SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

34 Black Box Testing (Ch. 6) DRAFT

Dr. Spencer Smith

Faculty of Engineering, McMaster University

December 15, 2017

34 Black Box Testing (Ch. 6) DRAFT

- Administrative details
- Black Box Testing
  - Formal using PointT
  - Function tables
- Testing boundary conditions
- The oracle problem
- Module testing
- Integration testing
- Testing OO and generic programs
- Testing concurrent and real-time systems

Administrative Details

TBD

Black Box Testing Example

The program receives as input a record describing an invoice. (A detailed description of the format of the record is given.) The invoice must be inserted into a file of invoices that is sorted by date. The invoice must be inserted in the appropriate position: If other invoices exist in the file with the same date, then the invoice should be inserted after the last one. Also, some consistency checks must be performed: The program should verify whether the customer is already in a corresponding file of customers, whether the customer's data in the two files match, etc.

What test cases would satisfy the complete-coverage principle?

Invoice Example Test Cases

1. An invoice whose date is the current date
2. An invoice whose date is before the current date (This might be even forbidden by law) This case, in turn, can be split into the two following subcases:
   2.1 An invoice whose date is the same as that of some existing invoice
   2.2 An invoice whose date does not exist in any previously recorded invoice
3. Several incorrect invoices, checking different types of inconsistencies

Systematic Black-Box Techniques

- Testing driven by logic specifications
- Function table based testing

Test Cases from MIS for PointT

Routine name	In	Out	Exceptions
PointT	real, real	PointT	InvalidPointException
xcoord		real	
ycoord		real	
dist	PointT	real	

$exc := ((\neg(0 \leq x \leq \text{Contants.MAX\_X}) \vee \neg(0 \leq y \leq \text{Constants.MAX\_Y})) \Rightarrow \text{InvalidPointException})$

dist(p):

- output: $out := \sqrt{(self.xc - p.xc)^2 + (self.yc - p.yc)^2}$
- exception: none

What test cases do you recommend?

TestPointT.java I

```java
import org.junit.*;
import static org.junit.Assert.*;
public class TestPointT
{
    private static double
        ADMISS_ERR_CONSTRUCTOR = 0;
    private static double ADMISS_ERR_DIST =
        1e-20;
    @Test
    public void testConstructorForx()
    {
        assertEquals(23, new PointT(23,
            38).xcoord(),
            ADMISS_ERR_CONSTRUCTOR);
```

TestPointT.java II

```java
        }
        @Test
        public void testConstructorFory()
        {
          assertEquals(38, new PointT(23,
            38).ycoord(),
            ADMISS_ERR_CONSTRUCTOR);
        }
        @Test
          (expected=InvalidPointException.class)
        public void testForExceptionNegx()
        {
          PointT p = new PointT(-10, 0);
        }
```

TestPointT.java III

```java
@Test
  (expected=InvalidPointException.class)
public void testForExceptionNegy()
{
  PointT p = new PointT(0, -10);
}
@Test
  (expected=InvalidPointException.class)
public void testForExceptionMaxx()
{
  PointT p = new
    PointT(Constants.MAX_X+1, 0);
}
@Test
  (expected=InvalidPointException.class)
```

TestPointT.java IV

```java
public void testForExceptionMaxy()
{
  PointT p = new PointT(0,
    Constants.MAX_Y+1);
}
@Test
public void testDistNormal()
{
  double x = Constants.MAX_X/2.0;
  double y = Constants.MAX_Y/2.0;
  PointT p = new PointT(x, y);
  assertEquals(Math.sqrt(x*x + y*y),
    p.dist(new PointT(0, 0)),
    ADMISS_ERR_DIST);
}
```

TestPointT.java V

```java
@Test
public void testDistLargestDiagonal()
{
  double x = Constants.MAX_X;
  double y = Constants.MAX_Y;
  PointT p = new PointT(x, y);
  assertEquals(Math.sqrt(x*x + y*y),
     p.dist(new PointT(0, 0)),
     ADMISS_ERR_DIST);
}
@Test
public void testDistAlongEdge()
{
  double x = Constants.MAX_X;
  double y = Constants.MAX_Y;
```

