Lecture 04: Basic White-Box Testing: Instruction & Control-Flow Coverage

Spring 2022
Lectures

- Lecture 1 (10.02) – Introduction to Software Testing
- Lecture 2 (17.02) – Basic Black-Box Testing Techniques: Boundary Value Analysis & Equivalence Class Partitioning
- Lecture 3 (03.03) – BBT advanced: Combinatorial Testing
- Lecture 4 (10.03) – Basic White-Box Testing Techniques: Control-Flow Coverage
- Lecture 5 (17.03) – BBT adv.: State-Transition, Metamorphic, Random Testing
- Lecture 6 (24.03) – Test Levels, Test Tools, Test Automation
- Lecture 7 (31.03) – BBT adv.: Exploratory Testing, Behaviour Testing
- Lecture 9 (14.04) – Security Testing of Mobile Applications
- Lecture 10 (21.04) – WBT adv.: Data-Flow Testing / Mutation Testing
- Lecture 12 (05.05) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
- Lecture 13 (12.05) – Exam Preparation
- Lecture 14 (19.05) – Advanced Topics (optional)
Structure of Lecture 4

- Code Coverage Introduction
- Control-Flow Criteria
  - Control Flow Graph
  - Node Coverage
  - Branch Coverage
  - Condition Coverage
  - Independent Path Coverage
  - Loop Coverage
  - Summary
- Lab 4
White Box Testing (WBT)

- WBT, also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing:
  - A test method where the internal structure/design/implemention of the item being tested is known to the tester.
  - The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential.

Aims to detect these types of issues:
- Control flow issues
- Data flow issues
- Algorithm issues
Testing Strategies

Black Box Testing

White Box Testing

requirements

input

output

events

Black Box Testing

White Box Testing

input

output

events

requirements
Black-Box vs. White-Box

**Specification-based Testing:**
Test against specification

**Structural Testing:**
Test against implementation

**System**

- **Specification**
  - Missing functionality: Cannot be (directly) revealed by white-box techniques

- **Implementation**
  - Unexpected functionality: Cannot be (directly) revealed by black-box techniques
How do Black-Box and White-Box Testing relate to one another?

- Develop an initial Test suite using BB techniques
- Analyze the parts of the code uncovered by BB test suite
  
  - Apply BB coverage criteria to enhance it
  
  - Enhance the Test suite using WB techniques
  
  - Apply WB coverage criteria to enhance it
Example: Statement (Node) Coverage

Assume a ‘magic’ Function $M$

$M(x, y) \rightarrow \text{sum} = x + y$ with $x, y: \text{int (32 bit)}$

Possible approaches:
- Execute each statement
- Execute paths based on:
  - Control-flow (decisions, conditions, loops, independent paths, etc.)
  - Data-flow (definition and usage of variables)
- Read (review) code

White Box

```c
...
if ( x - 100 <= 0 ) {
  if ( y - 100 <= 0 ) {
    if ( x + y - 200 == 0 ) {
      crash();
    }
  }
}
print(x + y);
```
Example: Statement (Node) Coverage

Assume a ‘magic’ Function M

\[ M(x, y) \rightarrow \text{sum} = x + y \]

with \(x, y: \text{int}\) (32 bit)

Possible approaches:
- Execute each statement
- Execute paths based on:
  - Control-flow (decisions, conditions, loops, independent paths, etc.)
  - Data-flow (definition and usage of variables)
- Read (review) code

White Box

```c
...
if ( x - 100 <= 0 ) {
    if ( y - 100 <= 0 ) {
        if ( x + y - 200 == 0 ) {
            crash();
        }
    }
}
print(x + y);
```

How many statements covered with BBT suite?
Example: Statement (Node) Coverage

Assume a ’magic’ Function M

\[ M(x, y) \rightarrow \text{sum} = x + y \]
with \( x, y: \text{int} \) (32 bit)

TC1: 0, 0 -> 0

\[
\begin{align*}
\text{int actual} &= \text{sum}(0,0); \\
\text{int expected} &= 0; \\
\text{assertEquals(actual, expected);} \\
\text{print}(x + y);
\end{align*}
\]

