Definition of White-Box Testing

- Testing based on analysis of internal logic (design, code, etc.). (But expected results still come from requirements.)
- Also know as structural testing.
- White-box testing concerns techniques for designing tests; it is not a “level” of testing.
- White-box testing techniques apply primarily to lower levels of testing (e.g., unit and component).
White-Box Testing Topics

- **Logic coverage (lecture I)**
- Dataflow coverage (lecture II)
- Path conditions and symbolic execution (lecture III)
- Other white-box testing strategies (e.g., “fault-based testing”) (lecture IV)
Types of 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

(cont’d)
Types of Logic Coverage (cont’d)

- **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
input(Y)
if (Y<=0) then
    Y := −Y
end_if
while (Y>0) do
    input(X)
    Y := Y - 1
end_while
Statement Coverage

- Statement Coverage requires that each statement will have been executed at least once.
- Simplest form of logic coverage.
- Also known as *Node Coverage*.
- What is the minimum number of test cases required to achieve statement coverage for the program segment given below?
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
    Y := −Y
end_if
while (Y>0) do
    input(X)
    Y := Y-1
end_while
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
    Y := −Y
end_if
while (Y>0) do
    input(X)
    Y := Y-1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
  Y := −Y
end_if
while (Y>0) do
  input(X)
  Y := Y-1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
    Y := −Y
end_if
while (Y>0) do
    input(X)
    Y := Y-1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
  Y := -Y
end_if
while (Y>0) do
  input(X)
  Y := Y-1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
  Y := −Y
end_if
while (Y>0) do
  input(X)
  Y := Y - 1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y <= 0) then
    Y := -Y
end_if
while (Y > 0) do
    input(X)
    Y := Y - 1
end_while

Claim: only 1 test case is required.
How many test cases required for Statement Coverage?

input(Y)
if (Y<=0) then
    Y := −Y
end_if
while (Y>0) do
    input(X)
    Y := Y - 1
end_while

Claim: only 1 test case is required.
Branch Coverage

- Branch Coverage requires that each branch will have been traversed, and that every program entry point will have been taken, at least once.
- Also known as *Edge Coverage*.

(cont’d)
Branch Coverage (cont’d)

- Why “...and that every program entry point will have been taken, at least once”? 
Branch Coverage (cont’d)

• Why “...and that every program entry point will have been taken, at least once”? 
Branch Coverage (cont’d)

• Why “...and that every program entry point will have been taken, at least once”?
Branch Coverage (cont’d)

- What is the relationship between Statement and Branch Coverage?
What is the relationship between Statement and Branch Coverage?

Possible relationships:

1. None.
2. Statement Coverage subsumes Branch Coverage ("statement => branch").
3. Branch Coverage subsumes Statement Coverage ("branch => statement").
4. Both (2) and (3) (i.e., they are equivalent)
Does “statement => branch” ???

Min. number of cases required for Statement Coverage?

Min. number of cases required for Branch Coverage?
Does “statement => branch” ???

Min. number of cases required for Statement Coverage?  1

Min. number of cases required for Branch Coverage?  2
Does “statement $\Rightarrow$ branch” ???

Min. number of cases required for Statement Coverage? \(1\)

Min. number of cases required for Branch Coverage? \(2\)

Therefore, Statement Coverage does \textbf{not} subsume Branch Coverage.
Does “branch => statement” ???

- Normally, YES...
Does “branch => statement” ???

- Normally, **YES** ... in the absence of “DEAD CODE”.

DEAD CODE is not reachable via any executable program path.

(cont’d)
Does “branch => statement” ?

- If a program has "dead (i.e., unreachable) code", then "statement coverage" is unachievable. (We would need to modify the program in order to bring the dead code back to “life”.)

- **Bottom line:** we will always assume the nominal case of “no dead code” unless explicitly stated otherwise. Under this assumption, Branch Coverage does indeed subsume Statement Coverage.
Condition Coverage

- A branch predicate may have more than one condition.

```plaintext
input(X,Y)
if (Y<=0) or (X=0) then
    Y := -Y
end_if
while (Y>0) and (not EOF) do
    input(X)
    Y := Y-1
end_while

(cont’d)
```
Condition Coverage (cont’d)

- Condition Coverage requires that each condition will have been True at least once and False at least once.

