1. A Git directory with name PRT452 was created and the pdf file and the code project is uploaded. The link to my directory is given below.
https://github.com/acowboy7/PRT452

2**. a)** The java program to find whether a graph is connected or not is given below. This program was tested with Junit which supports xunit.

```
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
public class GraphProblem
{
    private Queue<Integer> queue;
    public GraphProblem()
   {
      queue = new LinkedList<Integer>();
    }
   public String bfs(int adjacency_matrix[][], int source){
    {
      int number_of_nodes = adjacency_matrix[source].length - 1;
       int[] visited = new int[number_of_nodes + 1];
       int i, element;
       visited[source] = 1;
       queue.add(source);
       while (!queue.isEmpty())
       {
          element = queue.remove();
          i = element;
           while (i <= number_of_nodes)
           {
             if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
             {
                 queue.add(i);
                 visited[i] = 1;
             }
             i++;
           }
      }
      boolean connected = false;
      for (int vertex = 1; vertex <= number_of_nodes; vertex++)
       {
         if (visited[vertex] == 1)
           {
              connected = true;

           } else
           {
              connected = false;
              break;
           }
```

```
            }
            if (connected)
            {
                System.out.println("The graph is connected");
                return ("connected");
            } else
            {
                System.out.println("The graph is disconnected");
                return ("not connected");
            }
        }
        }
        public static void main(String... arg){
            int number_no_nodes, source;
            Scanner scanner = null;
            try
            {
                System.out.println("Enter the number of nodes in the graph");
                scanner = new Scanner(System.in);
                number_no_nodes = scanner.nextInt();
                int adjacency_matrix[][] = new int[number_no_nodes + 1][number_no_nodes + 1];
                System.out.println("Enter the adjacency matrix");
                for (int i = 1; i <= number_no_nodes; i++)
                    for (int j = 1; j <= number_no_nodes; j++)
                        adjacency_matrix[i][j] = scanner.nextInt();
                for (int i = 1; i <= number_no_nodes; i++)
                {
                    for (int j = 1; j <= number_no_nodes; j++)
                    {
                        if (adjacency_matrix[i][j] ==  1 && adjacency_matrix[j][i] == 0)
                        {
                            adjacency_matrix[j][i] = 1;
                        }
                    }
                }
                System.out.println("Enter the source for the graph");
                source = scanner.nextInt();
                GraphProblem undirectedConnectivity= new GraphProblem();
                undirectedConnectivity.bfs(adjacency_matrix, source);
            } catch (InputMismatchException inputMismatch)
            {
                System.out.println("Wrong Input Format");
            }
           scanner.close();
        }
    }
```

Output:

First Ouput

```
run:
Enter the number of nodes in the graph
```

```
3
Enter the adjacency matrix
0 2 1
2 0 0
1 0 0
Enter the source for the graph
2
The graph is disconnected
```

Second Output

```
Enter the number of nodes in the graph
3
Enter the adjacency matrix
0 1 1
1 0 1
1 1 0
Enter the source for the graph
1
The graph is connected
BUILD SUCCESSFUL (total time: 47 seconds)
BUILD SUCCESSFUL (total time: 17 seconds)
```

