MTAT.03.159: Software Testing

Lecture 02:
Basic Black-Box and White-Box Testing Techniques
(Textbook Ch. 4 & 5)
Spring 2018

Dietmar Pfahl
email: dietmar.pfahl@ut.ee
Structure of Lecture 2

• Black-Box vs. White-Box Testing
• Basic Black-Box Testing Techniques
• Basic White-Box Testing Techniques
• Lab 2
Testing is difficult

Assume a ’magic’ Function \( M \)

\[ M (x, y) \rightarrow z \]
with \( x, y \): int (32 bit)

Exhaustive testing:

How many test cases,

If only valid input (=int) used?
Testing is difficult

Assume a ’magic’ Function M

\[ M(x, y) \rightarrow z \]

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

Exhaustive pos. testing:

\[ 2^{32} \times 2^{32} = 2^{64} \approx 1.8 \times 10^{19} \text{ test cases (input data + expected output)} \]
Testing is difficult

Assume a ’magic’ Function M

\[ M(x, y) \rightarrow z \]

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

Possible approaches:
- ???

White Box

```c
... 
if ( x - 100 <= 0 ) {
    if ( y - 100 <= 0 ) {
        if ( x + y - 200 == 0 ) {
            z = x / (y - 100);
        }
    }
}
... 
```
Testing is difficult

Assume a 'magic' Function M

M (x, y) \rightarrow z

with x, y: int (32 bit)

Possible approaches:
- Execute each statement
- Read (review) code

White Box

```java
...
if ( x - 100 <= 0 ) {
    if ( y - 100 <= 0 ) {
        if ( x + y - 200 == 0 ) {
            z = x / (y - 100);
        }
    }
}
```
Testing is difficult

Assume a 'magic' Function M

\[ M(x, y) \rightarrow z \]

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

1st if = true: \( x \leq 100 \)

2nd if = true: \( y \leq 100 \)

3rd if = true: \( x + y = 200 \)

\[ M(100, 100) \rightarrow \text{crash} \]

White Box

\[
\begin{align*}
\text{...} \\
&\text{if } (x - 100 \leq 0) \\
&\quad \text{if } (y - 100 \leq 0) \\
&\quad \quad \text{if } (x + y - 200 = 0) \\
&\quad \quad \quad z = x / (y - 100); \\
&\quad \}
&\}
&\}
&\}
&... 
\end{align*}
\]
Black-Box vs. White-Box

External/user view:
- Check conformance with specification -> function coverage
- Abstraction from details:
  - Source code not needed
- Scales up:
  - Different techniques at different levels of granularity

Internal/developer view:
- Allows tester to be confident about code coverage
- Based on control and data flow:
  - Easier debugging
- Does not scale up:
  - Most useful at unit & integration testing levels, as well as regression testing

USE BOTH!
# Black-Box vs. White-Box

## Gray-Box Testing

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Combines black-box and white-box testing; typically, the focus is on input/output testing (black-box view) which is informed by structural information of the code (white-box view).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ab.</td>
<td>Example: The tester knows that certain constraints on the input are checked by the unit under test.</td>
</tr>
<tr>
<td>Sc.</td>
<td>Application, e.g., in regression testing: apply (or update) black-box test cases only where code has been changed;</td>
</tr>
<tr>
<td>USE</td>
<td>Most useful at unit &amp; integration testing levels, as well as regression testing</td>
</tr>
</tbody>
</table>

- Different techniques at different levels of granularity
- USE BOTH!
Black-Box vs. White-Box

System

Specification

Implementation
Black-Box vs. White-Box

Specification-based Testing:
Test against specification

Goal of BBT: Tries to check whether specified functionality is available and working correctly
Black-Box vs. White-Box

Specification-based Testing:
Test against specification

Goal of BBT: Tries to check whether specified functionality is available and working correctly

Unexpected functionality: Cannot be (directly) revealed by black-box techniques
Black-Box vs. White-Box

System

Specification

Implementation

Structural Testing: Test against implementation

Goal of WBT: Tries to check, whether the Implementation is working correctly (there is no dead code, it’s maintainable, etc.); useful for debugging;
Black-Box vs. White-Box

