Homework #3

CSE 565: Software Verification/ Validation/ Test

(2015 FALL)

Submitted By:

Madhu Meghana Talasila (1207740881)

PART 1

1. Description of the Tool: EclEmma

EclEmma [1] is the tool that I used to demonstrate the code coverage on binary search algorithm. It is available as Eclipse plug-in. Prerequisites that EclEmma requires is Eclipse 3.5 or higher and Java 1.5 or higher. With EclEmma, code coverage analysis can done directly from eclipse. It is based on JaCoCo code coverage library [2].

Features of EclEmma:

1. **Fast develop/ test cycle**: we can run JUnit tests directly from workbench and analyze for code coverage [1].
2. **Rich coverage analysis**: Coverage results are summarized and are highlighted in Java source code. Different colors are used to demonstrate the coverage in the code and exact values are used to display the coverage in each class.
   Green: Complete Coverage
   Yellow: Partial Coverage
   Red: No Coverage [1].
3. **Non-invasive**: Installation of tool/ plug-in does not require any modifications to workbench or to the code [1].

Types of Coverage by EclEmma Covers

Statement Coverage

Decision Coverage

Branch Coverage

2. Source Listing of Binary Search [3]

BinarySearch.java

```
public class BinarySearch {
	/* Input should be in ascending order*/
	public boolean binarySearchAlgo(int[] number_list, int key){
		return search(number_list,key,0,number_list.length-1);
	}

	public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
	{

		if(number_list.length == 0){
			return false;
		}

		if (lowerIndex > upperIndex) {
	return false;
	}
```

```
            int middleIndex = (lowerIndex + upperIndex ) / 2;

            if (number_list[middleIndex] > key) {
      return search(number_list, key, lowerIndex, middleIndex - 1);
    } else if (number_list[middleIndex] < key) {
      return search(number_list, key, middleIndex + 1, upperIndex);
    } else {
      return true;
    }
}
```

3. First Set of Test Cases

```
package test;

import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
import main.BinarySearch;

public class BinarySearchTestCase1 {
	private BinarySearch test;
	int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

	@Before
	public void setUp() throws Exception{
		test = new BinarySearch();
	}

	@Test
	public void shouldReturnTrueIfFoundInEndOfArray() {
	  assertEquals(true, test.binarySearchAlgo(number_list_test, 4));
	}

	@Test
	public void shouldReturnFalseIfArrayIsEmpty() {
		assertEquals(false,test.binarySearchAlgo(new int[]{}, 1));
	}
}
```

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	33.9 %	102	199	301
src	33.9 %	102	199	301
test	26.0 %	63	179	242
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase2.java	0.0 %	0	52	52
BinarySearchTestCase1.java	100.0 %	63	0	63
main	66.1 %	39	20	59
BinarySearch.java	66.1 %	39	20	59

Figure 1 Coverage of First Set of Test Cases

Second Set of Test Cases

```
package test;

import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
import main.BinarySearch;

public class BinarySearchTestCase2 {
	private BinarySearch test;
	int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

	@Before
	public void setUp() throws Exception{
		test = new BinarySearch();
	}

	@Test
	public void shouldReturnTrueIfFoundInBeginningOfArray() {
	 assertEquals(true, test.binarySearchAlgo(number_list_test, 1));
	}

	@Test
	public void shhouldReturnFalseIfInputArrayIsNotInteger(){
	   assertEquals(false,test.binarySearchAlgo(new int[]{'1','2','3','4','5'}, 1));
	}


	@Test
	public void shouldReturnTrueIfFoundInMiddleOfArray() {
	  assertEquals(true, test.binarySearchAlgo(number_list_test, 7));
	}

	@Test
	public void shouldReturnFalseIfNotFoundInArray() {
	   assertEquals(false, test.binarySearchAlgo(number_list_test, 10));
	}

	@Test
	public void shouldReturnFalseIfListIsEmpty() {
	   assertEquals(false, test.binarySearchAlgo(new int[]{}, 10));
	}
}
```

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	46.8 %	175	199	374
src	46.8 %	175	199	374
test	36.8 %	116	199	315
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	116	0	116
main	100.0 %	59	0	59
BinarySearch.java	100.0 %	59	0	59

Figure 2 Coverage for Second Set of Test Cases

4. Screen Shots

Test Case 1: From first set of test cases. Code coverage for this test case is 62.7%