- What is the relationship between Branch and Condition Coverage?
**Condition Coverage (cont’d)**

if \( A \) or \( B \) then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test 1</strong></td>
<td>T</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td><strong>test 2</strong></td>
<td>F</td>
<td>F</td>
<td>?</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 2</td>
<td>F</td>
<td>F</td>
<td>?</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 2</td>
<td>F</td>
<td>F</td>
<td>false</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 2</td>
<td>F</td>
<td>F</td>
<td>false</td>
</tr>
</tbody>
</table>

✔️  ✔️  ✗  ✔️
Condition Coverage (cont’d)

if A or B then
    s1
else
    s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 1</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 2</td>
<td>F</td>
<td>F</td>
<td>false</td>
</tr>
</tbody>
</table>

∴ Branch Coverage $\neq$ Condition Coverage
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test 3</strong></td>
<td>T</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td><strong>test 4</strong></td>
<td>F</td>
<td>T</td>
<td>?</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 3</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 4</td>
<td>F</td>
<td>T</td>
<td>?</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 3</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 4</td>
<td>F</td>
<td>T</td>
<td>true</td>
</tr>
</tbody>
</table>
if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 3</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 4</td>
<td>F</td>
<td>T</td>
<td>true</td>
</tr>
</tbody>
</table>
Condition Coverage (cont’d)

if A or B then
  s1
else
  s2
end_if_then_else

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>test 3</td>
<td>T</td>
<td>F</td>
<td>true</td>
</tr>
<tr>
<td>test 4</td>
<td>F</td>
<td>T</td>
<td>true</td>
</tr>
</tbody>
</table>

\[ \therefore \text{ Condition Coverage } \not\Rightarrow \text{ Branch Coverage} \]
Branch/Condition Coverage

- Branch/Condition Coverage requires that both Branch AND Condition Coverage will have been achieved.

- Therefore, Branch/Condition Coverage subsumes both Branch Coverage and Condition Coverage.
What if the compiler generates code that masks the evaluation of conditions?

That is, suppose

if (A) or (y/x=5) then...

is compiled in such a way that if A is true, y/x=5 will not be evaluated.
Compound Condition Coverage (cont’d)

- Compound Condition Coverage requires that all combinations of condition values at every branch statement will have been covered, and that every entry point will have been taken, at least once.

- Also know as *Multiple Condition Coverage*.

- Subsumes Branch/Condition Coverage, regardless of the order in which conditions are evaluated.

(cont’d)
Compound Condition Coverage (cont’d)

Combinations of condition values:  TT, TF, FT, FF

input(X,Y)
if \((Y \leq 0)\) or \((X = 0)\) then
    \(Y := -Y\)
end_if

(cont’d)
Compound Condition Coverage (cont’d)

• In general, how many different combinations of condition values must be considered when a branch predicate has $N$ conditions?
Compound Condition Coverage (cont’d)

• In general, how many different combinations of condition values must be considered when a branch predicate has $N$ conditions?

$$2^N$$
Path Coverage

- Path Coverage requires that all program paths will have been traversed at least once.
- Often described as the “strongest” form of logic coverage. (Is it stronger than Compound Condition Coverage?)
- Path Coverage is usually impossible when loops are present. (How many test cases would be required to cover all paths in the example below?)

(cont’d)
for I = 1 to 30 do
    input(X,Y)
    if (Y<=0) then
        if (X<=0) then
            Y := -X
        else
            Y := -Y
        end_if_else
    else
        Y := X+Y
    end_if_else
end_for_do

repeat 29 times
Path Coverage (cont’d)

3 paths

3 paths

3 X 3 = 9 paths
Path Coverage (cont’d)

repeat 29 times

$3 \times 3 \times \ldots \times 3$

$= 3^{30}$ paths

(cont’d)
Path Coverage (cont’d)

• Various strategies have been developed for identifying useful subsets of paths for testing when Path Coverage is impractical:
  
  – *Loop Coverage,*
  
  – *Basis Paths Coverage,* and
  
  – *Dataflow Coverage* (Lecture 8).
Loop Coverage

- Loop Coverage requires that the body of loops be executed 0, 1, 2, t, max, and max+1 times, where possible.
Loop Coverage

- Loop Coverage requires that the body of loops be executed 0, 1, 2, t, max, and max+1 times, where possible.

- Rationale:
  - **0**: Is some action taken in the body that must also be taken when the body is not executed?
  - **1**: Check lower bound on number of times body may be executed.

(cont’d)
Loop Coverage (cont’d)

- **Rationale: (cont’d)**
  - **2**: Check loop re-initialization.
  - **t**: Check *typical* number of iterations.
  - **max**: Check upper (valid) bound on number of times body may be executed.
  - **max+1**: If the maximum can be exceeded, what behavior results?
Basis Paths Coverage

• A coverage criterion associated with 
  McCabe’s Structured Testing.