System

Specification

Implementation

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

Structural Testing: Test against implementation

Goal of WBT: Tries to check, whether the Implementation is working correctly (there is no dead code, it’s maintainable, etc.); useful for debugging;
Black-Box vs. White-Box

Specification-based Testing: Test against specification

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

Structural Testing: Test against implementation

Unexpected functionality: Cannot be (directly) revealed by black-box techniques
How do Black-Box and White-Box Testing relate to one another?
Structure of Lecture 2

• Black-Box vs. White-Box Testing
• Basic Black-Box Testing Techniques
• Basic White-Box Testing Techniques
• Lab 2
Black-Box Testing Techniques

• Equivalence class partitioning (ECP)
• Boundary value analysis (BVA)
• Combinatorial testing
• State transition testing (State-based testing)
• Exploratory testing
• Usability testing
• A/B testing (UX)

Lecture 3
Equivalence Class Partitioning (ECP)

- Split input/output space into classes which the software handles equivalently
- Select test cases to cover each class

Input

Output

green = valid
white = invalid
ECP – Simple Example

public static boolean adultFunction(int age) {
    boolean adult;
    if (age >= 18)
        adult = true;
    else
        adult = false;
    return adult;
}

What are the ECs?

green = valid
white = invalid
ECP Guidelines [Myers 79/04]

If input condition:

is a range, e.g., \( x = [0, 9] \) or

is an ordered list of values, e.g., \( \text{owner} = <1, 2, 3, 4> \)

\( \rightarrow \) one in-range/list and two out-of-range/list classes are defined

is a set, e.g., \( \text{vehicle} \) is in \{car, motorcycle, truck\} or

is a “must be” condition (boolean)

\( \rightarrow \) one in-set and one out-of-set class are defined

is anything else (e.g., invalid)

\( \rightarrow \) partition further

Union set of all ECs should cover complete input/out space. ECs must not overlap.

Have enough test cases to cover all valid input classes
Have one test case for each invalid input class
Example – Insurance System

Specification Statement:

• System shall reject over-age insurance applicants

Specification Item:

• Reject male insurance applicants, if over the age of 80 years on day of application

• Reject female insurance applicants, if over the age of 85 years on day of application
Example (cont.)
Input: Gender & Age | Output: accept/reject

UI – Case A

Age:  
in \([18, 80]\)? O
        in \((80, 85]\)? O
        in \((85, 99]\)? O
        other ? O

Gender:  
        male O
        female O

Enter

Result: <text>

Message: <text>

Result text in \{<empty>, accept, reject\}
Message text in \{<empty>, missing input\}

UI – Case B

Please enter gender (m, f):
<Message text>

Please enter age (integer>0):
<Message text>
Result: <text>

Message text in \{<empty>, invalid input – retry or quit with Ctrl^D}\nResult text in \{accept, reject\}
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Classes

C1: ???

...
Example – UI Case A

Input: Gender & Age  |  Output: accept/reject

Classes

C1: InputAge: [18, 80]
C2: InputAge: (80, 85]
C3: InputAge: (85, 99]
C4: InputAge: other
C5: InputAge: <empty>
C6: InputGender: Male
C7: InputGender: Female
C8: InputGender: <empty>
C9: OutputResult: <empty>
C10: OutputResult: ‘accept’
C11: OutputResult: ‘reject’
C12: OutputMsg: <empty>
C13: OutputMsg: ’missing input’

What do you say about: C1, C3, and C4?
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Classes

C1: InputAge: [18, 80]
C2: InputAge: (80, 85]
C3: InputAge: (85, 99]
C4: InputAge: other
C5: InputAge: <empty>
C6: InputGender: Male
C7: InputGender: Female
C8: InputGender: <empty>
C9: OutputResult: <empty>
C10: OutputResult: ‘accept’
C11: OutputResult: ‘reject’
C12: OutputMsg: <empty>
C13: OutputMsg: ’missing input’

Test Cases

Data: age, gender, result, message

How many test cases to cover all classes?
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Classes

C1: InputAge: [18, 80]
C2: InputAge: (80, 85]
C3: InputAge: (85, 99]
C4: InputAge: other
C5: InputAge: <empty>
C6: InputGender: Male
C7: InputGender: Female
C8: InputGender: <empty>
C9: OutputResult: <empty>
C10: OutputResult: ‘accept’
C11: OutputResult: ‘reject’
C12: OutputMsg: <empty>
C13: OutputMsg: ’missing input’