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  import main.BinarySearch;
9
10 public class BinarySearchTestCase2 {
11     private BinarySearch test;
12     int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};
13
14     @Before
15     public void setUp() throws Exception{
16         test = new BinarySearch();
17     }
18
19     @Test
20     public void shouldReturnTrueIfFoundInEndOfArray() {
21       assertEquals(true, test.binarySearchAlgo(number_list_test, 4));
22     }
23
24 }
25
```

Figure 3 Code for Test Case 1

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	28.7 %	89	221	310
src	28.7 %	89	221	310
test	20.7 %	52	199	251
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	52	0	52
main	62.7 %	37	22	59
BinarySearch.java	62.7 %	37	22	59

Figure 4 Coverage for Test Case 1

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```
1  package main;
2
3  public class BinarySearch {
4      /* Input should be in ascending order*/
5      public boolean binarySearchAlgo(int[] number_list, int key){
6          return search(number_list,key,0,number_list.length-1);
7      }
8
9      public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
10     {
11
12         if(number_list.length == 0){
13             return false;
14         }
15
16         if (lowerIndex > upperIndex) {
17             return false;
18          }
19
20         int middleIndex = (lowerIndex + upperIndex) / 2;
21
22         if (number_list[middleIndex] > key) {
23             return search(number_list, key, lowerIndex, middleIndex - 1);
24         } else if (number_list[middleIndex] < key) {
25             return search(number_list, key, middleIndex + 1, upperIndex);
26         } else {
27             return true;
28         }
29     }
30 }
31
```

Figure 5 Code Coverage for Test Case 1

Test Case 2: From first set of test cases. Code coverage for this test is 30.5%

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package test;

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

import main.BinarySearch;

public class BinarySearchTestCase2 {
    private BinarySearch test;
    int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

    @Before
    public void setUp() throws Exception{
        test = new BinarySearch();
    }

    @Test
    public void shouldReturnFalseIfArrayIsEmpty() {
        assertEquals(false,test.binarySearchAlgo(new int[]{}, 1));
    }

}
```

Figure 6 Code for Test Case 2

BinarySearchTestCase2 (Oct 20, 2015 11:41:43 AM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	22.6 %	70	240	310
src	22.6 %	70	240	310
test	20.7 %	52	199	251
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	52	0	52
main	30.5 %	18	41	59
BinarySearch.java	30.5 %	18	41	59

Figure 7 Coverage for Test Case 2

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package main;

public class BinarySearch {
    /* Input should be in ascending order*/
    public boolean binarySearchAlgo(int[] number_list, int key){
        return search(number_list,key,0,number_list.length-1);
    }

    public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
    {

        if(number_list.length == 0){
            return false;
        }

        if (lowerIndex > upperIndex) {
            return false;
        }

        int middleIndex = (lowerIndex + upperIndex) / 2;

        if (number_list[middleIndex] > key) {
            return search(number_list, key, lowerIndex, middleIndex - 1);
        } else if (number_list[middleIndex] < key) {
            return search(number_list, key, middleIndex + 1, upperIndex);
        } else {
            return true;
        }
    }
}
```

Figure 8 Code Coverage for Test Case 2

Test Case 3: From second set of test cases. Code coverage for this test is 78.0%.

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | *BinarySearchTestCase2.java

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  import main.BinarySearch;
9
10 public class BinarySearchTestCase1 {
11     private BinarySearch test;
12     int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};
13
14     @Before
15     public void setUp() throws Exception{
16         test = new BinarySearch();
17     }
18
19     @Test
20     public void shouldReturnTrueIfFoundInBeginningOfArray() {
21         assertEquals(true, test.binarySearchAlgo(number_list_test, 1));
22     }
23
24  /*     @Test
```

Problems | Javadoc | Declaration | Console | Coverage

BinarySearchTestCase1 (Oct 20, 2015 11:00:46 AM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	30.3 %	98	225	323
src	30.3 %	98	225	323
test	19.7 %	52	212	264
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase2.java	0.0 %	0	85	85
BinarySearchTestCase1.java	100.0 %	52	0	52
main	78.0 %	46	13	59
BinarySearch.java	78.0 %	46	13	59

Figure 9 Test Case 3 and Coverage

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | *BinarySearchTestCase2.java