White Box

\[
\begin{align*}
\ldots
\text{if ( x - 100 }\leq 0 \text{ ) { & } } \\
\text{if ( y - 100 }\leq 0 \text{ ) { & } } \\
\text{if ( x + y - 200 }\leq 0 \text{ ) { & } } \\
\text{crash(); } & \\
\text{print}(x + y);
\end{align*}
\]

How many statements covered with BBT suite?
Example: Statement (Node) Coverage

Assume a ‘magic’ Function M

M(x, y) \rightarrow \text{sum} = x + y
with x, y: \text{int} (32 \text{ bit})

TC1: 0, 0 -> 0

White Box

... 
if (x - 100 <= 0) {
    if (y - 100 <= 0) {
        if (x + y - 200 == 0) {
            crash(); }}}
print(x + y);

How many statements covered with BBT suite?

\rightarrow 80\%
Example: Statement (Node) Coverage

Assume a 'magic' Function $M$

$M(x, y) \rightarrow \text{sum} = x + y$
with $x, y$: int (32 bit)

White Box

```python
...
if ( x - 100 <= 0 ) {
    if ( y - 100 <= 0 ) {
        if ( x + y - 200 == 0 ) {
            crash();
        }
    }
}
print(x + y);
```

TC1: 0, 0 -> 0
TC2: notInt, 0 -> WrongInputException
TC3: 0, notInt -> WrongInputException
TC4: MinInt, MinInt -> ArithmeticException
TC5: MaxInt, MaxInt -> ArithmeticException
TC6: MaxInt/2, MaxInt/2 -> MaxInt
TC7: MinInt/2, MinInt/2 -> MinInt

How many statements covered with BBT suite?
Example: Statement (Node) Coverage

Assume a 'magic' Function $M$

$M(x, y) \Rightarrow \text{sum} = x + y$
with $x, y: \text{int (32 bit)}$

White Box

\[
\begin{align*}
\text{...} & \\
\text{if ( } x - 100 \leq 0 \text{ ) { } } \\
& \quad \text{if ( } y - 100 \leq 0 \text{ ) { } } \\
& \quad \quad \text{if ( } x + y - 200 == 0 \text{ ) { } } \\
& \quad \quad \quad \text{crash(); } \\
}\text{print}(x + y); \end{align*}
\]

TC1: 0, 0 -> 0
TC2: notInt, 0 -> WrongInputException
TC3: 0, notInt -> WrongInputException
TC4: MinInt, MinInt -> ArithmeticException
TC5: MaxInt, MaxInt -> ArithmeticException
TC6: MaxInt/2, MaxInt/2 -> MaxInt
TC7: MinInt/2, MinInt/2 -> MinInt

How many statements covered with BBT suite?

\[\rightarrow 80\%\]
Example: Statement (Node) Coverage

- If we try to cover all statements, we must find input data such that all three if-statements are ’true’:

Traverse code and combine conditions:
(x<=100) and (y<=100) and (x+y=200) ->
(200-y<=100) and (y<=100) ->
(y>=100) and (y<=100) ->
y = 100 ->
x = 100

White Box

... if ( x - 100 <= 0 ) {
   if ( y - 100 <= 0 ) {
      if ( x + y - 200 == 0 ) {
         crash(); }}}

print(x + y);
Example: Statement (Node) Coverage

Assume a 'magic' Function M

\[ M(x, y) \rightarrow \text{sum} = x + y \]
with \( x, y: \text{int (32 bit)} \)

TC1: \( M(0, 0) \rightarrow 0 \) Pass
TC2: \( M(100, 100) \rightarrow 200 \) Fail: crash()

1st if = true: \( x \leq 100 \)
2nd if = true: \( y \leq 100 \)
3rd if = true: \( x + y = 200 \)

\[ \rightarrow 100\% \text{ Statement Coverage} \]

White Box

\[
\ldots
\]

\[
\text{if} \ ( x - 100 \leq 0 ) \{
\text{if} \ ( y - 100 \leq 0 ) \{
\text{if} \ ( x + y - 200 = 0 ) \{
\text{crash}(); \}}
\}
\]

\( \text{print}(x + y); \)
Example: Control-Flow Graph

Assume a ‘magic’ Function $M$

$M(x, y) \rightarrow \text{sum} = x + y$
with $x, y$: int (32 bit)

Control-Flow Graph (CFG)?