Test Cases

Data: age, gender, result, message

TC1: <empty>, <empty>, <empty>, ’missing input’
TC2: 56, male, ’accept’, <empty>
TC3: 83, male, ’reject’, <empty>
TC4: 88, female, ’reject’, <empty>
TC5: other, female, ’reject’, <empty>

minimal,
TCs cover all classes
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Classes

C1: InputAge: [18, 80]
C2: InputAge: (80, 85]
C3: InputAge: (85, 99]
C4: InputAge: other
C5: InputAge: <empty>
C6: InputGender: Male
C7: InputGender: Female
C8: InputGender: <empty>
C9: OutputResult: <empty>
C10: OutputResult: ‘accept’
C11: OutputResult: ‘reject’
C12: OutputMsg: <empty>
C13: OutputMsg: ’missing input’

Test Cases

TC1: <empty>, male, <empty>, ’missing input’
TC2: other, <empty>, <empty>, ’missing input’
TC3: 56, male, ’accept’, <empty>
TC4: 83, male, ’reject’, <empty>
TC5: 88, female, ’reject’, <empty>
TC6: other, female, ’reject’, <empty>

If we consider ’missing input’ to be an error message caused by invalid input (<empty>), then it’s good practice to check for the effect of each invalid input class independently
Example – UI Case A
Input: Gender & Age | Output: accept/reject

<table>
<thead>
<tr>
<th>Input</th>
<th>Valid EC</th>
<th>Invalid EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>C1: [18, 80]</td>
<td>C5: &lt;empty&gt;</td>
</tr>
<tr>
<td></td>
<td>C2: (80, 85]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C3: (85, 99]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C4: other</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>C6: Male</td>
<td>C8: &lt;empty&gt;</td>
</tr>
<tr>
<td></td>
<td>C7: Female</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>em</td>
<td>oth</td>
<td>56</td>
<td>83</td>
<td>88</td>
<td>oth</td>
</tr>
<tr>
<td>Gend.</td>
<td>M</td>
<td>em</td>
<td>M</td>
<td>M</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Tests</td>
<td>C5</td>
<td>C8</td>
<td>C1</td>
<td>C6</td>
<td>C2</td>
<td>C6</td>
</tr>
</tbody>
</table>
Example – UI Case A
Input: Gender & Age | Output: accept/reject

<table>
<thead>
<tr>
<th>Input</th>
<th>Valid EC</th>
<th>Invalid EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>C1: [18, 80]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2: (80, 85)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C3: (85, 99)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C4: other</td>
<td>C5: &lt;empty&gt;</td>
</tr>
<tr>
<td>Gender</td>
<td>C6: Male</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C7: Female</td>
<td>C8: &lt;empty&gt;</td>
</tr>
</tbody>
</table>
Example – UI Case A
Input: Gender & Age | Output: accept/reject

<table>
<thead>
<tr>
<th>Input</th>
<th>Valid EC</th>
<th>Invalid EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>C1: [18, 80]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2: (80, 85)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C3: (85, 99)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C4: other</td>
<td>C5: &lt;empty&gt;</td>
</tr>
<tr>
<td>Gender</td>
<td>C6: Male</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C7: Female</td>
<td>C8: &lt;empty&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age Gend.</td>
<td>em M</td>
<td>oth M</td>
<td>56 M</td>
<td>83 M</td>
<td>88 F</td>
<td>oth F</td>
</tr>
<tr>
<td>Tests</td>
<td>C5</td>
<td>C8</td>
<td>C1 C6</td>
<td>C2 C6</td>
<td>C3 C7</td>
<td>C4 C7</td>
</tr>
</tbody>
</table>

Must also check coverage of output ECs!
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Classes

C1: InputAge: [18, 80]
C2: InputAge: (80, 85]
C3: InputAge: (85, 99]
C4: InputAge: other
C5: InputAge: <empty>
C6: InputGender: Male
C7: InputGender: Female
C8: InputGender: <empty>
C9: OutputResult: <empty>
C10: OutputResult: ‘accept’
C11: OutputResult: ‘reject’
C12: OutputMsg: <empty>
C13: OutputMsg: ‘missing input’

Test Cases

TC1: <empty>, male, <empty>, ’missing input’
TC2: other, <empty>, <empty>, ’missing input’
TC3: 56, male, ’accept’, <empty>
TC4: 83, male, ’reject’, <empty>
TC4*: 83, female, ’accept’, <empty>
TC5: 88, female, ’reject’, <empty>
TC6: other, female, ’reject’, <empty>

Now, we have covered all cause-effect relationships (--> Cause-Effect Graphing)
Example – UI Case B

Input: Gender & Age | Output: accept/reject

Classes

C1: ???