```
1  package main;
2
3  public class BinarySearch {
4      /* Input should be in ascending order*/
5      public boolean binarySearchAlgo(int[] number_list, int key){
6          return search(number_list,key,0,number_list.length-1);
7      }
8
9      public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
10     {
11
12         if(number_list.length == 0){
13             return false;
14         }
15
16         if (lowerIndex > upperIndex) {
17             return false;
18         }
19
20         int middleIndex = (lowerIndex + upperIndex) / 2;
21
22         if (number_list[middleIndex] > key) {
23             return search(number_list, key, lowerIndex, middleIndex - 1);
24         } else if (number_list[middleIndex] < key) {
25             return search(number_list, key, middleIndex + 1, upperIndex);
26         } else {
27             return true;
28         }
29     }
30 }
31
```

Figure 10 Code Coverage for Test Case 3

Test Case 4: From second set of test cases. Code coverage for this test case is 69.5%

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | *BinarySearchTestCase2.java

```
1  package test;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  import main.BinarySearch;
9
10 public class BinarySearchTestCase1 {
11     private BinarySearch test;
12     int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};
13
14     @Before
15     public void setUp() throws Exception{
16         test = new BinarySearch();
17     }
18
19     /*@Test
20     public void shouldReturnTrueIfFoundInBeginningOfArray() {
21         assertEquals(true, test.binarySearchAlgo(number_list_test, 1));
22     } */
23
24     @Test
25     public void shhouldReturnFalseIfInputArrayIsNotInteger(){
26        assertEquals(false,test.binarySearchAlgo(new int[]{'1','2','3','4','5'}, 1));
27     }
28
29 }
30
```

Figure 11 Test Case 4

Problems | Javadoc | Declaration | Console | Coverage

BinarySearchTestCase1 (Oct 20, 2015 11:31:22 AM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	32.9 %	113	230	343
src	32.9 %	113	230	343
test	25.4 %	72	212	284
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase2.java	0.0 %	0	85	85
BinarySearchTestCase1.java	100.0 %	72	0	72
main	69.5 %	41	18	59
BinarySearch.java	69.5 %	41	18	59

Figure 12 Coverage for Test Case 4

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | *BinarySearchTestCase2.java

```
1  package main;
2
3  public class BinarySearch {
4      /* Input should be in ascending order*/
5      public boolean binarySearchAlgo(int[] number_list, int key){
6          return search(number_list,key,0,number_list.length-1);
7      }
8
9      public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
10     {
11
12         if(number_list.length == 0){
13             return false;
14         }
15
16         if (lowerIndex > upperIndex) {
17             return false;
18          }
19
20         int middleIndex = (lowerIndex + upperIndex) / 2;
21
22         if (number_list[middleIndex] > key) {
23             return search(number_list, key, lowerIndex, middleIndex - 1);
24         } else if (number_list[middleIndex] < key) {
25             return search(number_list, key, middleIndex + 1, upperIndex);
26         } else {
27             return true;
28         }
29     }
30 }
```

Figure 13 Code Coverage for Test Case 4

Test Case 5: From second set of test cases. Code coverage for test case is 78.0%

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package test;

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

import main.BinarySearch;

public class BinarySearchTestCase2 {
    private BinarySearch test;
    int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

    @Before
    public void setUp() throws Exception{
        test = new BinarySearch();
    }

    @Test
    public void shouldReturnTrueIfFoundInMiddleOfArray() {
      assertEquals(true, test.binarySearchAlgo(number_list_test, 7));
    }

}
```

Figure 14 Code for Test Case 5

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	31.6 %	98	212	310
src	31.6 %	98	212	310
test	20.7 %	52	199	251
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	52	0	52
main	78.0 %	46	13	59
BinarySearch.java	78.0 %	46	13	59

Figure 15 Coverage for Test Case 5

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package main;

public class BinarySearch {
    /* Input should be in ascending order*/
    public boolean binarySearchAlgo(int[] number_list, int key){
        return search(number_list,key,0,number_list.length-1);
    }

    public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
    {

        if(number_list.length == 0){
            return false;
        }

        if (lowerIndex > upperIndex) {
            return false;
         }

        int middleIndex = (lowerIndex + upperIndex) / 2;

        if (number_list[middleIndex] > key) {
            return search(number_list, key, lowerIndex, middleIndex - 1);
        } else if (number_list[middleIndex] < key) {
            return search(number_list, key, middleIndex + 1, upperIndex);
        } else {
            return true;
        }
    }
}
```

Figure 16 Code Coverage for Test Case 5

Test Case 6: From second set of test cases. Code Coverage for this test case is 78.0%

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package test;

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

import main.BinarySearch;

public class BinarySearchTestCase2 {
    private BinarySearch test;
    int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

    @Before
    public void setUp() throws Exception{
        test = new BinarySearch();
    }

    @Test
    public void shouldReturnFalseIfNotFoundInArray() {
        assertEquals(false, test.binarySearchAlgo(number_list_test, 10));
    }

}

