ICS 52: Introduction to Software Engineering

Fall Quarter 2004

Professor Richard N. Taylor

Lecture Notes: Testing

http://www.ics.uci.edu/~taylor/ICS_52_FQ04/syllabus.html
Two Approaches

- **White box testing**
  - **Structural testing**
  - Test cases designed, selected, and ran based on structure of the source code
  - Scale: tests the nitty-gritty
  - Drawbacks: need access to source; thought-process driven by what you’ve got

- **Black box testing**
  - **Specification-based testing**
  - Test cases designed, selected, and ran based on specifications
  - Scale: tests the overall system behavior
  - Drawback: less thorough? Thought-process limited to what you’re supposed to have.
Structural Testing

- Use source code to derive test cases
  - Build a graph model of the system
    » Control flow
    » Data flow
  - State test cases in terms of graph coverage
- Choose test cases that guarantee different types of coverage
  - Node coverage
  - Edge coverage
  - Loop coverage
  - Condition coverage
  - Path coverage
Example

1 Node getSecondElement() {
2     Node head = getHead();
3     if (head == null)
4         return null;
5     if (head.next == null)
6         return null;
7     return head.next.node;
8 }
Example

```java
1 float homeworkAverage(float[] scores) {
  2    float min = 99999;
  3    float total = 0;
  4    for (int i = 0 ; i < scores.length ; i++) {
        if (scores[i] < min) {
            5            min = scores[i];
            6            total += scores[i];
        }
    }
  7    total = total – min;
  8    return total / (scores.length – 1);
  9 }
```
Node Coverage

- Select test cases such that every node in the graph is visited
  - Also called statement coverage
    » Guarantees that every statement in the source code is executed at least once
- Selects minimal number of test cases

Test case: \{ 2 \}
Edge Coverage

- Select test cases such that every edge in the graph is visited
  - Also called branch coverage
    » Guarantees that every branch in the source code is executed at least once
- More thorough than node coverage
  - More likely to reveal logical errors

Test case: \{ 1, 2 \}
Other Coverage Criteria

- Loop coverage
  - Select test cases such that every loop *boundary* and *interior* is tested
    » Boundary: 0 iterations
    » Interior: 1 iteration and > 1 iterations
  - Watch out for nested loops
  - Less precise than edge coverage

- Condition coverage
  - Select test cases such that all conditions are tested
    » if (a > b || c > d) …
  - More precise than edge coverage
Other Coverage Criteria

- Path coverage
  - Select test cases such that every path in the graph is visited
  - Loops are a problem
    » 0, 1, average, max iterations
- Most thorough...
- ...but is it feasible?
Challenges

- Structural testing can cover all nodes or edges without revealing obvious faults
  - No matter what input, program always returns 0
- Some nodes, edges, or loop combinations may be infeasible
  - Unreachable/unexecutable code
- “Thoroughness”
  - A test suite that guarantees edge coverage also guarantees node coverage
  - …but it may not find as many faults as a different test suite that only guarantees node coverage
More Challenges

- Interactive programs
- Listeners or event-driven programs
- Concurrent programs
- Exceptions
- Self-modifying programs
- Mobile code
- Constructors/destructors
- Garbage collection
Specification-Based Testing

- Use specifications to derive test cases
  - Requirements
  - Design
  - Function signature
- Based on some kind of input domain
- Choose test cases that guarantee a wide range of coverage
  - Typical values
  - Boundary values
  - Special cases
  - Invalid input values
“Some Kind of Input Domain”

- Determine a basis for dividing the input domain into subdomains
  - Subdomains may overlap
    - Possible bases
      - Size
      - Order
      - Structure
      - Correctness
      - Your creative thinking
- Select test cases from each subdomain
  - One test case may suffice
Example

```java
1 float homeworkAverage(float[] scores) {
2     float min = 99999;
3     float total = 0;
4     for (int i = 0 ; i < scores.length ; i++) {
5         if (scores[i] < min) {
6             min = scores[i];
7             total += scores[i];
8         }
9     }
10     total = total - min;
11     return total / (scores.length - 1);
12 }
```
Possible Bases

- Array length
  - Empty array
  - One element
  - Two or three elements
  - Lots of elements

Input domain: float[]
Basis: array length

small
one
large
empty
Possible Bases

- Position of minimum score
  - Smallest element first
  - Smallest element in middle
  - Smallest element last

Input domain: float[]
Basis: position of minima

somewhere in middle
first
last
Possible Bases

- Number of minima
  - Unique minimum
  - A few minima
  - All minima

Input domain: float[]
Basis: number of minima

1 minimum  all data equal  2 minima
Testing Matrix

<table>
<thead>
<tr>
<th>Test case (input)</th>
<th>Basis (subdomain)</th>
<th>Expected output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

University of California, Irvine
### homeworkAverage 1

<table>
<thead>
<tr>
<th>Test case (input)</th>
<th>Basis: Array length</th>
<th>Expected output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>x</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>(87.3)</td>
<td>x</td>
<td>87.3</td>
<td>crashes!</td>
</tr>
<tr>
<td>(90,95,85)</td>
<td>x</td>
<td>92.5</td>
<td></td>
</tr>
<tr>
<td>(80,81,82,83,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84,85,86,87,</td>
<td>x</td>
<td>86.0</td>
<td></td>
</tr>
<tr>
<td>88,89,90,91)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# homeworkAverage 2

<table>
<thead>
<tr>
<th>Test case (input)</th>
<th>Basis: Position of minimum</th>
<th>Expected output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(80,87,88,89)</td>
<td>x</td>
<td>88.0</td>
<td></td>
</tr>
<tr>
<td>(87,88,80,89)</td>
<td>x</td>
<td>88.0</td>
<td></td>
</tr>
<tr>
<td>(99,98,0,97,96)</td>
<td>x</td>
<td>97.5</td>
<td></td>
</tr>
<tr>
<td>(87,88,89,80)</td>
<td>x</td>
<td>88.0</td>
<td></td>
</tr>
</tbody>
</table>
### Test case (input) vs. Basis: Number of minima

<table>
<thead>
<tr>
<th>Test case (input)</th>
<th>Basis: Number of minima</th>
<th>Expected output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(80,87,88,89)</td>
<td>x</td>
<td>88.0</td>
<td></td>
</tr>
<tr>
<td>(87,86,86,88)</td>
<td>x</td>
<td>87.0</td>
<td></td>
</tr>
<tr>
<td>(99,98,0,97,0)</td>
<td>x</td>
<td>73.5</td>
<td></td>
</tr>
<tr>
<td>(88,88,88,88)</td>
<td>x</td>
<td>88.0</td>
<td></td>
</tr>
</tbody>
</table>

University of California, Irvine