White Box

... 
if ( $x - 100 \leq 0$ ) {
    if ( $y - 100 \leq 0$ ) {
        if ( $x + y - 200 = 0$ ) {
            crash(); } 
    } 
}
print($x + y$);
Example: Control-Flow Graph

Control Flow Graph (CFG)

White Box

... 
if ( x - 100 <= 0 ) { 
  if ( y - 100 <= 0 ) { 
    if ( x + y - 200 == 0 ) { 
      crash(); 
    } 
  } 
} 
print(x + y);
Example: Branch (Decision) Coverage

- If we try to cover all branches, we must find input data such that all three if-statements are once evaluated to 'true' and once to 'false':

TC1: $M(0, 0) \rightarrow 0$
TC2: $M(100, 100) \rightarrow 200$

How much branch coverage?
Example: Branch (Decision) Coverage

Assume a ‘magic’ Function M

TC1: M(0, 0) -> 0
TC2: M(100, 100) -> 200

-> 66% Branch (or Decision) Coverage
Example: Branch (Decision) Coverage

Assume a 'magic' Function M

TC1: M(0, 0) \(\rightarrow\) 0
TC2: M(100, 100) \(\rightarrow\) 200

\(\rightarrow\) 66\% Branch (or Decision) Coverage

How calculated?
Example: Branch (Decision) Coverage

Assume a ‘magic’ Function M

TC1: M(0, 0) -> 0
TC2: M(100, 100) -> 200

-> 66% Branch (or Decision) Coverage

TC1: if1 = true / if2 = true / if3 = false
TC2: if1 = true / if2 = true / if3 = true

Missing:
if1=false and if2=false
-> 2 additional TCs needed
White-Box Testing Techniques

- Control-Flow Testing
- Data-Flow Testing
- Mutation Testing
- Symbolic Execution
- Static Code Analysis
- Reviews

Lecture 9

Lecture 10
Testing Strategies

Black Box Testing

White Box Testing
White-Box Testing: Path Explosion!

There are many possible paths!

How many?
White-Box Testing: Path Explosion!

There are many possible paths!
\[5^{20} \approx 10^{14}\] different paths

Selectiv Testing
Solution: Selective Testing

Goal:

Cover as many elements of the CFG (and conditions that can be attached to it) with as little effort as possible.
Code Coverage

Definition:

• Measures the extent to which certain code items related to a defined test adequacy criterion have been executed (covered) by running a set of test cases (= test suite)

Goal:

• Define test suites such that they cover as many (disjoint) code items as possible
Main Classes of Code Coverage Criteria
(= Test Adequacy Criteria)

Control Flow Criteria:

• Statement (node), decision (branch), condition, and path coverage are examples of control flow criteria
• They rely on syntactic characteristics of the program (ignoring the semantics of the program computation)

Data Flow Criteria:

• Require the execution of path segments that connect parts of the code that are intimately connected by the flow of data (-> ‘annotated control flow graph’)

---

LTAT.05.006 / Lecture 04 / © Dietmar Pfahl 2022
Code Coverage Measure – Example

Statement Coverage ($CV_s$)

- Portion of the statements tested by at least one test case.

\[
CV_s = \left( \frac{S_t}{S_p} \right) \times 100\%
\]

$S_t$ : number of statements tested

$S_p$ : total number of statements
Code Coverage Measure – Tools

For Java:

IntelliJ code coverage
Emma
JaCoCo
Clover
etc.

http://www.eclemma.org/index.html

Note: EclEmma requires Eclipse
Code Coverage Measure – EclEmma

Branch coverage

Line coverage

http://www.eclemma.org/index.html
Code Coverage Measure – EclEmma

Branch coverage

Line coverage

Source lines containing executable code get the following color code:
- green for fully covered lines
- yellow for partly covered lines (some instructions or branches missed)
- red for lines that have not been executed at all
- white lines are not considered instructions
Code Coverage Measure – IntelliJ Code Coverage Tool (JaCoCo)

View coverage results:

In the Project tool window:

In the dedicated Coverage tool window:
Code Coverage Measure – IntelliJ Code Coverage Tool (JaCoCo)

Use the color indicators in the left gutter to detect the uncovered lines of code.