```

Figure 17 Code for Test Case 6

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	31.6 %	98	212	310
src	31.6 %	98	212	310
test	20.7 %	52	199	251
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	52	0	52
main	78.0 %	46	13	59
BinarySearch.java	78.0 %	46	13	59

Figure 18 Coverage for Test Case 6

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package main;

public class BinarySearch {
    /* Input should be in ascending order*/
    public boolean binarySearchAlgo(int[] number_list, int key){
        return search(number_list,key,0,number_list.length-1);
    }

    public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
    {

        if(number_list.length == 0){
            return false;
        }

        if (lowerIndex > upperIndex) {
            return false;
         }

        int middleIndex = (lowerIndex + upperIndex) / 2;

        if (number_list[middleIndex] > key) {
            return search(number_list, key, lowerIndex, middleIndex - 1);
        } else if (number_list[middleIndex] < key) {
            return search(number_list, key, middleIndex + 1, upperIndex);
        } else {
            return true;
        }
    }
}

```

Figure 19 Code Coverage for Test Case 6

Test Case 7: From second set of test cases

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package test;

import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
import main.BinarySearch;

public class BinarySearchTestCase2 {
    private BinarySearch test;
    int[] number_list_test = new int[] { 1, 2, 3, 4, 5, 6, 7};

    @Before
    public void setUp() throws Exception{
        test = new BinarySearch();
    }

    @Test
    public void shouldReturnFalseIfListIsEmpty() {
       assertEquals(false, test.binarySearchAlgo(new int[]{}, 10));
    }
}

```

Figure 20 Code for Test Case 7

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Binary_Search	22.6 %	70	240	310
src	22.6 %	70	240	310
test	20.7 %	52	199	251
BinarySearchTest.java	0.0 %	0	127	127
BinarySearchTestCase1.java	0.0 %	0	72	72
BinarySearchTestCase2.java	100.0 %	52	0	52
main	30.5 %	18	41	59
BinarySearch.java	30.5 %	18	41	59

Figure 21 Coverage for Test Case 7

BinarySearch.java | BinarySearchTest.java | BinarySearchTestCase1.java | BinarySearchTestCase2.java

```java
package main;

public class BinarySearch {
    /* Input should be in ascending order*/
    public boolean binarySearchAlgo(int[] number_list, int key){
        return search(number_list,key,0,number_list.length-1);
    }

    public boolean search(int[] number_list, int key, int lowerIndex, int upperIndex)
    {

        if(number_list.length == 0){
            return false;
        }

        if (lowerIndex > upperIndex) {
           return false;
         }

        int middleIndex = (lowerIndex + upperIndex) / 2;

        if (number_list[middleIndex] > key) {
           return search(number_list, key, lowerIndex, middleIndex - 1);
        } else if (number_list[middleIndex] < key) {
           return search(number_list, key, middleIndex + 1, upperIndex);
        } else {
           return true;
        }
    }
}

```

Figure 22 Code Coverage for Test Case 7

5. Evaluation of Tool Usefulness

EclEmma is a very useful tool to find code coverage. It is available as a plug-in with public license in eclipse which makes the tool easier to download [4] and use. Also, it gives statement, condition and decision coverage. Tool uses symbols and colors to differentiate the coverage. A diamond symbol is showed by the side of the code which defines whether that particular branch is covered or missed. It used 3 different colors to show which statements are executed, partially executed and not executed. Moreover, it shows coverage, covered instructions, missed instructions and total instructions in each level of the project (i.e., project as total and also each class individually). Using all the above features, developer can easily improve the quality of test cases by achieving more coverage for each test case or writing test cases which gives more coverage. Displayed figures are very useful for the management to know the quality of the test cases written by developers.

Part 2

1. Static Source Code Analysis Tool – PMD

PMD (Programming Mistake Detector) is a static code analysis tool. PMD is available as a plugin in eclipse with public license. It supports Java, JavaScript, XML, and XSL. It includes CPD (Copy Paste Detector) which is used to find copy paste code in Java, C, C++, C#, PHP, Ruby, FORTRAN, and JavaScript [5]. It shows analysis results in different colors.