TestPointT.java VI

```java
    PointT p = new PointT(x, y);
    assertEquals(Constants.MAX_X,
        p.dist(new PointT(0,
        Constants.MAX_Y)),
        ADMISS_ERR_DIST);
  }
  @Test
  public void testDistZero()
  {
    double x = Constants.MAX_X/2.0;
    double y = Constants.MAX_Y/2.0;
    PointT p = new PointT(x, y);
    assertEquals(0, p.dist(p),
        ADMISS_ERR_DIST);
  }
```

TestPointT.java VII

```
    // etc .
}
```

Function Table-Based Testing

- Boundaries are obvious in table predicates
- Make test cases that exercise between and on boundaries
- Coverage already aided by function table "rules"

Condition		Result
		f_name
x < 0	y ≤ 5	res 1
	y > 5	res 2
x = 0		res 3
x > 0		res 4

What test cases do you recommend?

Function Table-Based Testing

- Boundaries are obvious in table predicates
- Make test cases that exercise between and on boundaries
- Coverage already aided by function table "rules"

Condition		Result
		f_name
x < 0	y ≤ 5	res 1
	y > 5	res 2
x = 0		res 3
x > 0		res 4

What test cases do you recommend?
What if you use the heuristic [-Large, -Normal, Boundary-1, Boundary, Boundary+1, Normal, Large]?

Function Table-Based Testing

- Boundaries are obvious in table predicates
- Make test cases that exercise between and on boundaries
- Coverage already aided by function table "rules"

Condition		Result f_name
x < 0	y ≤ 5	res 1
	y > 5	res 2
x = 0		res 3
x > 0		res 4

Testing Boundary Conditions

- Testing criteria partition input domain in classes, assuming that behavior is "similar" for all data within a class
- Some typical programming errors, however, just happen to be at the boundary between different classes
  - Off by one errors
  - $<$ instead of $\leq$
  - equals zero

Criterion

- After partitioning the input domain D into several classes, test the program using input values not only "inside" the classes, but also at their boundaries
- This applies to both white-box and black-box techniques
- In practice, use the different testing criteria in combinations

The Oracle Problem

When might it be difficult to know the "expected" output/behaviour?

The Oracle Problem

- Given input test cases that cover the domain, what are the expected outputs?
- Oracles are required at each stage of testing to tell us what the right answer is
- Black-box criteria are better than white-box for building test oracles
- Automated test oracles are required for running large amounts of tests
- Oracles are difficult to design - no universal recipe

The Oracle Problem Continued

- Determining what the right answer should be is not always easy
  - ▶ Air traffic control system
  - ▶ Scientific computing
  - ▶ Machine learning
  - ▶ Artifical intelligence

The Oracle Problem Continued

What are some strategies we can use when we do not have a test oracle?

Strategies Without An Oracle

- Using an independent program to approximate the oracle (pseudo oracle)
- Method of manufactured solutions
- Properties of the expected values can be easier than stating the expected output
  - Examples?

Strategies Without An Oracle

- Using an independent program to approximate the oracle (pseudo oracle)
- Method of manufactured solutions
- Properties of the expected values can be easier than stating the expected output

  - List is sorted
  - Number of entries in file matches number of inputs
  - Conservation of energy or mass
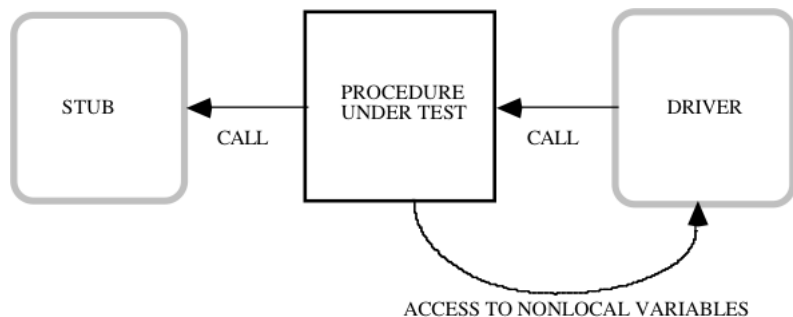 - Expected trends in output are observed (metamorphic testing)
  - etc.

