

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

**Winter 2017**

# **34 Black Box Testing (Ch. 6) DRAFT**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 19, 2017



# Black Box Testing

- Administrative details
- Black Box Testing
  - ▶ Formal using RegionT
  - ▶ Function tables
- Testing boundary conditions

# Administrative Details

- Today's slide are partially based on slides by Dr. Wassying

# 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 customers 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

<b>Routine name</b>	<b>In</b>	<b>Out</b>	<b>Exceptions</b>
PointT	real, real	PointT	InvalidPointException
xcoord		real	
ycoord		real	
dist	PointT	real	

# TestPointT.java I

```
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

```
}  
@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

```
@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

```
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

@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

```
        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);
    }
```

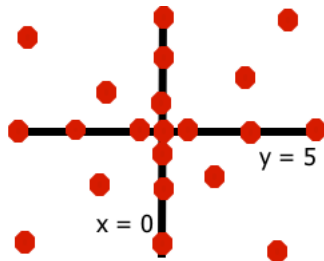
# TestPointT.java VII

}

# 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”

		Result
		<b>f_name</b>
$x < 0$	$y \leq 5$	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

- 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
  - ▶ Parallel testing can approximate an oracle
  - ▶ Properties of the expected values can be easier than stating the expected output

# 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