Types of Analysis PMD provides [6]

1. Possible bugs: Empty code blocks. Example: try/ catch
2. Dead Code: Unused variables, parameters and methods
3. Overcomplicated expressions: Unnecessary loops
4. Suboptimal Code: unused String/ StringBuffer
5. Classes with high Cyclomatic Complexity
6. Duplicate Code: Detected using CPD

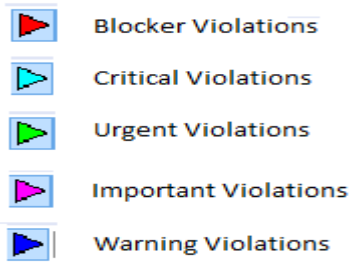

Figure 23 Colored classification

2. Source Listing of Code [3]

```
package main;
/**
 * @author Madhu Meghana
 * Class implementing binary search
 */
public class BinarySearch {
	/**
	 * @param number_list
	 * @param key
	 * @return
	 */
	public static boolean binarySearchAlgo(int[] number_list, int key){
		/* Anomaly 1 (Dataflow Anomaly: Possible bugs): empty try catch block*/
		try{

		} catch(Exception e){

		}
		return search(number_list,key,0,number_list.length-1);
	}

	/**
	 * @param number_list
	 * @param key
	 * @param lowerIndex
	 * @param upperIndex
	 * @return
	 */
	public static boolean search(int[] number_list, int key, int lowerIndex, int
upperIndex)
	{
		/* Anomaly 2 (Dataflow Anomaly) */
		int middleIndex = 0;
```

```
            middleIndex = (lowerIndex + upperIndex ) / 2;

            /* Anomaly 3 (Dataflow Anomaly: Dead Code) */
            if(5 < 3){
                  return true;
            }

            if(number_list.length == 0){
                  return false;
            }

            if (lowerIndex > upperIndex) {
                  return false;
            }
            if (number_list[middleIndex] > key) {
                  return search(number_list, key, lowerIndex, middleIndex - 1);
            } else if (number_list[middleIndex] < key) {
                  return search(number_list, key, middleIndex + 1, upperIndex);
            } else {
                  return true;
            }
      }
}
```

3. Screenshot showing analysis performed

Included data flow anomalies are caught by the tool. Please see fig [24].

Violations Outline

P	Line	created	Rule	Error Message
▶	8	Tue Oct 20 16:52:09 PDT 2015	UseUtilityClass	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
▶	15	Tue Oct 20 16:52:09 PDT 2015	VariableNamingConventions	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'number_list' is not final.
▶	15	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'number_list' is not assigned and could be declared final
▶	15	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'key' is not assigned and could be declared final
▶	17	Tue Oct 20 16:52:09 PDT 2015	EmptyTryBlock	Avoid empty try blocks
▶	17	Tue Oct 20 16:52:09 PDT 2015	EmptyTryBlock	Avoid empty try blocks
▶	19	Tue Oct 20 16:52:09 PDT 2015	EmptyCatchBlock	Avoid empty catch blocks
▶	19	Tue Oct 20 16:52:09 PDT 2015	EmptyCatchBlock	Avoid empty catch blocks
▶	19	Tue Oct 20 16:52:09 PDT 2015	AvoidCatchingGenericException	Avoid catching generic exceptions such as NullPointerException, RuntimeException, Exception in try-catch block
▶	33	Tue Oct 20 16:52:09 PDT 2015	VariableNamingConventions	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'number_list' is not final.
▶	33	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'number_list' is not assigned and could be declared final
▶	33	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'upperIndex' is not assigned and could be declared final
▶	33	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'lowerIndex' is not assigned and could be declared final
▶	33	Tue Oct 20 16:52:09 PDT 2015	MethodArgumentCouldBeFinal	Parameter 'key' is not assigned and could be declared final
▶	36	Tue Oct 20 16:52:09 PDT 2015	DataflowAnomalyAnalysis	Found 'DD'-anomaly for variable 'middleIndex' (lines '36'-'37').
▶	37	Tue Oct 20 16:52:09 PDT 2015	DataflowAnomalyAnalysis	Found 'DU'-anomaly for variable 'middleIndex' (lines '37'-'59').
▶	40	Tue Oct 20 16:52:09 PDT 2015	AvoidLiteralsInIfCondition	Avoid using Literals in Conditional Statements
▶	40	Tue Oct 20 16:52:09 PDT 2015	AvoidLiteralsInIfCondition	Avoid using Literals in Conditional Statements
▶	41	Tue Oct 20 16:52:09 PDT 2015	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
▶	45	Tue Oct 20 16:52:09 PDT 2015	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
▶	49	Tue Oct 20 16:52:09 PDT 2015	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
▶	53	Tue Oct 20 16:52:09 PDT 2015	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method
▶	55	Tue Oct 20 16:52:09 PDT 2015	OnlyOneReturn	A method should have only one exit point, and that should be the last statement in the method