To find out how many times a line has been hit, click the line in the gutter area.

The pop-up window that opens shows the statistic for the line at caret. For lines with conditions, the pop-up window also provides statistic.
Code Coverage Measure – IntelliJ Code Coverage Tool

Use the color indicators in the left gutter to detect the uncovered lines of code.

To find out how many times a line has been hit, click the line in the gutter area.

The pop-up window that opens shows the statistic for the line at caret. For lines with conditions, the pop-up window also provides statistic.
**Code Coverage Measure – IntelliJ Code Coverage Tool**

Use the color indicators in the left gutter to detect the uncovered lines of code.

To find out how many times a line has been hit, click the line in the gutter area.

The pop-up window that opens shows the statistic for the line at caret. For lines with conditions, the pop-up window also provides a statistic.
Structure of Lecture 4

• Code Coverage Introduction
• Control-Flow Criteria
  • Control Flow Graph
  • Node Coverage
  • Branch Coverage
  • Condition Coverage
  • Independent Path Coverage
  • Loop Coverage
  • Summary
• Lab 4
Control Flow Graph (CFG)

Program

```plaintext
x = z-2;
y = 2*z;
if (c) {
    x = x+1;
y = y+1;
}
else {
    x = x-1;
y = y-1;
}
z = x+y;
```

Control Flow Graph

```
B_1  x = z-2;
y = 2*z;
     c=T

B_2  x = x+1;
y = y+1;

B_3  x = x-1;
y = y-1;
     c=F

B_4  z = x+y;
```
Control Flow Graph (CFG)

```
Program
x = z-2;
y = 2*z;
if (c) {
    x = x+1;
y = y+1;
}
else {
    x = x-1;
y = y-1;
}
```

```
Control Flow Graph

B1: x = z-2;
y = 2*z;
c = T

B2: x = x+1;
y = y+1;

B3: x = x-1;
y = y-1;
c = F

B4: empty

Blocks (=Nodes): 4
Edges: 4
```
Control Flow Graph (CFG)

Program

\[
\begin{align*}
x &= z - 2; \\
y &= 2 \times z; \\
\text{if } (c) \{ \\
&\quad x = x + 1; \\
&\quad y = y + 1; \\
\} \\
\text{else } \{ \\
&\quad x = x - 1; \\
&\quad y = y - 1; \\
\} \\
z &= x + y;
\end{align*}
\]
Control Flow Graph (CFG)

d: decision node
s: statement node

Nodes: 9
Edges: 9

Blocks: 4
Edges: 4

entry and exit nodes are ‘dummy nodes’

Control Flow Graph

B1
x = z-2;
y = 2*z;

c=T

B2
x = x+1;
y = y+1;

c=F

B3
x = x-1;
y = y-1;

B4
End

START

D: decision node
S: statement node

Nodes: 9
Edges: 9

Blocks: 4
Edges: 4

entry and exit nodes are ‘dummy nodes’
Control Flow Graph – Example

How to construct the CFG?

If (d1) then {
    if (d2) then {s1}
    s2
    while (d3) do {s3}
}
else {
    if (d4) then {
        repeat {s4} until (d5)
    }
}

Control Flow Graph – Example

If \( (d_1) \) then {
  
  if \( (d_2) \) then {\( s_1 \)}
  
  \( s_2 \)
  
  while \( (d_3) \) do {\( s_3 \)}
  
} 

else {
  
  if \( (d_4) \) then {
    
    repeat {\( s_4 \)} until \( (d_5) \)
    
  } 

}
Control Flow Graph – Example

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}
else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}
Control Flow Graph – Example

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}
else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}
Control Flow Graph – Example

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}
else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}
else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}
Control Flow Graph – Example

Often the entry node is not shown because it is redundant

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}

else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
}

else {
  if (d4) then {
    repeat {s4} until (d5)
  }
}
Control-Flow Graph for Exception Handling

```java
void tryCatchTestMethod(int b, int c, int t) {
    try {
        mightThrowAnException(b);
    } catch (Exception e) {
        b = 3;
    } finally {
        t = b*3;
        c = 3;
    }
}
```