...
Example – UI Case B

Input: Gender & Age | Output: accept/reject

Classes

C1: InputGender: m
C2: InputGender: f
C3: InputGender: not(m, f)
C4: InputGender: Ctrl^D
C5: InputAge: integer in [18, 80]
C6: InputAge: integer in (80, 85]
C7: InputAge: integer <18
C8: InputAge: integer >85
C9: InputAge: Ctrl^D
C10: InputAge: other than C5-C9
C11: OutputMsg: <empty>
C12: OutputMsg: 'invalid input ...'
C13: OutputResult: accept
C14: OutputResult: reject

Please enter gender (m, f):
<Message text>
Please enter age (integer>0):
<Message text>
Result: <text>

Message text in {
<empty>,
invalid input – retry or quit with Ctrl^D}
Result text in {
accept,
reject}
Example – UI Case B

Input: Gender & Age | Output: accept/reject

Classes

C1: InputGender: m
C2: InputGender: f
C3: InputGender: not(m, f)
C4: InputGender: Ctrl^D
C5: InputAge: integer in [18, 80]
C6: InputAge: integer in (80, 85]
C7: InputAge: integer <18
C8: InputAge: integer >85
C9: InputAge: Ctrl^D
C10: InputAge: other than C5-C9
C11: OutputMsg: <empty>
C12: OutputMsg: ’invalid input ...
C13: OutputResult: accept
C14: OutputResult: reject

Test Cases

TC1: Ctrl^D
TC2: not(m, f), ’invalid’, Ctrl^D
TC3: m, <empty>, Ctrl^D
TC4: m, <empty>, other, ’invalid’, Ctrl^D
TC5: m, <empty>, [18, 80], <empty>, accept
TC6: m, <empty>, (80, 85], <empty>, reject
TC7: f, <empty>, <18, <empty>, reject
TC8: f, <empty>, >85, <empty>, reject
...
Example – UI Case B
Input: Gender & Age | Output: accept/reject

Classes
C1: InputGender: m
C2: InputGender: f
C3: InputGender: not(m, f)
C4: InputGender: Ctrl^D
C5: InputAge: integer in [18, 80]
C6: InputAge: integer in (80, 85]
C7: InputAge: integer <18
C8: InputAge: integer >85
C9: InputAge: Ctrl^D
C10: InputAge: other than C5-C9
C11: OutputMsg: <empty>
C12: OutputMsg: ’invalid input ...’
C13: OutputResult: accept
C14: OutputResult: reject

Test Cases
TC1: Ctrl^D
TC2: g, ’invalid’, Ctrl^D
TC3: m, <empty>, Ctrl^D
TC4: m, <empty>, 3.5, ’invalid’, Ctrl^D
TC5: m, <empty>, 56, <empty>, accept
TC6: m, <empty>, 83, <empty>, reject
TC7: f, <empty>, 5, <empty>, reject
TC8: f, <empty>, 103, <empty>, reject
...
Example – UI Case B

Input: Gender & Age | Output: accept/reject

Classes

C1: InputGender: m
C2: InputGender: f
C3: InputGender: not(m, f)
C4: InputGender: Ctrl^D
C5: InputAge: integer in [18, 80]
C6: InputAge: integer in (80, 85]
C7: InputAge: integer <18
C8: InputAge: integer >85
C9: InputAge: Ctrl^D
C10: InputAge: other than C5-C9
C11: OutputMsg: <empty>
C12: OutputMsg: ’invalid input ...’
C13: OutputResult: accept
C14: OutputResult: reject

Test Cases

Every path from the root to a leaf is (at least) one test case
Boundary Value Analysis

