ESOF-3255 Software Testing and Quality Assurance

Dr. Rachid Benlamri

Project 1

Khanjan Dabhi

0872326

Part 1

Functional Testing

Introduction: Finding Test Cases

Starting with Boundary Value Testing, the test methods based on the input domain of a function (program) are the most rudimentary of all specification-based testing methods. They share the common assumption that the input variables are truly independent; and when this assumption is not warranted, the methods generate unsatisfactory test cases (such as June 31, 1912).

Equivalence class testing is indicated when the program function is complex. In such cases, the complexity of the function can help identify useful equivalence classes, as in the NextDate function. Strong equivalence class testing makes a presumption that the variables are independent, and the corresponding multiplication of test cases raises issues of redundancy. If any dependencies occur, they will often generate “error” test cases, as they did in the NextDate function.

However, this makes it a perfect example for decision table–based testing, because decision tables can highlight such dependencies. The decision table format lets us emphasize such dependencies using the notion of the “impossible” action to denote impossible combinations of conditions (which are impossible rules)

Case ID	Month	Day	Year	Expected Output
1	April	15	2001	April 16,2001
2	April	30	2001	May 1,2001
3	April	31	2001	Impossible
4	January	15	2001	January 16,2001
5	January	31	2001	February 1,2001
6	December	15	2001	December 16,2001
7	December	31	2001	January 1,2002
8	February	15	2001	February 16,2001
9	February	28	2004	February 29,2004
10	February	28	2001	March 1,2001
11	February	29	2004	March 1,2004
12	February	29	2001	Impossible
13	February	30	2001	Impossible

1 Test Case Specification

1.1 Outline

The following is a functional testing of *NextDate* function from *BlackBox.exe.* The types of testing method to produce test case is Decision Tree.

1.1.1 Test items

The *NextDate* function's features will be exercised by this test case

1.1.2 Input Specifications

Data

- **Month**- [January, February, March, April, May, June, July, August, September, October, November, December]
- **Day**- [1,2…,31]
- **Year**- [1812,1813…,2012]

Expected Output- Expected output of given date either a **Date** or "Impossible"

1.1.3 Output Specifications

Data

- **Month**- [January, February, March, April, May, June, July, August, September, October, November, December]
- **Day**- [1,2…,31]
- **Year**- [1812,1813…,2012]

Status

Pass – Output matches expected output

Fail - Output does not match expected output

1.2 Test Log

Given Below is the log for results of all the test cases and their result status

```
"""This are the Results"""
"April",15,2001,"April 16, 2001","April 16, 2001","Pass"
"April",30,2001,"May 1, 2001","May 1, 2001","Pass"
"April",31,2001,"Impossible","May 1, 2001","Fail"
"January",15,2001,"January 16, 2001","January 16, 2001","Pass"
"January",31,2001,"February 1, 2001","February 1, 2001","Pass"
"December",15,2001,"December 16, 2001","December 16, 2001","Pass"
"December",31,2001,"January 1, 2002","January 1, 2001","Fail"
"February",15,2001,"February 16, 2001","February 16, 2001","Pass"
"February",28,2004,"February 29, 2004","February 29, 2004","Pass"
"February",28,2001,"March 1, 2001","March 1, 2001","Pass"
"February",29,2004,"March 1, 2004","March 1, 2004","Pass"
"February",29,2001,"Impossible","March 1, 2001","Fail"
"February",30,2001,"Impossible","March 1, 2001","Fail"
```

1.3 Test Incident Report

After execution there were 4 failed test cases. They were cases 3,7,12,13. This indicates that the function is not able to give an error when the input is invalid (3,12,13). However, the error for test case 7 is different.

1.4 Test Summary Report

Q1) What confidence do you have that your test(s) have uncovered all the errors in the code? On what premises do you base your confidence level?

I have a moderate confidence that this test has uncovered all the errors in the code. Since, the test cases were prepared from a decision tree with various parameters and after executing them few errors were also discovered.

Q2) Speculate what bugs you might have missed with your test(s) and why?

Since there was an error in giving a “invalid input” for invalid number of days in the test cases, we can infer that there might be more errors like this in the function.

Q3) Was the specification sufficiently detailed? Justify your answer:

