Lecture 03:
White-Box Testing
(Textbook Ch. 5)

Dietmar Pfahl
email: dietmar.pfahl@ut.ee

Spring 2017
Lecture

Chapter 5

• White-box testing techniques (Lab 3)
Structure of Lecture 3

• Introduction to White-Box Testing
• Control-Flow Testing
• Loop Testing
• Data-Flow Testing
• Lab 2
Testing Strategies

Black Box Testing

White Box Testing
White-Box Testing

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

Selective Testing
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
Selective Testing

2 Major Strategies

- Control flow testing
- Data flow testing
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:
- EclEmma -> Lab 2
- Clover
- etc.

Note:
EclEmma requires Eclipse
Code Coverage Measure – EclEmma

Branch coverage

Line coverage

```
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;
    }
}
```

http://www.eclemma.org/index.html
Structure of Lecture 3

• Introduction to White-Box Testing
• Control-Flow Testing
• Loop Testing
• Data-Flow Testing
• Lab 2
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;
}
z = x+y;
```

Control Flow Graph

```
B1: x = z-2;
y = 2*z;
T  F
B2: x = x+1;
y = y+1;
B3: x = x-1;
y = y-1;
B4: z = x+y;
```
Control Flow Graph (CFG)

Program

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

Control Flow Graph

\[
\begin{align*}
B_1: &\quad x = z - 2; \\
&\quad y = 2 \times z; \\
T: &\quad x = x + 1; \\
&\quad y = y + 1; \\
F: &\quad x = x - 1; \\
&\quad y = y - 1; \\
B_2: &\quad x = x + 1; \\
&\quad y = y + 1; \\
B_3: &\quad x = x - 1; \\
&\quad y = y - 1; \\
B_4: &\quad \text{empty}
\end{align*}
\]

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

d_1 is a 'dummy node'

\(e_3\) \(e_4\)

\(s_1\) \(s_2\)

\(e_2\)

\(e_1\)

\(s_3\) \(s_4\)

\(e_7\) \(e_8\)

\(s_5\) \(s_6\)

Nodes: 8
Edges: 8

entry and exit nodes are 'dummy nodes'

\(d_1\) is a 'dummy node'

Control Flow Graph

B_1

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

T

B_2

\(x = x + 1;\)
\(y = y + 1;\)

F

B_3

\(x = x - 1;\)
\(y = y - 1;\)

B_4

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(f)
CFG(t)
Control Flow – Example

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

Diagram:

- Nodes: s1, s2, s3, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13, e14, d1, d2, d3, d4, d5
- Edges: s1 → e1, e1 → d1, d1 → e2, e2 → d2, d2 → e3, e3 → s2, s2 → e4, e4 → e6, e6 → e5, e5 → s3, s3 → e7, e7 → d3, d3 → e8, e8 → e9, e9 → e10, e10 → e11, e11 → e12, e12 → e13, e13 → e14, e14 → d5, d5 → d4, d4 → e10, e10 → s4, s4 → e12, e12 → e13, e13 → e11, e11 → d1

MTAT.03.159 / Lecture 03 / © Dietmar Pfahl 2017
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
Statement 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:

0 %
Statement 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

40 %
Statement Coverage /3

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)
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

Where is the bug?

80 %
Same Test Suite but Incorrect Code in Life Insurance Example (1)

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

1 fault
triggers
1 failure

1 fault
1 failure

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

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)->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

100 %

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

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

Dead code ?

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
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

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 %
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);

T/F
d1 = c1 & c2

d2 = c3 & c4

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

Test:

0 %
**Condition Coverage /2**

Boolean function `AccClient`:

```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) -> reject`

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 %
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

Path 1: 1-0-0-1-0-0-1-0-0-1---0---0---0---0
Path 2: 1-0-1-0-1-1-1-1-1-0---0---0---0---0
Path 3: 1-0-0-1-0-1-1-1-1-0---0---0---0---0
Path 4: 0-1-0-0-0-0-0-0-0-1---0---1---0---1
Path 5: 0-1-0-0-0-0-0-0-0-1---0---1---1---1
Path 6: 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: 1-0-1-0-1-1-0-1-0---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: 1-0-0-1-0-1-0-0-1-0---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 $C_2$

