

# CS608

# Software Testing

Dr. Stephen Brown

Room Eolas 116

[stephen.brown@mu.ie](mailto:stephen.brown@mu.ie)

# Tutorial: Lab 3

- Develop BVA tests for Insurance.premium()
  1. Analysis (from EP)
    1. value lines (natural and specification-based ranges)
    2. partitions
  2. Test Coverage Items
  3. Test Cases
  4. Review
  5. Implement
  6. Run
  7. Results

# CS608

## Boundary Value analysis in More Detail

(Essentials of Software Testing, Chapter 3.4-3.7)

# Fault Model

- The boundary value analysis fault model is where the boundary values of ranges of values are not identified properly
- This leads to incorrect processing near the boundary or edge of the equivalence partition
- These faults tend to be associated with minor typos in the source code, where the wrong comparison operator has been used
- By testing with two values from every equivalence partition – minimum and maximum values – boundary value analysis tests are able to find these kind of faults

# Description

- Programming faults often involve incorrect **boundary conditions**
- Obvious extension to EP: test with the boundary values
- Doubles number of tests, but likely to find these faults
- Each boundary value for each parameter is a test coverage item
- As for EP, number of test cases is minimised by selecting as many uncovered test coverage items as possible in each new test case
- Error test cases are always considered separately: only one error boundary value is included per error test case
- The goal is to achieve 100% coverage of the boundary values

# Summary

- Analysis
  - Identifying Boundary Values
- Test Coverage Items
- Test Cases

# Analysis/Identifying Boundary Values

- Example Boolean isNegative(int x)
- The boundary values for x are as follows:

- Integer.MIN VALUE
- -1
- 0
- Integer.MAX VALUE

Integer.MIN_VALUE	-1	0	Integer.MAX_VALUE

- The boundary values for the return value are the same as the equivalence partitions as it is a boolean type – each equivalence partition is a range with only one data value:

- true
- false

true	false

# Some rules for picking boundary values

- Every parameter has a boundary value at the bottom (min) and top (max) of every equivalence partition
- For a contiguous (or numeric) data type, the successor to the value at the top of one partition must be the value at the bottom of the next
- For `isNegative(int x)`
  - the upper value -1 from the first partition
  - is directly followed by the lower value 0 from the next partition
- The natural range of the parameter provides the ultimate maximum and minimum values



# No Overlap – No Gaps

- Boundary values do not overlap
- There must be no gap between partitions
- A convenient shorthand for specifying partitions and their boundary values is as follows:
  - x: [Integer.MIN VALUE..-1][0..Integer.MAX VALUE]
  - return value: [true][false]

# Test Coverage Items (TCI)

- Each boundary value for each partition for each input and output is a **test coverage item**
- Give each test coverage item a unique identifier
- Use the prefix "BV"
- In equivalence partition testing, the tester has to decide on representative values from each partition
- In boundary value analysis testing, the tester uses the already identified upper and lower values for each partition

# Test Cases

- Input test data is selected, based on test coverage items which are not yet covered
- Each normal test case should include as many additional normal test coverage items as possible
- Each error test case must only include one error test coverage item
- Expected return values are derived from the specification
- However, the tester must ensure that all the test coverage items related to the output parameters are covered
- It may be necessary to “read the specification backwards” to determine input values that will result in a return value having the required boundary value

# Test Cases: Hint

- Identify test cases by going through the test coverage items in order, and selecting the next uncovered boundary value for each parameter

# Pitfalls

- As for equivalence partition testing, boundary value analysis should use a minimal number of tests
- Do **not** provide test cases for every combination of input boundary values
- Do **not** provide a separate test case for each test coverage item

# Evaluation

- Boundary value analysis enhances the testing provided by equivalence partitions
- Experience indicates that this is likely to find significantly more errors than equivalence partitions
- Boundary value analysis tests two additional values from every input and output partition, the minimum and maximum
- These tests provide some assurance that the correct decisions are made in the code
- Boundary value analysis provides exactly the same test coverage items as equivalence partitions for boolean and for enumerated parameters

# Typical Fault in Boundary Value Handling

```
1    // Return true if x is greater than 100
2    public void greater(int x) {
3        if (!(x<100)) return true;
4        else return false;
5    }
```

- Line 3 contains a fault  
**!(x<100) is wrong**
- Either !(x<=100) or (x>100) would be correct

# BVA: Decisions and Combinations

- Boundary value analysis does **not** explore the decisions in detail
- In particular BVA does not explore decisions associated with different combinations of inputs
- Combinations of inputs will be addressed next



# Limitations

- The software has passed all the EP & BVA tests
- So is it fault free?

# Is It Fault Free?

- Only exhaustive testing can answer this question
- Faults may remain
- Explore limitations of equivalence partition testing by injecting two different categories of fault into the code

# Correct Code

```
22      public static Status giveDiscount(long bonusPoints, boolean
        goldCustomer)
23      {
24          Status rv = FULLPRICE;
25          long threshold=120;
26
27          if (bonusPoints<=0)
28              rv = ERROR;
29
30          else {
31              if (goldCustomer)
32                  threshold = 80;
33              if (bonusPoints>threshold)
34                  rv=DISCOUNT;
35          }
36
37          return rv;
38      }
```


**NOTE:** If  
bonusPoints is  
greater than the  
threshold, return  
DISCOUNT

# Fault 2

- Equivalence partition tests are not designed to find faults at the values at each end of an equivalence partition
- If we inject a fault which moves the boundary value for the processing that returns DISCOUNT, then we do not expect to see any failed tests

```
33      if (bonusPoints>threshold)
34          rv=DISCOUNT;

33      if (bonusPoints>=threshold) // fault 2
34          rv=DISCOUNT;
```