How to construct the CFG?
Control-Flow Graph for Exception Handling

```java
void tryCatchTestMethod(int b, int c, int t) {
    try {
        mightThrowAnException(b);
    }
    catch (Exception e) {
        b = 3;
    }
    finally {
        t = b*3;
    }
    c = 3;
}
```
Control-Flow Graph for Exception Handling

```java
void tryCatchTestMethod(int b, int c, int t) {
    try {
        mightThrowAnException(b);
    } catch (Exception e) {
        b = 3;
    } finally {
        t = b*3;
    }
    c = 3;
}
```
Overview of Control Flow Criteria

- Statement (or Block or Node) Coverage – all nodes
- Decision (or Branch) Coverage – all edges
- Condition Coverage
- Condition/Decision Coverage
- Multiple Condition Coverage
- Modified Condition Decision Coverage (MC/DC)
- Linearly Independent Paths
- Loop Testing

...
Statement (Node) Coverage

Execute each statement at least once

• Use tools to monitor execution
• More practice in Lab 4

A possible concern may be:

• Dead code
Exercise
with
Life Insurance Program
**Life Insurance Example**

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:    return (TRUE);
3: if (gender == male && age < 80)
4:    return (TRUE);
5: return (FALSE);
```

In the following assume that the following pre-conditions have been checked:
- Parameter 'gender' is in {female, male}
- Parameter 'age' is integer and >= 18
Life Insurance Example

```java
boolean AccClient(int age; gtype gender)
1: if (gender == female && age < 85) 
2: return (TRUE);
3: if (gender == male && age < 80) 
4: return (TRUE);
5: return (FALSE);
```

In the following assume that the following pre-conditions have been checked:
- Parameter 'gender' is in {female, male}
- Parameter 'age' is integer and >= 18
Life Insurance Example

\[
\text{boolean AccClient(int age; gtype gender)} \\
\]

1: \textit{if} (gender == female && age < 85) \quad d_1 = c_1 \& c_2 \\
2: \quad \text{return(} \text{TRUE)\}; \\
3: \textit{if} (gender == male && age < 80) \\
4: \quad \text{return(} \text{TRUE)\}; \\
5: \quad \text{return(} \text{FALSE)\}; \\

In the following assume that the following pre-conditions have been checked:
- Parameter 'gender' is in \{female, male\}
- Parameter 'age' is integer and \(\geq 18\)
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85) d₁ = c₁ & c₂
2:   return (TRUE);
3: if (gender == male && age < 80)
4:   return (TRUE);
5: return (FALSE);

Test:

0 %
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85) return(TRUE);
2: if (gender == male && age < 80) return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true
Statement Coverage /3

boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85) d1 = c1 & c2
2: return(TRUE);
3: if (gender == male && age < 80) d2 = c3 & c4
4: return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false

80 %
Statement Coverage /4

```cpp
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85) return (TRUE);
2: return (TRUE);
3: if (gender == male && age < 80) return (TRUE);
4: return (FALSE);
```

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

100 %
Same Test Suite but Incorrect Code in Life Insurance Program: Example 1

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 80) d1 = c1 & c2
2: return (TRUE);
3: if (gender == male && age < 80) d2 = c3 & c4
4: return (TRUE);
5: return (FALSE);
```

Test:
- `AccClient(83, female)` -> false
- `AccClient(83, male)` -> false
- `AccClient(25, male)` -> true

80% failure rate

Where is the bug?
Same Test Suite but Incorrect Code in Life Insurance Program: Example 1

```java
boolean AccClient(int age; gtype gender)
1: if (gender == female && age < 80) return (TRUE);
2:   return (TRUE);
3: if (gender == male && age < 80) return (TRUE);
4:   return (FALSE);
```

Test:
- AccClient(83, female) -> false
- AccClient(83, male) -> false
- AccClient(25, male) -> true

80%
Same Test Suite but Incorrect Code in Life Insurance Program: Example 2

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female && age > 85)
2:   return (TRUE);
3: if (gender == male && age < 80)
4:   return (TRUE);
5: return (FALSE);
```

Test:
- AccClient(83, female)->false
- AccClient(83, male)->false
- AccClient(25, male)->true

1 fault triggers 1 failure

80 %
Same Test Suite but Incorrect Code in Life Insurance Program: Example 3

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female && age > 80)
2: return(TRUE);
3: if (gender == male && age < 80)
4: return(TRUE);
5: return(FALSE);
```

2 faults
trigger
0 failures

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

100 %

NB: For the given test suite!
### Statement Coverage: Dead Code?