---

```
\begin{tikzpicture}
    \node (s) {Statement};
    \node (d) [above of=s] {Decision};
    \node (c) [above of=d] {Condition};
    \node (mc) [above of=c] {C/DC};
    \node (a) [above of=mc] {MC/DC};
    \node (ap) [above of=a] {All Path};
    \node (lip) [left of=ap] {Linearly Indep. Path};
    \node (mp) [left of=a] {Multiple Condition};

    \draw[->] (s) -- (d);
    \draw[->] (d) -- (c);
    \draw[->] (c) -- (mc);
    \draw[->] (mc) -- (a);
    \draw[->] (a) -- (ap);
    \draw[->] (ap) -- (lip);
    \draw[->] (mp) -- (a);
\end{tikzpicture}
```
Structure of Lecture 3

• Introduction to White-Box Testing
• Control-Flow Testing
• Loop Testing
• Data-Flow Testing
• Lab 2
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 3

• Introduction to White-Box Testing
• Control-Flow Testing
• Loop Testing
• Data-Flow Testing
• Lab 2
Data Flow Testing

• Identifies paths in the program that go
  – from the assignment of a value to a variable to
  – the use of such variable,
  to make sure that the variable is properly used.

  \[
  X := 14; \quad \ldots \quad Y := X - 3;
  \]
Data Flow Testing – Motivation

• Middle ground in structural testing
  – Node (=statement) and edge (=branch) coverage don’t test interactions between statements
  – All-path testing is infeasible
  – Need a coverage criterion that is stronger than branch testing but feasible

• Intuition: Statements interact through data flow
  – Value computed in one statement, used in another
  – Bad value computation revealed only when it is used
Data Flow Testing – Definitions

- **Def** – assigned or changed
- **Uses** – utilized (not changed)
  - **C-use** (Computation) e.g. right-hand side of an assignment, an index of an array, parameter of a function.
  - **P-use** (Predicate) branching the execution flow, e.g. in an if statement, while statement, for statement.

```c
[0] bool AccClient(int age;
   gtype gender)
[1] bool accept = false
[2] if (gender==female & age<85)
  [3] accept = true;
[4] if (gender==male & age<80)
  [5] accept = true;
[6] return accept
```

![Diagram](image)

- Def(0/1) = \{age, gender, accept\}
- P-use(2) = \{age, gender\}
- Def(3) = \{accept\}
- P-use(4) = \{age, gender\}
- Def(5) = \{accept\}
- C-use(6) = \{accept\}
Data Flow Testing – Criteria

• **All def-use paths**
  – requires that each simple (i.e., traversing a loop at most once) definition-clear path from a definition of a variable to its use is executed

• **All uses paths**
  – requires that for each definition-use pair of a variable at least one simple definition-clear path is executed

• **All definitions paths**
  – requires that at least one path from the definition of a variable to one of its uses is executed

• ...
Data Flow Testing – Example

Considering age, there are two DU paths:

(a) [0]-[2]
(b) [0]-[4]

Test cases:
AccClient(*, *)-> *

```
[0] bool AccClient(int age;
               gtype gender)
[1] bool accept = false
[2] if (gender==female & age<85)
    [3]   accept = true;
[4] if (gender==male & age<80)
    [5]   accept = true;
[6] return accept
```
Data Flow Testing – Example

Considering gender, there are two DU paths:

(a) [0]-[2]
(b) [0]-[4]

Test cases:
AccClient(*, *) -> *

```
[0] bool AccClient(int age;
    gtype gender)
[1] bool accept = false
[2] if (gender==female & age<85)
    accept = true;
[3] if (gender==male & age<80)
    accept = true;
[5] return accept
```

Test cases needed:
AccClient() is executed
Data Flow Testing – Example

Considering `accept`, there are three DU paths:
(a) [1]-[6] (b) [3]-[6] (c) [3]-[6]

Test cases:
(a) `AccClient(\*, 85)` -> false
(b) `AccClient(f, 80)` -> true
(c) `AccClient(m, 80)` -> true

```c
[0] bool AccClient(int age; 
gtype gender)
[1] bool accept = false
[2] if (gender==female & age<85)
[3]   accept = true;
[4] if (gender==male & age<80)
[5]   accept = true;
[6] return accept
```
Data Flow Criteria

All c-uses  All def-use paths  All p-uses

All c-uses, some p-uses  All p-uses, some c-uses

All uses

All def-use paths

Weaker

# tests

Stronger

All uses

All def-use paths

All p-uses

All def-use paths

All c-uses, some p-uses

All c-uses
Data Flow Criteria

Weaker

All c-uses
All uses
All c-uses, some p-uses
All c-uses

Stronger

All def-use paths
All paths
All def-use paths
All branches

# tests

All p-uses
All p-uses, some c-uses
All p-uses

All uses
All uses
All uses
Effectiveness of Control-Flow & Data-Flow Test Criteria

TABLE I. SYSTEMS UNDER TEST

<table>
<thead>
<tr>
<th>Program</th>
<th>KLOC</th>
<th>Test KLOC</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart</td>
<td>96</td>
<td>50</td>
<td>2,205</td>
</tr>
<tr>
<td>Closure</td>
<td>90</td>
<td>83</td>
<td>7,927</td>
</tr>
<tr>
<td>Math</td>
<td>85</td>
<td>19</td>
<td>3,602</td>
</tr>
<tr>
<td>Time</td>
<td>28</td>
<td>53</td>
<td>4,130</td>
</tr>
<tr>
<td>Lang</td>
<td>22</td>
<td>6</td>
<td>2,245</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>321</td>
<td>211</td>
<td>20,109</td>
</tr>
</tbody>
</table>

Source:
Effectiveness of Control-Flow & Data-Flow Test Criteria

<table>
<thead>
<tr>
<th></th>
<th>% of detected faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>32%  5%  10%  0%  14%  10%</td>
</tr>
<tr>
<td>Branch</td>
<td>32% 18% 18% 11% 18% 19%</td>
</tr>
<tr>
<td>MC/DC</td>
<td>24% 18% 18% 11% 25% 19%</td>
</tr>
<tr>
<td>Loop</td>
<td>12% 5% 18% 0%  8%  8%</td>
</tr>
<tr>
<td>All Control-flow</td>
<td>44% 24% 33% 15% 29% 28%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th># of undetected faults by control flow criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14    84    41   23    36   198</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>% of detected faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>def-use (DUA)</td>
<td>86%  87%  80%  91%  50%  79%</td>
</tr>
<tr>
<td>Data &amp; control-flow</td>
<td>92%  90%  87%  92%  65%  85%</td>
</tr>
</tbody>
</table>

Source:
Recommended Textbook Exercises

• Chapter 5
  – 2,
  – 5, 6,
  – 9, 10, 11,
  – 14
Structure of Lecture 3

• Introduction to White-Box Testing
• Control-Flow Testing
• Loop Testing
• Data-Flow Testing
• Lab 2
Lab 2: Part 2 – White-Box Testing

Lab 2 (week 22: Feb 28 – Mar 01) - White-Box Testing (5%)

Lab 3 Instructions
Lab 3 Sample Code

Submission Deadlines:
• Tuesday Labs: Monday, 06 Mar, 23:59
• Wednesday Labs: Tuesday, 07 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

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

Code

Coverage Criteria:
- Statement (Instruction)
- Branch (Decision)
Tool: Eclipse Plugin

Test Report 1 &
Test Coverage 1a + 1b

Test Report 2 &
Test Coverage 2a + 2b

Code
Next Week

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

• Lecture 4:
  – Mutation Testing
  – Static Testing (Inspection/Review)
  – Defect Estimation

• In addition to do:
  – Read textbook chapter 5 (available via OIS)