VISUAL SOFTWARE ANALYTICS

Project

JAVA SourceCode Analyzer

Section- WED-THUS (08:30/10:00)

Group Members

S. No	Roll Number	Name
1	37888	Haris Bin Irfan
2	37728	Yasir Ul Haq
3	38633	Syed Usama Ahmed Hashmi

Submitted to: Sir. Hassan Adil Received: _____

Table of Contents

1 Introduction:

JAVA SourceCode Analyzer is a software metric calculator used to measure the size of a software program by counting the number of lines in the text of the program's source code. When you start the application and enter the folder path the user see a next output window in which user can easily view CLOC, SLOC, LOC, BLOC and McCabe CyclomaticComplexity.

2 Code:

2.1 CalculateLOC.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CalculateLOC
      {
       static ArrayList<String> result=new ArrayList<String>();
            public static void CalculateLOC(String Path) throws
IOException
            {
int LOC = 0, CLOC = 0, SLOC = 0, BLOC = 0,NCLOC=0,temp=0;
            File f=new File(Path);
            BufferedReader reader = new BufferedReader(new
FileReader(f));
             String line;
            while ((line = reader.readLine()) != null) {
            if (line.equals(""))
             BLOC++;


      if(line.contains("import")||line.contains("{")||line.contains(
"}")) {
                  temp++;
            }
```

```
        if (line.contains("/*") && !line.contains("*/"))
        {
            String x = "";
            CLOC++;
            while (!x.contains("*/"))
            {
                x = reader.readLine();
                CLOC++;
                LOC++;
            }
        }
        else if (line.contains("/*") && line.contains("*/"))
        {
            CLOC++;
        }
        else if (line.contains("//"))
        {
            CLOC++;
        }
        LOC++;

        SLOC=LOC-CLOC-BLOC;
    }
           result.add(LOC+"");
           result.add(CLOC+"");
           result.add(SLOC+"");
           result.add(BLOC+"");
           result.add((SLOC -temp)+"");
           }
}
```

2.2 McCabe.java

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class McCabe
{
	private String _sourceCode;

	public McCabe(String sourceCode)
	{
		this._sourceCode = sourceCode;
	}

	public final int CalculateCyclomaticComplexity()
	{
		final int numberOfConnectedComponentsCoefficient = 2;

		int numberOfGraphVertices = 0;
		int numberOfGraphArcs = 0;
		int count = 0;
		String loopsPattern = "(\\bswitch\\b)";

		Pattern r = Pattern.compile(loopsPattern);
		Matcher m = r.matcher(this._sourceCode);
		  while(m.find()) {
			count++; }

		numberOfGraphVertices += count;

		loopsPattern =
"(\\bcase\\b)|(\\bdefault\\b)|(\\bfor\\b)|(\\bforeach\\b)|(\\bwhile\
\b)";
		Pattern r1 = Pattern.compile(loopsPattern);
		Matcher m1= r1.matcher(this._sourceCode);
		count=0;
		  while(m1.find()) {
			count++; }

		numberOfGraphArcs += count;

		loopsPattern = "(\\bif\\b)";

		Pattern r2 = Pattern.compile(loopsPattern);
		Matcher m2= r2.matcher(this._sourceCode);
		count=0;
		  while(m2.find()) {
			count++; }
```

```
            numberOfGraphVertices += count;
            numberOfGraphArcs += 2 * count;

            numberOfGraphVertices++;
            return numberOfGraphArcs - numberOfGraphVertices +
numberOfConnectedComponentsCoefficient;
      }
}
```

2.3 Output :