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female){
2:   if (age < 85)
3:     return(TRUE);
4:   return(FALSE);
5: if (gender == male){
6:   if (age < 80)
7:     return(TRUE);
8:   return(FALSE);
9: return(FALSE);
```

Test:
- AccClient(83, female)->true
- AccClient(83, male)->false
- AccClient(25, male)->true

78 %
boolean AccClient(int age; gtype gender)

1: if (gender == female){
2:   if (age < 85)
3:     return(TRUE);
4:   return(FALSE);} 
5: if (gender == male){
6:   if (age < 80)
7:     return(TRUE);
8:   return(FALSE);} 
9: return(FALSE); 

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

78 %
Statement Coverage: Dead Code?

```java
boolean AccClient(int age; gtype gender)

1: if (gender == female){
2:   if (age < 85)
3:     return(TRUE);
4:   return(FALSE);} 
5: if (gender == male){
6:   if (age < 80)
7:     return(TRUE);
8:   return(FALSE);} 
9: return(FALSE);
```

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

78 %
boolean AccClient(int age; gtype gender)

1: if (gender == female){
2:   if (age < 85)
3:     return(TRUE);
4:     return(FALSE);
5: if (gender == male){
6:   if (age < 80)
7:     return(TRUE);
8:     return(FALSE);
9: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

100 %
Decision (Branch) Coverage /1

boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:     return (TRUE);
3: if (gender == male && age < 80)
4:     return (TRUE);
5: return (FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

Branch coverage?
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:   return(TRUE);
3: if (gender == male && age < 80)
4:   return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true

25 %
Decision (Branch) Coverage /3

boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85) return(TRUE);
2: return(TRUE);
3: if (gender == male && age < 80) return(TRUE);
4: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false

75 %
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:     return(TRUE);
3: if (gender == male && age < 80)
4:     return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

100 %
Condition Coverage

• Test all conditions (in all predicate nodes):
  • Minimum: Each condition must evaluate at least once
  • Simple: Each condition must evaluate at least once to 'true' and once to 'false'

• Example of a decision (predicate) with two conditions:
  
  If (A==female & B<85) then ...

• A predicate may contain several conditions connected via Boolean operators
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)  
   d₁ = c₁ & c₂
2:   return (TRUE);
3: if (gender == male && age < 80)   
   d₂ = c₃ & c₄
4:   return (TRUE);
5: return (FALSE);

Test:

0 %
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:    return (TRUE);
3: if (gender == male && age < 80)
4:    return (TRUE);
5: return (FALSE);

Test: AccClient(83, female)->true

25 % (or 50 %)
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:    return(TRUE);
3: if (gender == male && age < 80)
4:    return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false

62.5 % (or 100 %)
boolean AccClient(int age; gtype gender)

1: if (gender == female && age < 85)
2:    return(TRUE);
3: if (gender == male && age < 80)
4:    return(TRUE);
5: return(FALSE);

Test:
AccClient(83, female)->true
AccClient(83, male)->false
AccClient(25, male)->true

75 % (or 100 %)
Advanced Condition Coverage

Condition/Decision Coverage (C/DC)
  • as DC plus: every condition in each decision is tested in each possible outcome

Modified Condition/Decision coverage (MC/DC)
  • as above plus, every condition shown to independently affect a decision outcome (by varying that condition only)
    Def: A condition independently affects a decision when, by flipping that condition’s outcome and holding all the others fixed, the decision outcome changes
