Lecture 04: Basic White-Box Testing (Textbook Ch. 5)

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Spring 2019
Lectures (J. Liivi 2-111)

Lecture 1 (14.02) – Introduction to Software Testing
Lecture 2 (21.02) – Basic Black-Box Testing Techniques
Lecture 3 (28.02) – BBT advanced: Combinatorial Testing
Lecture 4 (07.03) – Basic White-Box Testing Techniques
Lecture 5 (14.03) – Test Lifecycle, Test Tools, Test Automation
Lecture 7 (28.03) – BBT advanced: State-Transition Testing & Exploratory Testing
Lecture 8 (04.04) – BBT advanced: Usability Testing and A/B Testing
Lecture 9 (11.04) – WBT advanced: Data-Flow Testing / Mutation Testing
Lecture 11 (25.04) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
Lecture 12 (09.05) – Industry Guest Lecture (to be announced)
Lecture 13 (16.05) – Exam Preparation
Structure of Lecture 4

• Code Coverage Introduction
• Control-Flow Criteria
  • Branch Coverage
  • Condition Coverage
  • Independent Path Coverage
  • Loop Coverage
  • Summary
• Lab 4
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?

1. Develop an initial Test suite using BB techniques
2. Analyze the parts of the code uncovered by BB test suite
3. Apply BB coverage criteria to enhance it
4. Enhance the Test suite using WB techniques
5. Apply WB coverage criteria to enhance it
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

There are many possible paths! $5^{20} (\sim 10^{14})$ different paths

Selective Testing
Selective Testing

2 Major Strategies

- Control flow testing
- Data flow testing

a selected path
Code Coverage (Test 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

Control Flow Criteria:

Statement, 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’).
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

```java
public boolean addAll(int index, Collection c) {
    if (c.isEmpty()) {
        return false;
    } else if (_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListItemAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while (it.hasNext()) {
            pred = insertListItem(pred, succ, it.next());
        }
        return true;
    }
}
```

Line coverage

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

View coverage results:

In the Project tool window:

In the dedicated Coverage tool window:
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.
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.
Structure of Lecture 4

• Code Coverage Introduction
• Control-Flow Criteria
  • Branch Coverage
  • Condition Coverage
  • Independent Path Coverage
  • Loop Coverage
  • Summary
• Lab 4
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;
}
z = x+y;
```

Control Flow Graph

- **B₁**: x = z-2; y = 2*z;
- **B₂**: x = x+1; y = y+1;
- **B₃**: x = x-1; y = y-1;
- **B₄**: z = x+y;

- **c=T**: Transition from **B₁** to **B₂**
- **c=F**: Transition from **B₁** to **B₃**
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;
}
```

Control Flow Graph

Blocks (=Nodes): 4
Edges: 4

- \( B_1 \):
  - \( x = z-2; \)
  - \( y = 2*z; \)

- \( c = T \) branch:
  - \( B_2 \):
    - \( x = x+1; \)
    - \( y = y+1; \)

- \( c = F \) branch:
  - \( B_3 \):
    - \( x = x-1; \)
    - \( y = y-1; \)

- \( B_4 \) (empty)
Control Flow Graph (CFG)

d₁ is a 'dummy node'

entry and exit nodes are 'dummy nodes'

Nodes: 8
Edges: 8

B₁
x = z-2;
y = 2*z;

B₂
x = x+1;
y = y+1;
c=T

B₃
x = x-1;
y = y-1;
c=F

B₄
empty

Blocks: 4
Edges: 4
Control Flow – Example

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

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
} else {
  if (d4) then {
    repeat {s4} until (c5)
  }
}

CFG(t) ➔ d1 ➔ e1 ➔ e2 ➔ CFG(f)
Control Flow – Example

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

If (d1) then {
  CFG(d1=true)
}
else {
  CFG(d1=false)
}
Control Flow – Example

If (d1) then {
  if (d2) then {s1}
  s2
  while (d3) do {s3}
} else {
  if (d4) then {
    repeat {s4} until (c5)
  }
}

CFG(if)

CFG(while)

CFG(repeat)
Control Flow – Example

\[
\text{If (d1) then } \begin{cases} 
\text{if (d2) then } \{s1\} \\
\text{s2} \\
\text{while (d3) do } \{s3\} 
\end{cases} \\
\text{else } \begin{cases} 
\text{if (d4) then } \{ \text{repeat } \{s4\} \text{ until (d5)} \} 
\end{cases}
\]
Control Flow – Example

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

- Statement (or Block) 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
...
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;
}
```
Statement Coverage

Execute each statement at least once

Use tools to monitor execution

More practice in Lab 2

A possible concern may be:

Dead code
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
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:

0 %
Statement Coverage /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)->true

40 %
### Statement Coverage \( /3 \)

```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)->true
AccClient(83, male)->false

80 %
Statement Coverage /4

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 %
Same Test Suite but Incorrect Code in Life Insurance Example (1)

```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);
```

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

80 %

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

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

1: if (gender == female & age < 80) return (TRUE);
2: return (TRUE);
3: 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 Example (2)

```java
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)->false
- AccClient(83, male)->false
- AccClient(25, male)->true

80 %
Same Test Suite but Incorrect Code in Life Insurance Example (2)

```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 %
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

Statement Coverage: Dead Code?

Dead code?

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 %
boolean AccClient(int age; gtype gender)

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

Test:

0 %
Decision (Branch) Coverage /2

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 %
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

75 %
Decision (Branch) Coverage /4

```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)->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
Condition Coverage /1

```cpp
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:

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

50 % or 25 %
**Condition Coverage /3**

```java
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

100 % or 62.5 %
Condition Coverage /4

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 % or 75 %
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==fem & B<85) ...

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

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

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

Multiple Condition (M-CC):
(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)

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

If (A=fem and 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
Paths 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.

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: 1-0-0-1-0-0-1-0-0-1-0-0-0-0-0
Path2: 1-0-1-0-1-1-1-1-1-0-0-0-0-0-0
Path3: 1-0-0-1-0-1-1-1-1-0-0-0-0-0-0
Path4: 0-1-0-0-0-0-0-0-0-1-0-1-0-1-0
Path5: 0-1-0-0-0-0-0-0-0-1-0-1-1-1-1
Path6: 0-1-0-0-0-0-0-0-0-0-1-0-0-0-0
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: 1-0-1-0-1-0-1-0-1-0-0---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: 1-0-0-1-0-1-0-1-0---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: 1-0-1-0-1-0-0-1-0---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

\( \ldots \) 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
**Subsumption:**

a criterion C1 subsumes another criterion C2, if any test set \{T\} that satisfies C1 also satisfies C2
Structure of Lecture 4

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

Lab 4 (week 28: Mar 12 & 13) – Basic White-Box Testing (10 points)

WBT Instructions
WBT Sample Code

Submission Deadlines:
• Tuesday Labs: Monday, 18 Mar, 23:59
• Wednesday Labs: Tuesday, 19 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)

Instructions

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

Coverage Criteria:
- Instruction/Statement (Line)
- Branch (Decision)
Tool: IntelliJ IDEA or Eclipse Plugin

Test Report 1 &
Test Coverage 1a + 1b

Test Report 2 &
Test Coverage 2a + 2b
Recommended Textbook Exercises

Chapter 5

2, 5, 6, 9, 10, 11, 14
To Do & Next Week

• Quiz 4 (in Moodle!):
  • Opens tomorrow morning – closes on Monday at 11:30am!

• Lab 4:
  – Basic White-Box Testing

• Lecture 5:
  – Test Lifecycle, Test Levels, Test Tools

• In addition to do:
  – Read textbook chapters 5