• Based on idea of identifying a “spanning” 
  (i.e., basis) set of paths for a program’s “path space.”

• The number, $C$, of such paths is equal to the number of (2-way) branch statements in the program + 1. (This is also the number of enclosed regions in the program graph + 1.)
Spanning vectors in 3n-space

\[ \vec{A} = (x, y, z) = x (1, 0, 0) + y (0, 1, 0) + z (0, 0, 1) \]
Basis Paths Coverage

- A coverage criterion associated with *McCabe’s Structured Testing*.
- Based on idea of identifying a “spanning” (i.e., *basis*) set of paths for a program’s “path space.”
- The number, $C$, of such paths is equal to the number of (2-way) branch statements in the program + 1. (This is also the number of enclosed regions in the program graph + 1.)
Basis Paths Coverage (cont’d)

- C is what McCabe calls the *Cyclomatic Complexity* of a program.

- Any C distinct, *simple* program paths that provide branch coverage also form a *basis* set of paths. (In a *simple* program path, while loop bodies are executed at most once and repeat-until loop bodies are executed at most twice.)
Example 1

if a then s1
else if b then s2
    else if c then s3
    else s4
    end_if_then_else
end_if_then_else
end_if_then_else
end_if_then_else
end_if_then_else
end_if_then_else
end_if_then_else
end_if_then_else

Paths: ____ Basis Paths: ____ Cases for branch coverage: ____
Example 1

if a then s1
else if b then s2
  else if c then s3
  else s4
  end_if_then_else
end_if_then_else
end_if_then_else

Paths: 4 Basis Paths: ____ Cases for branch coverage: ___
Example 1

if a then s1
else if b then s2
  else if c then s3
  else s4
  end_if_then_else
end_if_then_else
end_if_then_else

Paths: 4  Basis Paths: ___  Cases for branch coverage: ___
Example 1

if a then s1
else if b then s2
  else if c then s3
  else s4
  end_if_then_else
end_if_then_else
end_if_then_else

Paths: 4 Basis Paths: 4 Cases for branch coverage: ___
Example 1

if a then s1
else if b then s2
  else if c then s3
  else s4
  end_if_then_else
end_if_then_else
end_if_then_else

Paths: ___ Basis Paths: ___ Cases for branch coverage: ___
Example 2

if a then
    s1
end_if_then
if b then
    s2
end_if_then
if c then
    s3
end_if_then

Paths: ___ Basis Paths: ___ Cases for branch coverage: ___
Example 2

if a then
  s1
end_if_then
if b then
  s2
end_if_then
if c then
  s3
end_if_then

Paths: ___ Basis Paths: ___ Cases for branch coverage: ___
Example 2

if a then
  s1
end_if_then
if b then
  s2
end_if_then
if c then
  s3
end_if_then

Paths: 8 Basis Paths: ___ Cases for branch coverage: ___
Example 2

if a then
  s1
end_if_then
if b then
  s2
end_if_then
if c then
  s3
end_if_then

Paths: 8  Basis Paths: 4  Cases for branch coverage: ___
Example 2

if a then
  s1
end_if_then
if b then
  s2
end_if_then
if c then
  s3
end_if_then

Paths: 8  Basis Paths: 4  Cases for branch coverage: 2
Example 3

while a do
    if b then s1
    else s2
    end_if_then_else
end_while

Paths: ____ Basis Paths: ____ Cases for branch coverage: ____
Example 3

while a do
  if b then s1
  else s2
end_if_then_else
end_while

Paths: \(\infty\) Basis Paths: ___ Cases for branch coverage: ___
Example 3

while a do
  if b then s1
  else s2
end_if_then_else
end_while

Paths: $\infty$  Basis Paths: ___  Cases for branch coverage: ___
Example 3

while a do
  if b then s1
  else s2
  end_if_then_else
end_while

Paths: \(\infty\) Basis Paths: 3 Cases for branch coverage: ___
Example 3

while a do
  if b then s1
  else s2
end_if_then_else
end_while

Paths: ___ Basis Paths: ___ Cases for branch coverage: ___
In General...

Number of program Paths \(\geq\) Number of Basis Paths \(\geq\) Number of test cases required for branch coverage

Path Coverage \(\Rightarrow\) Basis Paths Coverage \(\Rightarrow\) Branch Coverage
Exercise

Prove that **Path** and **Compound Condition Coverage** are **independent**.

(Hint: consider the proof that Branch and Condition Coverage are independent.)
In the next lecture we consider a family of path selection criteria based on the idea that program paths along which variables are defined and then used should be covered.

The strategy is popularly known as *Dataflow Coverage*.