cases.

**(C0) Statement Coverage test: (Class Lecture)**

|  |  |
| --- | --- |
| * Execute each statement (at least once) = each node in CSG(control structure graph) * ***Example***: * *Test cases* [0, 0] A (input to get all node only once.) * A, ˽ (space) 🡪 (100% sure for coverage measure) * Empty branches are not executed; * *Example*: empty else branch. * **Pros*:*** * about 18% of all bugs can be identified. * Dead code may be identified. Still possible with ‘structured **goto “s”** (statement): * **Example** : return, break/continue, exceptions(dead code); * **Cons**: * Empty branches not checked. * Any data dependencies are neglected * **Example** :   **If** b  Then a1, a2, …… a9 // 90% statement  Else a10 //10%  **endif** | import dssz.io.stdin;  class Counter {  int vocals;  int total;  } **// class Counter**  public class CountChar {  public static void countChar (Counter count) {  char c = stdin.getChar("");  **while** ( c >= 'A' && c <= 'Z' && count.total < Integer.MAX\_VALUE) {  count.total++;  **if** ( c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U' ) {  count.vocals++;  } **// if**  c = stdin.getChar("");  } **// while**  } **// method countChar**  } **// class CountChar** |

**Graphical representation:**

|  |  |
| --- | --- |
| C’ ( coverage)  # of test cases |  |

**(C1) Branch Coverage test:**

|  |  |
| --- | --- |
| * A set P of execution paths satisfies the branch coverage criterion if and only if for all edges e in the flow graph, there is at least one path p in P such that p contains the edge e. * Each control flow branch is executed at least once by some test case * Satisfies the criterion **→** complete (100 %) branch coverage * Perfect branch coverage **→** perfect statement coverage * Usually achieving branch coverage takes more test cases than achieving statement coverage * Branch coverage is ”stronger” than statement coverage | * Percentage of decision outcomes exercised by a test suite      * Number of decisions outcomes exercised/total number of decision outcomes * Example: * Program has 6 branches * Tests exercises 3 branches * Branch coverage = 50%   + Statement coverage = 85,7%   Typical ad hoc testing achieves 40 – 60% |

**(C1) Branch Coverage test (Class Lecture):**

|  |  |
| --- | --- |
| * Execute all branches = all arcs in CSG = all transition in PN, at least once; * Each conditions : T/F, at least once; * C0 ⊆ C1 ; coincide for programs with end empty branches; * Generally accepted as minimal criteria. (in this sections); * Example: [0,0], A, B, ˽ (space) [ then we can cover all branch] * **Pros**: * Identify non-reachable branches;   Example: x:=1;  If (x>=1)  Then a1 else a2  Endif   * 34% error identification rate; | * **Cons**: * Loops not sufficiently tested; * Neglecting any dependencies between branches; * Example: ?? * Composed conditions are not exhaustively tested. ( 🡪 C3 ) * Code instrumentation: count array entry for each branch;   Example:  If (a<0)  If ((a<0)&++count[2] ≠ 0 || ++count[3]=0)  Or  If (b)  If ((b)&(increment + TRUE)||(increment + FALSE )   * (++count[i] ≠ 0 || ++count[j]=0) 🡪is only executed if first part yields false; |