Figure 24 Data Flow violations in the code

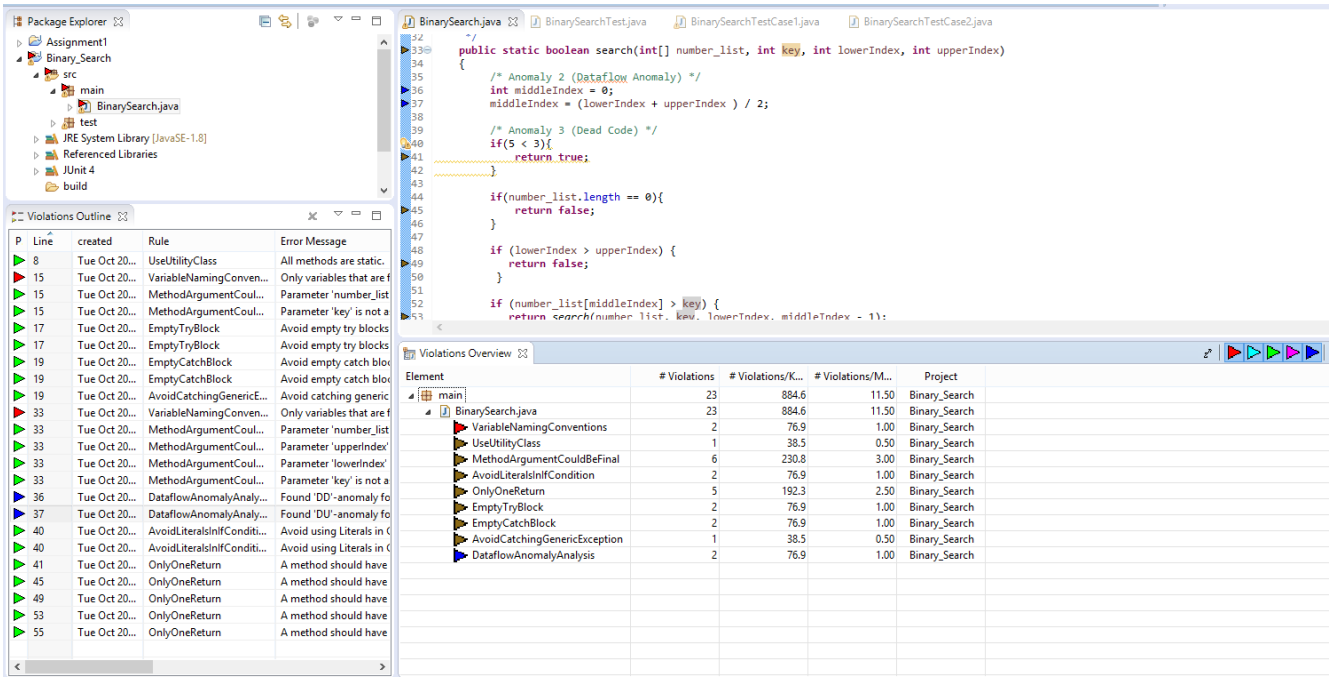

Figure 25 Analysis Performed

4. Evaluation of Tool Usefulness

PMD is available as a plugin from eclipse, it is easy to download [7], install and use. It shows various colors to categorize the anomalies which is used to know how critical a particular bug is and developer can learn from his/ her mistakes and can take care while writing code next time. Also, it shows how to fix the errors/ violations. It is very useful for beginner to fix the violations when prompts are given. Also user can define his own set of rules. It also generates reports to show static code analysis overtime. Static code analysis using PMD can be performed at any point in the implementation of code. Which helps developer to reduce the effort of correcting the violations after everything is done.

References

1. http://eclemma.org/index.html
2. http://eclemma.org/jacoco/
3. http://www.digizol.com/2013/08/java-binary-search-recursive-testcases.html
4. https://marketplace.eclipse.org/content/eclemma-java-code-coverage
5. http://sourceforge.net/projects/pmd/
6. https://en.wikipedia.org/wiki/PMD_(software)
7. https://marketplace.eclipse.org/content/eclipse-pmd