• Adds to the equivalence partitioning method
• Select test cases to represent each side of the class boundaries
Boundary Value Analysis Guidelines

- Range $a..b \Rightarrow a$, $b$, just above $a$, just below $b$
- List of values $\Rightarrow$ max, min, just below min, just above max

- Boundaries of externally visible data structures shall be checked (e.g. ordered sets, arrays)
- Output bounds should be checked
Example – UI Case B
Input: Gender & Age | Output: accept/reject

Classes
C1: InputGender: m
C2: InputGender: f
C3: InputGender: not(m, f)
C4: InputGender: Ctrl^D
C5: InputAge: integer in [18, 80]
C6: InputAge: integer in (80, 85]
C7: InputAge: integer <18
C8: InputAge: integer >85
C9: InputAge: Ctrl^D
C10: InputAge: other than C5-C9
C11: OutputMsg: <empty>
C12: OutputMsg: 'invalid input ...'
C13: OutputResult: accept
C14: OutputResult: reject

Test Cases
TC1: Ctrl^D
TC2: g, 'invalid', Ctrl^D
TC3: m, <empty>, Ctrl^D
TC4: m, <empty>, 3.5, 'invalid', Ctrl^D
TC5: m, <empty>, 56, <empty>, accept
TC5L: m, <empty>, 18, <empty>, accept
TC5U: m, <empty>, 80, <empty>, accept
TC6: m, <empty>, 83, <empty>, reject
... 
TC7: f, <empty>, 5, <empty>, reject
TC8: f, <empty>, 103, <empty>, reject
...
Combinatorial Designs

- ECP and BVA define test cases per class
- Combinations of equivalence classes need to be handled
- Combinatorial explosion needs to be handled
Structure of Lecture 2

• Black-Box vs. White-Box Testing
• Basic Black-Box Testing Techniques
• Basic White-Box Testing Techniques
• Lab 2
White-Box Testing Techniques

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

Lectures 4 & 6
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 Test Adequacy 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.

Note:
EclEmma requires Eclipse

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

public boolean addAll(int index, Collection c) {
    if (c.isEmpty()) {
        return false;
    } else if (_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while (it.hasNext()) {
            pred = insertListable(pred, succ, it.next());
        }
        return true;
    }
}
Code Coverage Measure – IntelliJ Code Coverage Tool

View coverage results:

In the Project tool window:

In the dedicated Coverage tool window:
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.
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.
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

- **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**
- **c=F**
Control Flow Graph (CFG)

Program

```c
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
Control Flow Graph (CFG)

d₁ is a 'dummy node'

Nodes: 8
Edges: 8

entry and exit nodes are 'dummy nodes'

Blocks: 4
Edges: 4

\[ x = z - 2; \]
\[ y = 2 \times z; \]

\[ x = x + 1; \]
\[ y = y + 1; \]

\[ x = x - 1; \]
\[ y = y - 1; \]
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
} else {
  if (d4) then {
    repeat {s4} until (c5)
  }
}

CFG(f)

e₁

CFG(t)

e₂

d₁

If (d1) then {
  CFG(d1=true)
} else {
  CFG(d1=false)
}

else {
  }

Else { }
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)
  }
}

If (d1) then {
  CFG(if)
  s2
  CFG(while)
} else {
  if (d4) then {
    CFG(repeat)
  }
}

CFG(if)
CFG(while)
Control Flow – Example

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

MTAT.03.159 / Lecture 02 / © Dietmar Pfahl 2018
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
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);

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

Test:

0 %
Statement Coverage /2

```c
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 %
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 %
boolean AccClient(int age; gtype gender)

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

Where is the bug?

1 failure
80 %
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);
```

1 fault triggers 1 failure

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

1 fault
1: if (gender == female & age > 85)
1 failure
d1 = c1 & c2

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);
```

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

100 %

2 faults trigger
0 failures
Statement Coverage : Dead Code ?

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);

Dead code ?