Module Testing

Is it possible to begin testing before all of the modules have been implemented when there is a use relation between modules?

Module Testing

- Scaffolding needed to create the environment in which the module should be tested
- Stubs - a module used by the module under test
- Driver - module activating the module under test
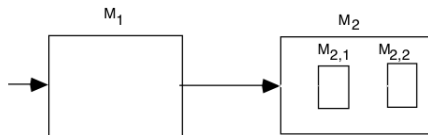
Testing a Functional Module

Integration Testing

- Big-bang approach
  - First test individual modules in isolation
  - Then test integrated system
- Incremental approach
  - Modules are progressively integrated and tested
  - Can proceed both top-down and bottom-up according to the USES relation
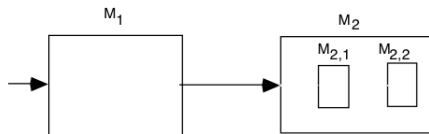
Integration Testing and USES relation

- If integration and test proceed bottom-up only need drivers
- Otherwise if we proceed top-down only stubs are needed

Example

- M_1 USES M_2 and M_2 IS_COMPOSED_OF $\{M_{2,1}, M_{2,2}\}$
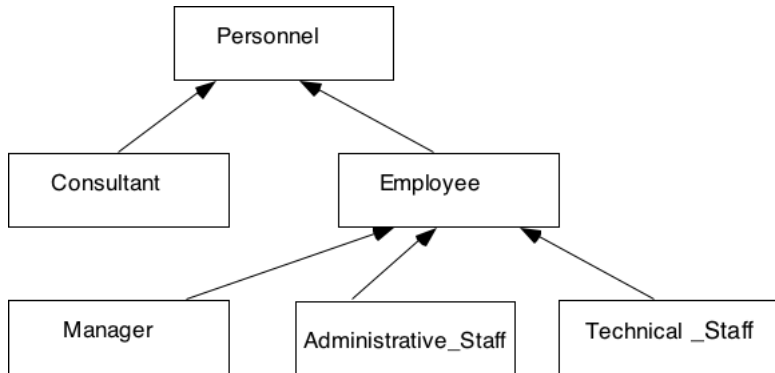- In what order would you test these modules?

Example

- M_1 USES M_2 and M_2 IS_COMPOSED_OF $\{M_{2,1}, M_{2,2}\}$
- Case 1
  - Test M_1 providing a stub for M_2 and a driver for M_1
  - Then provide an implementation for $M_{2,1}$ and a stub for $M_{2,2}$
- Case 2
  - Implement $M_{2,2}$ and test it by using a driver
  - Implement $M_{2,1}$ and test the combination of $M_{2,1}$ and $M_{2,2}$ (i.e. M_2) by using a driver
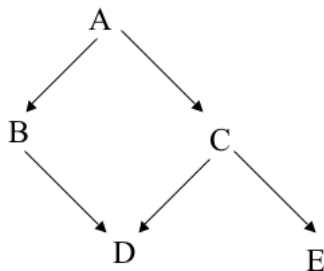 - Finally implement M_1 and test it with M_2 using a driver for M_1

Testing OO and Generic Programs

- New issues
  - Inheritance
  - Genericity
  - Polymorphism
  - Dynamic binding
- Open problems still exist

Inheritance

How to Test Classes of the Hierarchy

- "Flattening" the whole hierarchy and considering every class as totally independent component
- This does not exploit incrementality
- Finding an ad-hoc way to take advantage of the hierarchy
- Think about testing PointT.java and PointMassT.java

A Sample Strategy

- A test that does not have to be repeated for any heir
- A test that must be performed for heir class X and all of its further heirs
- A test that must be redone by applying the same input data, but verifying that the output is not (or is) changed
- A test that must be modified by adding other input parameters and verifying that the the output changes accordingly

Testing Concurrent and Real-time Systems

- Nondeterminism inherent in concurrency affects repeatability
- For real-time systems, a test case consists not only of input data, but also of the times when such data are supplied
- Considerable care and detail when testing real-time systems