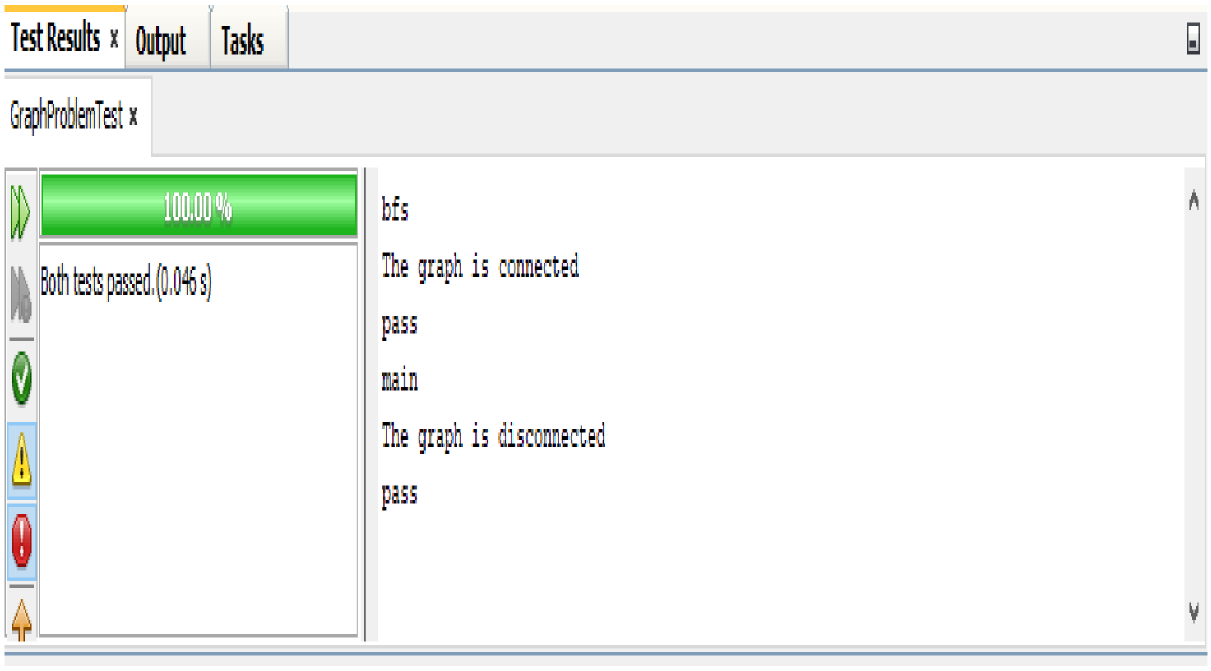
b) The test cases for the above program is given below

```
import java.util.InputMismatchException;
import java.util.Scanner;
import static org.junit.Assert.fail;
import org.junit.*;
/**
 *
* @author Yunesh
 */
public class GraphProblemTest {

    public GraphProblemTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
    public void setUp() {
    }

    @After
```

```
public void tearDown() {
}

/**
 * Test of bfs method, of class GraphProblem.
 */
@Test
public void testBfs() {
    System.out.println("bfs");
    int[][] adjacency_matrix = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
    int source = 1;
    GraphProblem instance = new GraphProblem();
    String output = instance.bfs(adjacency_matrix, source);
    String next_output1 = "connected";
    String next_output2 = "not connected";
    if (next_output1.equals(output)) {
        System.out.println("pass");
    } else if (next_output2.equals(output)) {
        System.out.println("pass");
    } else {
        fail("Not the desired result");

    }

}

/**
 * Test of main method, of class GraphProblem.
 */
@Test
public void testMain() {
    System.out.println("main");
    int number_no_nodes, source;
    number_no_nodes = 3;
    int[][] adjacency_matrix = {{0,0,0},{0,0,0},{0,0,0}};
    source = 1;
    GraphProblem undirectedConnectivity = new GraphProblem();
    String output = undirectedConnectivity.bfs(adjacency_matrix, source);
    String next_output1 = "connected";
    String next_output2 = "not connected";
    if (next_output1.equals(output)) {
        System.out.println("pass");
    } else if (next_output2.equals(output)) {
        System.out.println("pass");
    } else {
        fail("Not the desired result");


    }
}
```

}

c) At first when only the default test was created the test failed to run. After continuous updating and inserting of main class, methods and objects finally the test was successful.



Both tests passed.(0.046 s)

```
bfs
The graph is connected
pass
main
The graph is disconnected
pass
```

3. Code bad smell in computer programming refers to the symptoms in the program code which indicates the deeper problem. According to Martin Fowler, “a code smell is a surface indication that usually corresponds to a deeper problem in the system”. Code smells are not the bugs and technically they are not incorrect but rather they indicates the weakness in the design of the program which will slow down the development and increase the risk of the failure of the programs. Whereas fault in program causes the program to function in an inappropriate manner. This may be due to the incorrect steps, process or data definition in a computer program. Since the code smell increases the risk of failing the system it is however related to the faults.

Some of the types of code bad smells are:

a) Data Clumps
If there are identical group of parameters then they should be combined into the same class. This will eliminate the bad smell that may present when they are not classified into the same class. To ensure some data are clump not we have to remove one of the data values and see if any other data values make sense or not. If they didn’t make any sense then they should be combined into an object.

b) Message Chains
When a client request an object to perform some task and the object ask another one to do it. This is known as message chain. Message chain is one of the smell code which creates a fault later on when the previous is removed.

c) Speculative Generality
   There are lots of codes in the program which are written for the further development of the program or to support the future cases that may occur. But most of these are unused and this just increases the size of the fine and slows down the processing of the execution. These also fall under the code bad smell.
d) Switch Statement
   Most of the switch statement falls under the code bad smell. Whenever a new case has to be added then it's added into the same statement of switch which makes the program longer and the testing becomes complex. So instead of switch statement for each case new class can be added which can just be invoked if required.
e) Middle man
   If a class is linked to many methods and the methods refer to other object then there occurs a problem. If anyone of the method is no longer in use then the class cannot access the objects. This problem can be solved by creating the separate classes for methods and objects.