  • this criterion was created at Boeing and is required for aviation software according to RCTA/DO-178B

Multiple-Condition Coverage (M-CC)
  • all possible combinations of condition outcomes within each decision is checked
CC, DC, C/DC, M-CC, MC/DC Examples

If \((A==\text{fem} \& B<85)\) ...

Minimum and Simple Condition (CC):
\((TF) A = \text{fem}; B = 200 \ (D: \text{False})\)
\([\text{FT} A = \text{male}; B = 80 \ (D: \text{False})]\)

Decision (DC):
\((TT) A = \text{fem}; B = 80 \ (D: \text{True})\)
\((FT) A = \text{male}; B = 80 \ (D: \text{False})\)

Condition/Decision (C/DC):
\((TT) A = \text{fem}; B = 80 \ (D: \text{True})\)
\((FF) A = \text{male}; B = 200 \ (D: \text{False})\)

Multiple Condition (M-CC):
\((TT) A = \text{fem}; B = 80 \ (D: \text{True})\)
\((FT) A = \text{male}; B = 80 \ (D: \text{False})\)
\((TF) A = \text{fem}; B = 200 \ (D: \text{False})\)

Modified Condition/Decision (MC/DC):
\((TT) A = \text{fem}; B = 80 \ (D: \text{True})\)
\((FT) A = \text{male}; B = 80 \ (D: \text{False})\)
\((TF) A = \text{fem}; B = 200 \ (D: \text{False})\)
Modified Condition/Decision (MC/DC)

If (A=fem & B<85) then ... 

<table>
<thead>
<tr>
<th>TC</th>
<th>A</th>
<th>B</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T (fem)</td>
<td>T  (80)</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F (male)</td>
<td>T  (80)</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T (fem)</td>
<td>F  (200)</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F (male)</td>
<td>F  (200)</td>
<td>F</td>
</tr>
</tbody>
</table>

Multiple Condition:

(TT) A = fem; B = 80 (D: True)
(FT) A = male; B = 80 (D: False)
(TF) A = fem; B = 200 (D: False)
(FF) A = male; B = 200 (D: False)

TC1+TC2: change in A -> Dec changed
TC1+TC3: change in B -> Dec changed

All other TC combinations in which only one condition outcome changes don’t have an effect on the decision outcome.

Result: only TC1, TC2, and TC3 needed
Path Coverage

- **Path Coverage Criterion**: Select a test set $T$ such that, by executing $P$ for each test case $t$ in $T$, all paths leading from the initial to the final node of $P$'s control flow graph are traversed.
- In practice, however, the number of paths is too large, if not infinite (e.g., when we have loops).
- Some paths are infeasible (e.g., not practical given the system’s business logic).
- It may be important to determine “critical paths”, leading to more system load, security intrusions, etc.
Independent Path Coverage

- McCabe cyclomatic complexity estimates number of test cases needed
- The number of independent paths needed to cover all simple paths at least once in a program
  - Visualize by drawing a CFG
  - $CC = \#(edges) - \#(nodes) + 2$
  - $CC = \#(decisions) + 1$
Independent Paths Coverage – Example

- Independent Paths Coverage
  - Requires that a minimum set of linearly independent paths through the control flow-graph be executed
- This test strategy is the rationale for McCabe’s cyclomatic number (McCabe 1976) …
  - … which is equal to the number of test cases required to satisfy the strategy.
Independent Paths Coverage – Example

- Independent Paths Coverage
  - Requires that a minimum set of linearly independent paths through the control flow-graph be executed

- This test strategy is the rationale for McCabe’s cyclomatic number (McCabe 1976) …
  - … which is equal to the number of test cases required to satisfy the strategy.

Cyclomatic Complexity = ?
Independent Paths Coverage – Example

- Independent Paths Coverage
  - Requires that a minimum set of linearly independent paths through the control flow graph be executed

- This test strategy is the rationale for McCabe’s cyclomatic number (McCabe 1976) …
  - … which is equal to the number of test cases required to satisfy the strategy.

Cyclomatic Complexity = 5 + 1 = 6
Independent Paths Coverage – Example