# Run BVA Tests Against Fault 2



```
PASSED: test_giveDiscount("T1.1", 40, true, FULLPRICE)
PASSED: test_giveDiscount("T1.2", 100, false, FULLPRICE)
PASSED: test_giveDiscount("T1.3", 200, false, DISCOUNT)
PASSED: test_giveDiscount("T1.4", -100, false, ERROR)
PASSED: test_giveDiscount("T2.1", 1, true, FULLPRICE)
PASSED: test_giveDiscount("T2.2", 80, false, FULLPRICE)
PASSED: test_giveDiscount("T2.3", 81, false, FULLPRICE)
PASSED: test_giveDiscount("T2.5", 121, false, DISCOUNT)
PASSED: test_giveDiscount("T2.6", 9223372036854775807, false, DISCOUNT)
PASSED: test_giveDiscount("T2.7", -9223372036854775808, false, ERROR)
PASSED: test_giveDiscount("T2.8", 0, false, ERROR)
FAILED: test_giveDiscount("T2.4", 120, false, FULLPRICE)
java.lang.AssertionError: expected [FULLPRICE] but found [DISCOUNT]
```

```
=====
Command line suite
```

```
Total tests run: 12, Passes: 11, Failures: 1, Skips: 0
```

# Notes

- The fault is found by one of the boundary value analysis tests
- This is expected as the fault was inserted at the processing of a boundary value

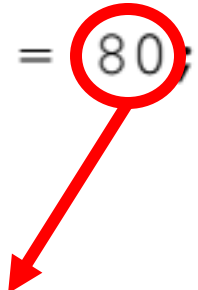
`bonusPoints >= threshold`

- Note: All the tests are run, even though one of them has failed

# Fault 3

- EP and BVA tests are not designed to find faults associated with the correct processing of combinations of values from the input partitions
- If we inject a fault that causes a combination of input values to be incorrectly processed, then we do not expect to see any failed tests (note: this fault is based on a review of the EP and BVA test cases)

```
--  
31         if (goldCustomer)  
32             threshold = 80;  
  
31         if (goldCustomer)  
32             threshold = 120; // fault 3
```



# Run BVA Tests Against Fault 3



```
PASSED: test_giveDiscount("T1.1", 40, true, FULLPRICE)
PASSED: test_giveDiscount("T1.2", 100, false, FULLPRICE)
PASSED: test_giveDiscount("T1.3", 200, false, DISCOUNT)
PASSED: test_giveDiscount("T1.4", -100, false, ERROR)
PASSED: test_giveDiscount("T2.1", 1, true, FULLPRICE)
PASSED: test_giveDiscount("T2.2", 80, false, FULLPRICE)
PASSED: test_giveDiscount("T2.3", 81, false, FULLPRICE)
PASSED: test_giveDiscount("T2.4", 120, false, FULLPRICE)
PASSED: test_giveDiscount("T2.5", 121, false, DISCOUNT)
PASSED: test_giveDiscount("T2.6", 9223372036854775807, false, DISCOUNT)
PASSED: test_giveDiscount("T2.7", -9223372036854775808, false, ERROR)
PASSED: test_giveDiscount("T2.8", 0, false, ERROR)
```

```
=====
```

```
Command line suite
```

```
Total tests run: 12, Passes: 12, Failures: 0, Skips: 0
```

```
=====
```



# Notes

- The fault is not discovered by our tests
- In a small example like this, depending on the data values selected, the equivalence partition and boundary value analysis tests might have found this combination of values that expose this fault
- But in general, and for larger examples, neither equivalence partition nor boundary value analysis provide a systematic way to find and test all combinations

# Demonstrating The Fault

- The results of executing the code with Fault 3, using specially selected input values, are:

```
$ check 100 true  
FULLPRICE
```



- Here, the wrong result is returned for the inputs (100,true)
  - The correct result is DISCOUNT
  - Not FULLPRICE

# Strengths & Weaknesses

- Strengths
  - Test data values are directly provided by the technique
  - Tests focus on areas where faults are more likely to be created by the developer
- Weaknesses
  - Doubles the number of test coverage item compared to EP
  - Combinations of values from different input partitions are not tested

# Key Points

- Boundary value analysis is used to ensure correct processing with data values at the start and end of every equivalence partition
- Each boundary value is a test coverage item
- Each boundary value is used in a test case

# Notes for Experienced Testers

- May add boundary values to the test code directly from the specification
  - Does not allow the test design process to be easily reviewed
  - Not good practice in a mission critical development environment

# Notes for Experienced Testers (continued)

- The example demonstrates “2-point boundary values\*”:
  - At each boundary between partitions, two value are selected
  - The value just below it (top of the lower range)
  - The value just above (bottom of the upper range)
  - The ranges “ .. 0][1 .. ” gives the boundary values 0 and 1
- Experienced testers may decide to use “3-point boundary values\*”:
  - At the top of a partition, one additional value, just below the upper boundary value
  - At the bottom of a partition, one additional value, just above the lower boundary value
  - Consider the ranges “ .. 0][1 .. ”
    - The range “ .. 0]" gives the boundary values -1, 0, 1
    - The range "[1 .. ” gives the boundary values 0,1,2
    - The ranges "" .. 0][1 .. ” gives the boundary values -1, 0, 1, 2

(\*) Standard terminology...