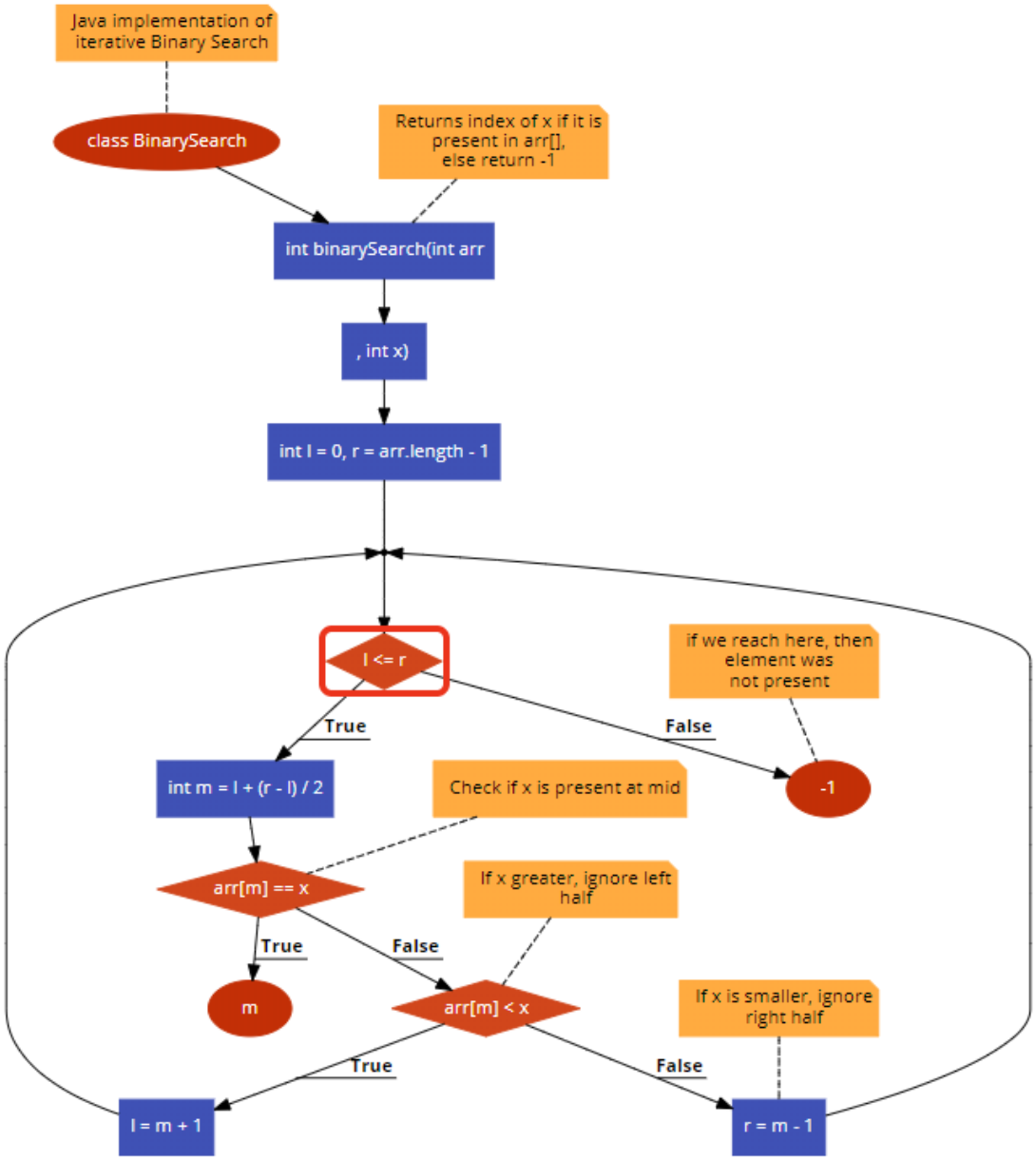
Yes, the specification was sufficiently detailed since there was not much information given related to the code as this was a functional testing.

PART 2
Structural Testing

Introduction

The class file was used to create a Flow Diagram which helped identify Test Cases given below is the flow diagram and the Test Data obtained from the test cases

```java
// Java implementation of iterative Binary Search
class BinarySearch {
    // Returns index of x if it is present in arr[],
    // else return -1
    int binarySearch(int arr[], int x,int n )
    {
        int l = 0, r = n - 1;
        while (l <= r) {
            int m = l + (r - l) / 2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was
        // not present
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr, x,n);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at "
                               + "index " + result);
    }
}
```


Java implementation of iterative Binary Search
class BinarySearch
Returns index of x if it is present in arr[], else return -1
int binarySearch(int arr
, int x)
int l = 0, r = arr.length - 1
l <= r
True
False
if we reach here, then element was not present
-1
int m = l + (r - l) / 2
Check if x is present at mid
arr[m] == x
True
False
m
If x greater, ignore left half
arr[m] < x
True
False
If x is smaller, ignore right half
l = m + 1
r = m - 1

Case ID	a	x	n	Expected Output{i}
1	{1,2,3}	4	3	-1
2	{1,2,3}	2	3	1
3	{1,2,3}	3	3	2
4	{1,2,3}	1	3	0
5	{2,3,4,10,40}	10	5	3
6	{2,3,4,10,40}	2	5	0
7	{0}	1	1	-1

2 Test Case Specification

2.1 Outline

The following is a structural testing of BinarySearch class written in Java. A flow graph is used to produce test cases.

2.1.1 Test Items

A test driver will be used to execute all the test cases for the given class file

2.1.2 Input Specifications

- a- An Array of float values
- n- Size of array a
- v- Value to search for

Precondition

- 0<= n < Size of array a
- For all i, j with 0 <= i < j < n, a[i] <= a[j]

Postcondition

a[result] == v or (result == -1 and there is no i, 0 <= i < n, such that a[i] == v)

2.1.3 Output Specifications

An index into array a or -1

2.2 Test Log

Case ID	a	x	n	Expected Output{i}	Actual Output
1	{1,2,3}	4	3	-1	-1
2	{1,2,3}	2	3	1	1
3	{1,2,3}	3	3	2	2
4	{1,2,3}	1	3	0	0
5	{2,3,4,10,40}	10	5	3	3
6	{2,3,4,10,40}	2	5	0	0
7	{0}	1	1	-1	-1

2.3 Test Incident Report

No incident report since there were no failed test cases (with given pre-conditions).

2.4 Test Summary Report

Q1) What confidence do you have that your test(s) have uncovered all the errors in the code? On what premises do you base your confidence level?

I have high confidence that the test cases have uncovered all the errors in the code. Since a flow diagram was used, we can find all the possible ways to reach the end statement and they have been implemented in the given test cases. However, invalid inputs have not been included in the test casus since we already knew the preconditions for the class function.

Q2) Speculate what bugs you might have missed with your test(s) and why?

The only bug that might have been missed is the if the array was not sorted in ascending order beforehand and search will shoot to the wrong end of the array.

Q3) Was the specification sufficiently detailed? Justify your answer:

Yes, the specification was sufficiently detailed since we have the source code a white board testing was very helpful in

making critical test cases. All the test data can be used to identify the test cases.