Edges: 1-2-3-4-5-6-7-8-9-10-11-12-13-14
Path1: 1-0-0-1-0-1-0-1-0---0---0---0---0
Path2: 1-0-1-0-1-1-1-1-1-0---0---0---0---0
Path3: 1-0-0-1-0-1-1-1-1-0---0---0---0---0
Path4: 0-1-0-0-0-0-0-0-0-1---0---1---0---1
Path5: 0-1-0-0-0-0-0-0-0-1---1---1---1---1
Path6: 0-1-0-0-0-0-0-0-0-0---1---0---0---0
Independent Paths Coverage – Example

Edges: 1-2-3-4-5-6-7-8-9-10-11-12-13-14

Why no need to cover Path7 ???

Path7: 1-0-1-0-1-1-0-0-1-0---0---0---0---0
Independent Paths Coverage – Example

Edges: 1-2-3-4-5-6-7-8-9-10-11-12-13-14

Why no need to cover Path7 ???

Path7: 1-0-1-0-1-1-0-0-1-0---0---0---0---0

Because it equals Path1+Path2-Path3 !!!

Path1: 1-0-0-1-0-1-0-0-1-0---0---0---0---0
Path2: 1-0-1-0-1-1-1-1-1-0---0---0---0---0
P1+P2: 2-0-1-1-1-2-1-1-2-0---0---0---0---0
Path3: 1-0-0-1-0-1-1-1-1-0---0---0---0---0
-P3: 1-0-1-0-1-1-0-0-1-0---0---0---0---0
Loop Testing

- Simple loop
- Nested loops
- Concatenated loops
- Unstructured loops
Loop Testing: Simple Loops

Minimum conditions - 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 m < n
5. set loop counter to (n-1), n and (n+1): passes twice through the loop and once not

… where n is the maximum number of allowable passes
Nested Loops

Extend simple loop testing

Reduce the number of tests:

• start at the innermost loop; set all other loops to minimum values
• conduct simple loop test; add out of range or excluded values
• work outwards while keeping inner nested loops to typical values
• continue until all loops have been tested
Control-Flow Coverage Criteria Summary

**Subsumption:**

A criterion C1 subsumes another criterion C2, if any test set \{T\} that satisfies C1 also satisfies C2.

Diagram:

- Multiple Condition
- MC/DC
- C/DC
- Condition
- Decision
- Statement
- All Path
- Linearly Indep. Path
WBT – Advantages & Disadvantages

Advantages

• Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
• Testing is more thorough than ECP, with the possibility of covering most paths.

Disadvantages

• Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
• Test script maintenance can be a burden if the implementation changes too frequently.
Structure of Lecture 4

- Code Coverage Introduction
- Control-Flow Criteria
  - Control Flow Graph
  - Node Coverage
  - Branch Coverage
  - Condition Coverage
  - Independent Path Coverage
  - Loop Coverage
  - Summary
- Lab 4
Lab 4: Basic White-Box Testing

HW 4 (week 29: Mar 15 & 16) – Basic White-Box Testing (9 points)

Submission Deadlines:
- Tuesday Labs: Monday, 21 Mar, 23:59
- Wednesday Labs: Tuesday, 22 Mar, 23:59

- Penalties apply for late delivery: 50% penalty, if submitted up to 24 hours late; 100% penalty, if submitted more than 24 hours late
Lab 4: Basic White-Box Testing (cont’d)

Coverage Criteria:
(a) Instruction/Statement (Line)
(b) Branch (Decision)
Tool: IntelliJ IDEA (JaCoCo) or Eclipse plugin (EclEmma)
Lab 4: Basic White-Box Testing (cont’d)

Instructions

Code

Control-Flow Graph
Set of 10+ Test Cases 1

Coverage Criteria:
(a) Instruction/Statement (Line)
(b) Branch (Decision)
Tool: IntelliJ IDEA (JaCoCo) or Eclipse plugin (EclEmma)

Code

Test Report 1 &
Test Coverage 1a + 1b

![Test Report and Coverage Screenshot]
Lab 4: Basic White-Box Testing (cont’d)

Instructions

Control-Flow Graph
Set of 10+ Test Cases 1
Set of 15+ Test Cases 2

Goal: increase (b)

Code

Coverage Criteria:
(a) Instruction/Statement (Line)
(b) Branch (Decision)
Tool: IntelliJ IDEA (JaCoCo) or Eclipse plugin (EclEmma)

Test Report 1 &
Test Coverage 1a + 1b

Test Report 2 &
Test Coverage 2a + 2b

Code

Code

Tool:
IntelliJ IDEA (JaCoCo) or
Eclipse plugin (EclEmma)
To Do & Next Week

• Quiz 4 (in Moodle!):
  • Opens at end of lecture – closes on Monday at 17:30am!

• HW 3:
  – Combinatorial Testing: finish up and submit on time!

• Lab 4:
  – Basic White-Box Testing

• Lecture 5: