The following are modified versions of the publicly-available slides for Chapter 7 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
Overview

• How to apply graph-based criteria to source code?

• **Graph**: Usually the control flow graph (CFG)

• **Node coverage**: Execute every statement

• **Edge coverage**: Execute every branch

• **Loops**: structures such as `for` loops, `while` loops, etc.

• **Data flow coverage**: Augment the CFG
  – `defs` are statements that assign values to variables
  – `uses` are statements that use variables
Control Flow Graphs

- **CFG** captures control structures in method executions

- **Nodes**: Statements or statement sequences (basic blocks)

- **Edges**: Transfers of control

- **Basic Block**: A sequence of statements such that if the first statement is executed, all statements will be (no branches)

- CFGs are sometimes annotated with extra information
  - branch predicates, defs, uses

- Rules for translating statements into graphs …
CFG: The if Statement

```c
if (x < y)
{
    y = 0;
    x = x + 1;
}
else
{
    x = y;
}
```

```
if (x < y)
{
    y = 0;
    x = x + 1;
}
else
{
    x = y;
}
```
CFG : The if-Return Statement

```java
if (x < y)
{
    return;
}
print (x);
return;
```

No edge from node 2 to 3.
The return nodes must be distinct.

© Ammann & Offutt
Loops

• Loops require “extra” nodes to be added

• Nodes that do not represent statements or basic blocks
CFG : while and for Loops

```
x = 0;
while (x < y)
{
   y = f(x, y);
   x = x + 1;
}
return (x);
```

```
for (x = 0; x < y; x++)
{
   y = f(x, y);
}
return (x);
```
x = 0;
do
{    y = f(x, y);
   x = x + 1;
} while (x < y);
return (y);

\[\begin{align*}
\text{x} &= 0; \\
do & \{ \\
\text{y} &= f(x, y); \\
\text{x} &= x + 1; \\
\} \text{ while (x < y); return (y);} \\
\end{align*}\]
Read (c);
switch (c) {
    case 'N':
        z = 25;
        break;
    case 'Y':
        x = 50;
        break;
    default:
        x = 0;
        break;
}
print (x);

Cases without breaks fall through to the next case
try
{
    s = br.readLine();
    if (s.length() > 96)
        throw new Exception ("too long");
    if (s.length() == 0)
        throw new Exception ("too short");
} (catch IOException e) {
    e.printStackTrace();
} (catch Exception e) {
    e.getMessage();
}
return (s);
public static void computeStats(int[] numbers) {
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;
    sum = 0;
    for (int i = 0; i < length; i++)
    {
        sum += numbers[i];
    }
    med = numbers[length / 2];
    mean = sum / (double) length;
    varsum = 0;
    for (int i = 0; i < length; i++)
    {
        varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
    }
    var = varsum / (length - 1.0);
    sd = Math.sqrt(var);
    System.out.println("length: " + length);
    System.out.println("mean: " + mean);
    System.out.println("median: " + med);
    System.out.println("variance: " + var);
    System.out.println("standard deviation: " + sd);
}
public static void computeStats(int[] numbers) {
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;
    sum = 0;
    for (int i = 0; i < length; i++) {
        sum += numbers[i];
    }
    med = numbers[length / 2];
    mean = sum / (double) length;
    varsum = 0;
    for (int i = 0; i < length; i++) {
        varsum = varsum + ((numbers[i] - mean) * (numbers[i] - mean));
    }
    var = varsum / (length - 1.0);
    sd = Math.sqrt(var);
    System.out.println("length: " + length);
    System.out.println("mean: " + mean);
    System.out.println("median: " + med);
    System.out.println("variance: " + var);
    System.out.println("standard deviation: " + sd);
}
Control Flow TRs and Test Paths—EC

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. [1, 2]</td>
<td>[1, 2, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>B. [2, 3]</td>
<td></td>
</tr>
<tr>
<td>C. [3, 4]</td>
<td></td>
</tr>
<tr>
<td>D. [3, 5]</td>
<td></td>
</tr>
<tr>
<td>E. [4, 3]</td>
<td></td>
</tr>
<tr>
<td>F. [5, 6]</td>
<td></td>
</tr>
<tr>
<td>G. [6, 7]</td>
<td></td>
</tr>
<tr>
<td>H. [6, 8]</td>
<td></td>
</tr>
<tr>
<td>I. [7, 6]</td>
<td></td>
</tr>
</tbody>
</table>
Control Flow TRs and Test Paths—EPC

Edge-Pair Coverage

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. [1, 2, 3]</td>
<td>i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>B. [2, 3, 4]</td>
<td>ii. [1, 2, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>C. [2, 3, 5]</td>
<td>iii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>D. [3, 4, 3]</td>
<td></td>
</tr>
<tr>
<td>E. [3, 5, 6]</td>
<td></td>
</tr>
<tr>
<td>F. [4, 3, 5]</td>
<td></td>
</tr>
<tr>
<td>G. [5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>H. [5, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>I. [6, 7, 6]</td>
<td></td>
</tr>
<tr>
<td>J. [7, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>K. [4, 3, 4]</td>
<td></td>
</tr>
<tr>
<td>L. [7, 6, 7]</td>
<td></td>
</tr>
</tbody>
</table>

Test Paths

<table>
<thead>
<tr>
<th>TP</th>
<th>TRs toured</th>
<th>sidetrips</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>A, B, D, E, F, G, I, J</td>
<td>C, H</td>
</tr>
<tr>
<td>ii</td>
<td>A, C, E, H</td>
<td></td>
</tr>
<tr>
<td>iii</td>
<td>A, B, D, E, F, G, I, J, K, L</td>
<td>C, H</td>
</tr>
</tbody>
</table>

TP iii makes TP i redundant. A minimal set of TPs is cheaper.
### Control Flow TRs and Test Paths—PPC

#### Prime Path Coverage

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. [3, 4, 3]</td>
<td>i. [1, 2, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>B. [4, 3, 4]</td>
<td>ii. [1, 2, 3, 4, 3, 4, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>C. [7, 6, 7]</td>
<td>iii. [1, 2, 3, 4, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>D. [7, 6, 8]</td>
<td>iv. [1, 2, 3, 5, 6, 7, 6, 8]</td>
</tr>
<tr>
<td>E. [6, 7, 6]</td>
<td>v. [1, 2, 3, 5, 6, 8]</td>
</tr>
<tr>
<td>F. [1, 2, 3, 4]</td>
<td></td>
</tr>
<tr>
<td>G. [4, 3, 5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>H. [4, 3, 5, 6, 8]</td>
<td></td>
</tr>
<tr>
<td>I. [1, 2, 3, 5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>J. [1, 2, 3, 5, 6, 8]</td>
<td></td>
</tr>
</tbody>
</table>

TP ii makes TP i redundant.

### Test Paths

<table>
<thead>
<tr>
<th>TP</th>
<th>TRs toured</th>
<th>sidetrips</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>A, D, E, F, G</td>
<td>H, I, J</td>
</tr>
<tr>
<td>ii</td>
<td>A, B, C, D, E, F, G</td>
<td>H, I, J</td>
</tr>
<tr>
<td>iii</td>
<td>A, F, H</td>
<td>J</td>
</tr>
<tr>
<td>iv</td>
<td>D, E, F, I</td>
<td>J</td>
</tr>
<tr>
<td>v</td>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>
Data Flow Coverage for Source

- **def**: a location where a value is stored into memory
  - \( x \) appears on the **left side** of an assignment (e.g., \( x = 44; \))
  - \( x \) is an **actual parameter** in a call site & method changes \( x \)’s value
  - \( x \) is a method’s **formal parameter** (implicit def on method start)
  - \( x \) is an **input** to a program

- **use**: a location where variable’s value is **accessed**
  - \( x \) appears on the **right side** of an assignment (e.g., \( y = \sqrt{x}; \))
  - \( x \) appears in a conditional **test**
  - \( x \) is an **actual parameter** to a method
  - \( x \) is an **output** of the program
  - \( x \) is an output of a method in a **return** statement

- A def and a use on the same node is only a DU-pair if the def occurs after the use and the node is in a loop
public static void computeStats (int [] numbers) {
    int length = numbers.length;
    double med, var, sd, mean, sum, varsum;

    sum = 0.0;
    for (int i = 0; i < length; i++) {
        sum += numbers [ i ];
    }
    med = numbers [ length / 2 ];
    mean = sum / (double) length;

    varsum = 0.0;
    for (int i = 0; i < length; i++) {
        varsum = varsum + ((numbers [ i ] - mean) * (numbers [ i ] - mean));
    }
    var = varsum / ( length - 1 );
    sd = Math.sqrt ( var );

    System.out.println ("length: " + length);
    System.out.println ("mean: " + mean);
    System.out.println ("median: " + med);
    System.out.println ("variance: " + var);
    System.out.println ("standard deviation: " + sd);
}
Control Flow Graph for Stats

1. \( \text{numbers} \) = sum = 0, length = numbers.length

2. \( i = 0 \)

3. \( i \geq length \)

4. \( i < length \)

5. \( i \geq length \)

6. \( i < length \)

7. \( \text{varsum} = \ldots \)

8. \( \text{var} = \text{varsum} / (\text{length} - 1.0) \)

\( \text{sd} = \text{Math.sqrt} (\text{var}) \)

\( \text{print} (\text{length, mean, med, var, sd}) \)

Annotate with the statements ...
CFG for Stats – With Defs & Uses

def (1) = { numbers, sum, length }
use (1) = { numbers}

def (2) = { i }

use (3, 5) = { i, length }
use (3, 4) = { i, length }

def (4) = { sum, i }
use (4) = { sum, numbers, i }

use (3, 4) = { i, length }
use (3, 5) = { i, length }
def (5) = { med, mean, varsum, i }
use (5) = { numbers, length, sum }
def (8) = { var, sd }
use (8) = { varsum, length, mean, med, var, sd }

use (6, 8) = { i, length }
use (6, 7) = { i, length }
def (7) = { varsum, i }
use (7) = { varsum, numbers, i, mean }

Turn the annotations into def and use sets …
# Defs and Uses Tables for Stats

<table>
<thead>
<tr>
<th>Node</th>
<th>Def</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>{ numbers, sum, length }</code></td>
<td><code>{ numbers }</code></td>
</tr>
<tr>
<td>2</td>
<td><code>{ i }</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>{ }</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>{ sum, i }</code></td>
<td><code>{ numbers, i, sum }</code></td>
</tr>
<tr>
<td>5</td>
<td><code>{ med, mean, varsum, i }</code></td>
<td><code>{ numbers, length, sum }</code></td>
</tr>
<tr>
<td>6</td>
<td><code>{ varsum, i }</code></td>
<td><code>{ varsum, numbers, i, mean }</code></td>
</tr>
<tr>
<td>7</td>
<td><code>{ varsum, i }</code></td>
<td><code>{ varsum, numbers, i, mean, var, sd }</code></td>
</tr>
<tr>
<td>8</td>
<td><code>{ var, sd }</code></td>
<td><code>{ varsum, length, var, mean, med, var, sd }</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2)</td>
<td></td>
</tr>
<tr>
<td>(2, 3)</td>
<td></td>
</tr>
<tr>
<td>(3, 4)</td>
<td><code>{ i, length }</code></td>
</tr>
<tr>
<td>(4, 3)</td>
<td></td>
</tr>
<tr>
<td>(3, 5)</td>
<td><code>{ i, length }</code></td>
</tr>
<tr>
<td>(5, 6)</td>
<td></td>
</tr>
<tr>
<td>(6, 7)</td>
<td><code>{ i, length }</code></td>
</tr>
<tr>
<td>(7, 6)</td>
<td></td>
</tr>
<tr>
<td>(6, 8)</td>
<td><code>{ i, length }</code></td>
</tr>
</tbody>
</table>
Recall: DU Pairs and DU Paths

- **def (n) or def (e)**: The set of variables that are defined by node n or edge e
- **use (n) or use (e)**: The set of variables that are used by node n or edge e

- **DU pair**: A pair of locations \((l_i, l_j)\) s.t. a variable \(v\) is defined at \(l_i\) and used at \(l_j\)

- **Def-clear**: Path from \(l_i\) to \(l_j\) is def-clear w.r.t. \(v\) if \(v\) is not given another value on any of the nodes or edges in the path

- **Reach**: If there is a def-clear path from \(l_i\) to \(l_j\) with respect to \(v\), the def of \(v\) at \(l_i\) reaches the use at \(l_j\)

- **du-path**: A simple subpath that is def-clear w.r.t. \(v\) from a def of \(v\) to a use of \(v\)
  - **Def-path set, du \((n_i, v)\)** – the set of du-paths that start at \(n_i\)
  - **Def-pair set, du \((n_i, n_j, v)\)** – the set of du-paths from \(n_i\) to \(n_j\)
## DU Pairs for Stats

<table>
<thead>
<tr>
<th>variable</th>
<th>DU Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>(1, 4)  (1, 5) (1, 7)</td>
</tr>
<tr>
<td>length</td>
<td>(1, 5)  (1, 8) (1, (3,4)) (1, (3,5)) (1, (6,7)) (1, (6,8))</td>
</tr>
<tr>
<td>med</td>
<td>(5, 8)</td>
</tr>
<tr>
<td>var</td>
<td>(8, 8)</td>
</tr>
<tr>
<td>sd</td>
<td>(8, 8)</td>
</tr>
<tr>
<td>mean</td>
<td>(5, 7)  (5, 8)</td>
</tr>
<tr>
<td>sum</td>
<td>(1, 4)  (1, 5) (4, 4) (4, 5)</td>
</tr>
<tr>
<td>varsum</td>
<td>(5, 7)  (5, 8) (7, 7) (7, 8)</td>
</tr>
<tr>
<td>i</td>
<td>(2, 4)  (2, (3,4)) (2, 5) (2, 7) (2, (6,7)) (2, (6,8))  (4, 4)  (4, (3,4)) (4, (3,5)) (4, 7) (4, (6,7)) (4, (6,8))  (5, 7)  (5, (6,7)) (5, (6,8))  (7, 7)  (7, (6,7)) (7, (6,8))</td>
</tr>
</tbody>
</table>

### Notes:
- **du clear path** ...
- **different scope for i**
- **No def-clear path** ...
- **No path through graph** from nodes 5 and 7 to 4 or 3
- **defs come before uses**, do not count as DU pairs
- **defs after use in loop**, these are valid DU pairs

---

*Introduction to Software Testing, Edition 2 (Ch 7) © Ammann & Offutt*
## DU Paths for Stats

<table>
<thead>
<tr>
<th>Variable</th>
<th>DU Pairs</th>
<th>DU Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>(1, 4)</td>
<td>[ 1, 2, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(1, 5)</td>
<td>[ 1, 2, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(1, 7)</td>
<td>[ 1, 2, 3, 5, 6, 7 ]</td>
</tr>
<tr>
<td>length</td>
<td>(1, 5)</td>
<td>[ 1, 2, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(1, 8)</td>
<td>[ 1, 2, 3, 5, 6, 8 ]</td>
</tr>
<tr>
<td></td>
<td>(1, (3,4))</td>
<td>[ 1, 2, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(1, (3,5))</td>
<td>[ 1, 2, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(1, (6,7))</td>
<td>[ 1, 2, 3, 5, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(1, (6,8))</td>
<td>[ 1, 2, 3, 5, 6, 8 ]</td>
</tr>
<tr>
<td>med</td>
<td>(5, 8)</td>
<td>[ 5, 6, 8 ]</td>
</tr>
<tr>
<td>var</td>
<td>(8, 8)</td>
<td>No path needed</td>
</tr>
<tr>
<td>sd</td>
<td>(8, 8)</td>
<td>No path needed</td>
</tr>
<tr>
<td>sum</td>
<td>(1, 4)</td>
<td>[ 1, 2, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(1, 5)</td>
<td>[ 1, 2, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(4, 4)</td>
<td>[ 4, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(4, 5)</td>
<td>[ 4, 3, 5 ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>DU Pairs</th>
<th>DU Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>(5, 7)</td>
<td>[ 5, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(5, 8)</td>
<td>[ 5, 6, 8 ]</td>
</tr>
<tr>
<td>varsum</td>
<td>(5, 7)</td>
<td>[ 5, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(5, 8)</td>
<td>[ 5, 6, 8 ]</td>
</tr>
<tr>
<td></td>
<td>(7, 7)</td>
<td>[ 7, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(7, 8)</td>
<td>[ 7, 6, 8 ]</td>
</tr>
<tr>
<td>i</td>
<td>(2, 4)</td>
<td>[ 2, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(2, (3,4))</td>
<td>[ 2, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(2, (3,5))</td>
<td>[ 2, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(4, 4)</td>
<td>[ 4, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(4, (3,4))</td>
<td>[ 4, 3, 4 ]</td>
</tr>
<tr>
<td></td>
<td>(4, (3,5))</td>
<td>[ 4, 3, 5 ]</td>
</tr>
<tr>
<td></td>
<td>(5, 7)</td>
<td>[ 5, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(5, (6,7))</td>
<td>[ 5, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(5, (6,8))</td>
<td>[ 5, 6, 8 ]</td>
</tr>
<tr>
<td></td>
<td>(7, 7)</td>
<td>[ 7, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(7, (6,7))</td>
<td>[ 7, 6, 7 ]</td>
</tr>
<tr>
<td></td>
<td>(7, (6,8))</td>
<td>[ 7, 6, 8 ]</td>
</tr>
</tbody>
</table>
DU Paths for Stats—No Duplicates

There are 38 DU paths for Stats, but only 12 unique

- [1, 2, 3, 4]
- [1, 2, 3, 5]
- [1, 2, 3, 5, 6, 7]
- [1, 2, 3, 5, 6, 8]
- [2, 3, 4]
- [2, 3, 5]
- [4, 3, 4]
- [4, 3, 5]
- [5, 6, 7]
- [5, 6, 8]
- [7, 6, 7]
- [7, 6, 8]

- **4** expect a loop not to be “entered”
- **6** require at least one iteration of a loop
- **2** require at least two iterations of a loop
### Test Inputs and Test Paths

**Test input:** `numbers = [44] ; length = 1
Test Path : [ 1, 2, 3, 4, 3, 5, 6, 7, 6, 8 ]`

Additional DU Paths covered (no sidetrips)

- [ 1, 2, 3, 4 ]
- [ 2, 3, 4 ]
- [ 4, 3, 5 ]
- [ 5, 6, 7 ]
- [ 7, 6, 8 ]

The five stars ✭ that require at least one iteration of a loop

**Test Input :** `numbers = [2, 10, 15] ; length = 3
Test Path : [ 1, 2, 3, 4, 3, 4, 3, 4, 3, 5, 6, 7, 6, 7, 6, 7, 6, 8 ]`

DU Paths covered (no sidetrips)

- [ 4, 3, 4 ]
- [ 7, 6, 7 ]

The two stars ☀ that require at least two iterations of a loop

Other DU paths ★ require arrays with length 0 to skip loops

But the method fails with index out of bounds exception…

```
med = numbers [length / 2];
```

A fault was found
Summary

• Applying the graph test criteria to control flow graphs is relatively straightforward
  – Most of the developmental research work was done with CFGs

• A few subtle decisions must be made to translate control structures into the graph

• Some tools will assign each statement to a unique node
  – These slides and the book uses basic blocks
  – Coverage is the same, although the bookkeeping will differ
Next

• Logic coverage

• Some announcements
  – Sprint 0.2 was due at 9:30am today
  – Talk to me if you are in a distant time zone
  – HW2 has been released on CMS, due 2/29 at 9:30am
  – HW2 is to be done individually, no discussion on Ed