Q1 GITHUB

Q2. xUnit test program

<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="xunit" version="2.2.0" targetFramework="net452" />

<package id="xunit.abstractions" version="2.0.1" targetFramework="net452" />

<package id="xunit.assert" version="2.2.0" targetFramework="net452" />

<package id="xunit.core" version="2.2.0" targetFramework="net452" />

<package id="xunit.extensibility.core" version="2.2.0" targetFramework="net452" />

<package id="xunit.extensibility.execution" version="2.2.0" targetFramework="net452" />

</packages>

using Xunit;

namespace graphs

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

PassingTest();

FailingTest();

}

[Fact]

public void PassingTest()

{

Assert.Equal(8, Add(5, 3));

}

[Fact]

public void FailingTest()

{

Assert.Equal(2, Add(3, 4));

}

int Add(int x, int y)

{

return x + y;

}

}

}

TDD code

package CheckGraphConnection;

import org.testng.Assert;

import org.testng.annotations.Test;

public class CheckGraphConnection{

@Test

public Boolean connected(Graph graph)  
{  
 Boolean connected = true;  
 string str\_index = vertexMap.Keys.First();  
 Vertex x\_axis = vertexMap[str\_index];  
 LinkedList<Vertex> Ls = new LinkedList<Vertex>();  
 Dictionary<string, Vertex> Km = vertexMap;  
 foreach(Edge edge in x\_axis.adj)  
{  
 Ls.AddFirst(edge.dest);  
}  
while(Km.Count != 0)  
{  
 find and remove some vertex y in K  
 for each edge (y, z)  
 if (z is not in L)  
 add z to both L and K   
}   
if(Ls.Count < n items)  
{

return false;  
}  
else  
{  
 return true;  
}  
} }

Q.3) CODE SMELL:

Code smell which is also referred to as bad smell in reference to computer programming is a term which refers to any kind of symptom or sign within the source code of any piece of software or program that indicates or shows that there might be a deeper problem with the program. Code re-factoring is the process or action of rearranging code in a computer which is modifying the factoring without changing the actual external behavior of a system or program. Refactoring of code improves the non functional attributes which are associated with a piece of software. Some common code smells which have been observed are duplicated code which is having very similar code replicated in many locations and a large class which is a class that has increased in size and has become too large.

Duplicated code can be solved by employing measures such as bakers algorithm to detect them and simply merge them with other similar code but still maintain the usability and purpose of the software. Duplicated code is present in software where you may find lines of code which are tasked with doing different actions but yet can be joined together to perform the same task. Refactoring duplicated code ensures that software metrics such as the lines of code coupling and cyclomatic complexity are improved. Refactoring of large classes also ensures that software metrics are improved and in general ensures that there is a lower compilation time for the code and decreased amount of human error in the code.

Reference List

Fowler M., Beck K., Brant J., Opdyke W. & Roberts D. *Refactoring: Improving the Design of Existing Code.* Addison Wesley. 1999.

Mens T. & Tourwe T. A survey of software refactoring. *Software Engineering, IEEE Transactions on 30,* 126-139. 2004.

DATA CLUMPS:

In the event that you generally observe similar data staying nearby together, perhaps it has a place together. Consider rolling the related data up into a bigger class.

MESSAGE CHAINS:

       Watch out for long successions of method calls or brief factors to get standard data. Intermediaries are conditions in mask.

example :cvs [server aborted]

MIDDLE MAN:

         On the off chance that a class is assigning all its work, why does it exist? Cut out the  middleman. Be careful classes that are only wrappers over different classes or existing usefulness in the system.

      example : cvs [command aborted]

SPECULATIVE GENERALITY:

             This code smell depicts a circumstance where individuals build up a class with a wide range of snares and uncommon cases to make sure it will deal with things that may be required later on yet not at this stage. The subsequent class is regularly harder to comprehend and keep up than if it just attempted to deal with the things that were really required.

        example :cvs update: could not patch file

SWITCH STATEMENTS:

                  Switch statements lead to Code Duplication and inhibit change.  Object Oriented switch is equal to Polymorphism.