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

MTAT.03.159 / Lecture 02 / © Dietmar Pfahl 2018
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

100 %
### Decision (Branch) Coverage /1

```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:

0 %
Decision (Branch) 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

25 %
Decision (Branch) Coverage /3

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

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

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

75 %
Decision (Branch) Coverage /4

boolean AccClient(int age; gtype gender)

1: if (gender == female & age < 85)
   return (TRUE);

2: if (gender == male & age < 80)
   return (TRUE);

3: 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) return(TRUE);
2: if (gender == male & age < 80) return(TRUE);
3: if (gender == unknown & age < 100) return(FALSE);
4: return(FALSE);

Test:

0 %
Condition 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

50 % or 25 %
Condition 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

100 % or 62.5 %
Condition Coverage /4

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

1: if (gender == female & age < 85)
   return (TRUE);
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 \( (A==\text{fem} \& B<85) \) …

Minimum and Simple Condition (CC):

(TF) \( A = \text{fem}; B = 200 \) (D: False)

(FT) \( A = \text{male}; B = 80 \) (D: False)

Decision (DC):

(TT) \( A = \text{fem}; B = 80 \) (D: True)

(FT) \( A = \text{male}; B = 80 \) (D: False)

Condition/Decision (C/DC):

(TT) \( A = \text{fem}; B = 80 \) (D: True)

(FF) \( A = \text{male}; B = 200 \) (D: False)

Multiple Condition (M-CC):

(TT) \( A = \text{fem}; B = 80 \) (D: True)

(FT) \( A = \text{male}; B = 80 \) (D: False)

(TF) \( A = \text{fem}; B = 200 \) (D: False)

(MF) \( A = \text{male}; B = 200 \) (D: False)

Modified Condition/Decision (MC/DC):

(TT) \( A = \text{fem}; B = 80 \) (D: True)

(FT) \( A = \text{male}; B = 80 \) (D: False)

(TF) \( A = \text{fem}; B = 200 \) (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)

Modified Condition/Decision:
- (TT) A = fem; B = 80 (D: True)
- (FT) A = male; B = 80 (D: False)
- (TF) A = fem; 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: 1-0-0-1-0-1-0-0-1-0---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---0---1---1---1
Path6: 0-1-0-0-0-0-0-0-0---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

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

-P3: 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
Control-Flow Coverage Relationships

**Subsumption:**

A criterion $C_1$ subsumes another criterion $C_2$, if any test set $\{T\}$ that satisfies $C_1$ also satisfies $C_2$.

Diagram:
- Multiple Condition
  - MC/DC
    - C/DC
      - Condition
      - Decision
        - Statement
  - All Path
    - Linearly Indep. Path
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
Structure of Lecture 2

• Black-Box vs. White-Box Testing
• Basic Black-Box Testing Techniques
• Basic White-Box Testing Techniques
• Lab 2
Lab 2: Part 1 – Black-Box Testing

Lab 2 (week 27: Mar 06 - Mar 07) – Black-Box Testing (5%)

BBT Instructions
BBT Documentation
BBT Application

Submission Deadlines:
• Tuesday Labs: Monday, 13 Mar, 23:59
• Wednesday Labs: Tuesday, 14 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 2: Part 1 – Black-Box Testing (cont’d)

Test Cases:
Input & Exp. Output
1 3 4 → triangle type x
2 5 5 → triangle type y
0 1 2 → no triangle
...

Strategies:
- Equivalence Class Partitioning
- Boundary Value Analysis

Test Report:
TC1 → pass
TC2 → pass
TC3 → pass
TC4 → fail → defect
...

Program
Lab 2: Part 2 – White-Box Testing

Lab 2 (week 22: Mar 06 - Mar 07) - White-Box Testing (5%)

WBT Instructions
WBT Sample Code

Submission Deadlines:
- Tuesday Labs: Monday, 13 Mar, 23:59
- Wednesday Labs: Tuesday, 14 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 2: Part 2 – White-Box Testing (cont’d)

Instructions

Code

Control-Flow Graph

Set of 10+ Test Cases 1
Set of 15+ Test Cases 2

Test Report 1 &
Test Coverage 1a + 1b

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

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

Chapter 4
1, 2, 3, 4, 5
8, 11, 12

Chapter 5
2, 5, 6, 9, 10,
11, 14
Next two weeks …

• Lab 2:
  – Black-Box and White-Box Testing

• Lecture 3:
  – Advanced Black-Box Testing Techniques

• In addition to do:
  – Read textbook chapters 4 & 5 (available via OIS)