The five different types of Bugs in common apache packages are:

a. Exceptions are suppressed incorrectly when copying files (IO-502)

Type: Bug
Status: Closed
Priority: Critical
Resolution: Fixed
Affects Version/s: 2.4, 2.5
Fix Version/s: 2.6
Component/s: Utilities

The problem was generated in below code which has already been resolved.

```
output.close();
    output = null;

    fos.close();
    fos = null;

    input.close();
    input = null;

    fis.close();
    fis = null;
  } finally {
     IOUtils.closeQuietly(output, fos, input, fis);
  }
```

Here the problem occurred when copying files or directories the exceptions thrown on closing streams are called incorrectly. This needs to be rounded by the caller. This occurs when it closes the output object first taking the precedence over input. It also closes the channel before the stream.

b. Add cancel() to Finder (IO-31)

Type: Bug
Status: Closed
Priority: Major
Resolution: Duplicate
Affects Version/s: None
Fix Version/s: None
Component/s: None

This below code contained the duplicate code. This issue was resolved when this duplicate code was removed.

issue.field.bugzillaimportkey

c. Allow influencing the setter method used (DBUTILS-121)

Type: Improvement
Status: Resolved
Priority: Major
Resolution: Fixed
Affects Version/s: 1.6
Fix Version/s: 1.7

The following contains the fault and it was because of message chaining. Message chaining is one of the code smell which decays the code and led to the fault.

```
 protected Method getWriteMethod(Object target, PropertyDescriptor prop, Object value) {
    Method method;
    try {
      method = target.getClass().getMethod(prop.getWriteMethod().getName(), value.getClass());
    } catch (NoSuchMethodException e) {
      method = prop.getWriteMethod();
    }
    return method;
  }
```

d. ClassLoaderObjectInputStream does not handle primitive typed members (IO-368)

- **Type:** Bug
- **Status:** Closed
- **Priority:** Major
- **Resolution:** Fixed
- **Affects Version/s:** 2.0.1
- **Fix Version/s:** 2.5
- **Component/s:** Streams/Writers
- **Environment:** Single node computer, running standard JVM (Oracle 1.6.0)

The following code cannot contain the simple primitive like long and int. The class loader of the below code are not in specified class loader which may resolve classes that should fail. This was due to code smell and it’s been resolved now.

Code

```
ObjectInputStream ois = null;
   try {
     ois = new ClassLoaderObjectInputStream(getClass().getClassLoader(), new
ByteArrayInputStream(bytes));
     return (T) ois.readObject();
   } catch (ClassNotFoundException e) {
     LOGGER.error("Deserialization failed for {}", objectClass, e);
   } catch (IOException e) {
     LOGGER.error("Deserialization failed for {}", objectClass, e);
   } finally {
     if (ois != null) {
       try {
         ois.close();
       } catch (IOException ignored) {
       }
     }
   }
```

e. When using the 2.0 Commons IO, there is no sort method for LastModifiedFileComparator(IO-283)

Type: Bug
Status: Closed
Priority: Minor
Resolution: Invalid
Affects Version/s: 2.0
Fix Version/s: None
Component/s: None
Labels: None
Environment: Sparc Solaris 2.10

```
package myfiles.rename;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.DirectoryWalker;
import org.apache.commons.io.comparator.NameFileComparator;

public class FileRenamer extends DirectoryWalker<String> {

   @Override
   protected File[] filterDirectoryContents(File directory, int depth, File[] files) throws
IOException {
        NameFileComparator.NAME_COMPARATOR.sort(files);
        return files;
    }
}
```

In the above code, the error message is "The method sort(File[]) is undefined for the type Comparator<File>".

Here it needs to call sort on a Comparator instance and the Comparator interface does not define the sort method. To make this work, the code was changed to:

```
package myfiles.rename;

import java.io.File;
import java.io.IOException;

import org.apache.commons.io.DirectoryWalker;
import org.apache.commons.io.comparator.NameFileComparator;

public class FileRenamer extends DirectoryWalker<String> {

    private static final NameFileComparator COMPARATOR = new NameFileComparator();

    @Override
    protected File[] filterDirectoryContents(File directory, int depth, File[] files) throws
IOException {
        COMPARATOR.sort(files);
        return files;
    }
}
```

References

1. FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (1999) *Refactoring: Improving the Design of Existing Code*, Addison Wesley.

2. GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. (1995) *Design patterns: elements of reusable object-oriented software*, Reading, Mass., Addison-Wesley.

3. SEAMAN, C. B., SHULL, F., REGARDIE, M., ELBERT, D., FELDMANN, R. L., GUO, Y. & GODFREY, S. (2008) Defect categorization: making use of a decade of widely varying historical data. *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement.* Kaiserslautern, Germany, ACM.

4. ZIMMERMANN, T., PREMRAJ, R. & ZELLER, A. (2007) Predicting Defects for Eclipse. IN PREMRAJ, R. (Ed.) *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on.*

5. Team, C. (2017). *Commons IO – Commons IO Issue tracking*. [online] Commons.apache.org. Available at: https://commons.apache.org/proper/commons-io/issue-tracking.html

6. Commons.apache.org. (2017). *DbUtils – JDBC Utility Component*. [online] Available at: https://commons.apache.org/proper/commons-dbutils.

7. Refactoring.guru. *Refactoring and Design Patterns*. [online] Available at: https://refactoring.guru/

8. Sanfoundry. (2017). Java Program to Check whether Directed Graph is Connected using BFS - Sanfoundry. [online] Available at: http://www.sanfoundry.com/java-program-check-whether-directed-graph-connected-using-bfs/