Testing Methods: White Box Testing II

Outline

• Today we continue our look at white box testing with more code coverage methods, and a data coverage method

• We’ll look at :

  - code coverage testing
    - decision coverage
    - condition coverage
    - branch coverage
    - loop coverage
    - path coverage

  - data coverage testing
    - data flow coverage
Decision Coverage

Decision (Branch) Coverage Method

• Causes every decision (if, switch, while, etc.) in the program to be made both ways (or every possible way for switch)

• **System**: Design a test case to exercise each decision in the program each way (true / false)

• **Completion criterion**: A test case for each side of each decision
  - Can be checked by *instrumentation injection* to track branches taken in execution
Example: Decision Coverage

```
// calculate numbers less than x
// which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();

1 if (y == 0)
c.println("y is zero");
else
2 if (x == 0)
c.println("x is zero");
else
{
    for (int i = 1; i <= x; i++)
    {
3        if (i % y == 0)
c.println(i);
    }
}
```
Example: Decision Coverage

Decision Coverage Tests

• We make one test for each side of each decision

<table>
<thead>
<tr>
<th>Decision</th>
<th>x input</th>
<th>y input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 true</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 false</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2 true</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2 false</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 true</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 false</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Condition Coverage

Condition Coverage Method

• Like decision coverage, but causes every condition expression to be exercised both ways (true / false)

• A condition is any true / false subexpression in a decision

Example:

```java
if (( x == 1 || y > 2 ) && z < 3 )
```

Requires separate condition coverage tests for each of:

- `x == 1` true / false
- `y > 2` true / false
- `z < 3` true / false

• More effective than simple decision coverage since exercises the different entry preconditions for each branch selected
Loop Coverage

Loop Coverage Method

• Most programs do their real work in do, while and for loops

• This method makes tests to exercise each loop in the program in four different states:
  - execute body zero times (do not enter loop)
  - execute body once (i.e., do not repeat)
  - execute body twice (i.e., repeat once)
  - execute body many times

• Usually used as an enhancement of a statement, block, decision or condition coverage method

• System: Devise test cases to exercise each loop with zero, one, two and many repetitions

• Completion criterion: A test for each of these cases for each loop
  - Can be verified using instrumentation injection in the code
Example: Loop Coverage

```
// calculate numbers less than x
// which are divisible by y

int x, y;
x = c.readInt();
y = c.readInt();

if (y == 0)
    c.println("y is zero");
else if (x == 0)
    c.println("x is zero");
else
{
    for (int i=1; i<=x ; i++)
    {
        if (i % y == 0)
            c.println(i);
    }
}
```

Loop Body

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Execution Paths

An **execution path** is a sequence of executed statements starting at the **entry** to the unit (usually the first statement) and ending at the **exit** from the unit (usually the last statement).

Two paths are **independent** if there is at least one statement on one path which is not executed on the other.

**Path analysis** (also known as **cyclomatic complexity** analysis) identifies all the **independent paths** through a unit.

* - a **code metric** we will look at later in the course
Execution Path Analysis

Flow Graphs

- It is easiest to do path analysis if we look at the execution flow graph of the program or unit.
- The flow graph simply shows program control flow between basic blocks.

[Diagram showing flow graphs for if-then-else, do-while, and switch structures]
Path Coverage Testing

**Advantages**

- Covers all **basic blocks** (does all of basic block testing)
- Covers all **conditions** (does all of decision/condition testing)
- Does all of both, but with **fewer tests**!
- Automatable (actually, in practice requires automation)

**Disadvantages**

- Does not take **data** complexity into account at all
Path Coverage Testing

Disadvantages

• **Example**: These fragments should be tested the same way, since they actually implement the same solution - but the one on the left gets five tests, whereas the one on the right gets only one

```java
// control-centric solution
switch (n) {
    case 1:
        s = "One";
        break;
    case 2:
        s = "Two";
        break;
    case 3:
        s = "Three";
        break;
    case 4:
        s = "Four";
        break;
    case 5:
        s = "Five";
        break;
}

// data-centric solution
String numbers[] =
    {"One", "Two",
     "Three", "Four", "Five"};
s = numbers[n];
```
White Box Data Coverage

Data Coverage Methods

- Data coverage methods explicitly try to cover the data aspects of the program code, rather than the control aspects.

- In this course we will cover data flow coverage including several different data flow coverage test criteria.

(Won’t do these in detail, just overview)
White Box Data Coverage

Data Flow Coverage

• Data flow coverage is concerned with variable definitions and uses along execution paths

• A variable is defined if it is assigned a new value during a statement execution
  • A variable definition in one statement is alive in another if there is a path between the two statements that does not redefine the variable

• There are two types of variable uses
  • A P-use of a variable is a predicate use (e.g. if statement)
  • A C-use of a variable is a computation use or any other use (e.g. I/O statements)
Example: Definition, P-Use, C-Use of Variables

```java
static int find (int list[], int n, int key)
{
    // binary search of ordered list
    int lo = 0;
    int hi = n - 1;
    int result = -1;
    <= Definition of result

    while (hi >= lo)
    {
        if (result != -1)  <= P-Use of result
            break;
        else
        {
            final int mid = (lo + hi) / 2;
            if (list[mid] == key)
                result = mid;  <= Definition of result
            else if (list[mid] > key)
                hi = mid - 1;
            else  // list[mid] < key
                lo = mid + 1;
        }
    }

    return result;
}
```
Example: Definition, P-Use, C-Use of Variables

```c
static int find (int list[], int n, int key)
{
    // binary search of ordered list
    int lo = 0;
    int hi = n - 1;  <!-- Definition of hi -->
    int result = -1;
    while (hi >= lo)  <!-- P-Use of hi -->
    {
        if (result != -1)
            break;
        else
        {
            final int mid = (lo + hi) / 2;  <!-- C-Use of hi -->
            if (list[mid] == key)
                result = mid;
            else if (list[mid] > key)
                hi = mid - 1;  <!-- Definition of hi -->
            else  // list[mid] < key
                lo = mid + 1;
        }
    }
    return result;
}
```
White Box Data Coverage

Data Flow Coverage

• There are a variety of different testing strategies related to data flow:
  
  • All-Uses coverage: test all uses of each definition
  
  • All-Defs coverage: test each definition at least once
  
  • All C-Uses/Some P-Uses coverage: test all computation uses. If no computation uses for a given definition then test at least one predicate use
  
  • All P-Uses/Some C-Uses coverage: test all predicate uses. If no predicate uses for a given definition then test at least one computation use
  
  • All P-Uses coverage: Test each predicate use
White Box Data Coverage

Data Flow Coverage

• We have covered definitions of data, uses of data, and testing strategies for data flow coverage.

• **System:** Identify definitions (and uses) of variables and testing strategy. Design a set of test cases that cover the testing strategy.

• **Completion criterion:** Depends on the test strategy. For example, in All-Defs we are done when we have a test case for each variable definition.
Summary

Testing Methods: White Box Testing II

• Code coverage methods:
  • Decision analysis methods (decision, condition, loop coverage, path coverage)

• Data coverage methods:
  • data flow coverage

Next Time

• Mutation testing