

**Visual Software Analytics**

**Project Report “Code Analyser”**

**YouTube Link:**

<https://youtu.be/PSJDZqw8Yfc>

**Google Drive Link:**

<https://drive.google.com/file/d/18fh-zjBIku4oEyD-pRZRqASzraS_4Bdk/view?usp=sharing>

**Submitted By:**

Muhammad Mubashir Noor (49331)

Abeerah Shamshad (51722)

Syeda Sumbul Kazmi (49569)

Raja Amir (41986)

**Project Description:**

This project can analyse the java source code and it will show us numbers of total operators and operands and unique operators and operands, Linearly independent paths in code, LOC, BLOC, SLOC and NCLOC. Code Analyser tools can reduce the number of bugs in one program therefore it can reduce the cost of the program. Many developers don’t use these tools losing a lot of time with manual code analysis (in some cases there is no analysis at all) and a lot of money with resources are used to do the analysis. In this project, we will test and study the results of three type of analysis (Halstead Metrics, Cyclomatic Complexity, Organizational Complexity) that by being inexpensive can efficiently remove the most common vulnerabilities in software. It can be difficult to compare tools with different characteristics, but we can get interesting results by testing the tools. This project measure or calculate Halstead complexity which are software metrics by counting the tokens and determining which are operators and which are operands. It also measures Cyclomatic Complexity which is another software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if the second command might immediately follow the first command. It also measures Organizational Complexity and it can also analyse GitHub repository and its complexity and advise us how much work done, changes happened and how many files uploaded on GitHub.

**Functional Requirement**:

* **Java Code**:

It works with java source code only with the measurement of 3 code analyser complexity.

* **Halstead Metrices:**

It calculatesthe complexity by identifying the total and unique operators and operand in the code. The size of code is the sum of the number of operators and operands and source line of codes.

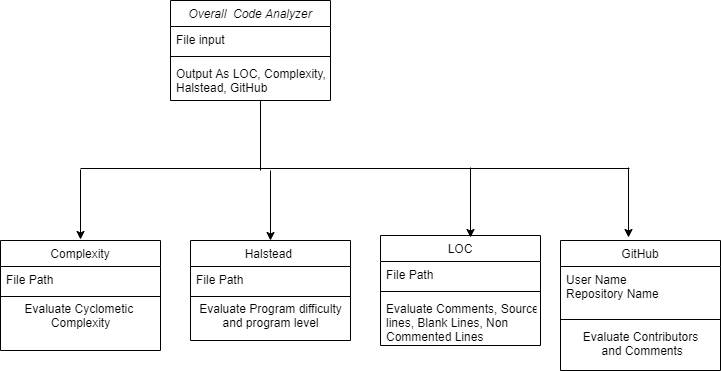
* **Cyclomatic Complexity**:

It tells us complexity by determining the linear independent path and the control flow of the project code.

* **Organizational Complexity**:

It identifies the number of developers and their frequency how often they push code in GitHub repository.

**Class Diagram:**

****

**Source Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Threading;

using System.Text.RegularExpressions;

using Octokit;

using ConsoleApplication5;

public delegate void ParameterizedThreadStart(string obj);

public delegate void ThreadStart();

namespace WindowsFormsApplication1

{

    public partial class Form1 : Form

    {

        List<FileDetail> Files;

        Tuple<int, int, int, int> H\_Matrix1;

        Tuple<double, double, double, double, double> H\_Matrix2;

        public static int tempCount = 0;

        public static Github gitInfo;

        public static string selectedFolder;

        int cloc = 0, loc = 0, bloc = 0, tsloc = 0, sloc = 0, ncloc = 0, abc;

        char star = 'e';

        char check = 'z';

        int f = 0, e = -1;

        char ex;

        int bd;

        string[] tc;

        public Form1()

        {

            InitializeComponent();

            Files = new List<FileDetail>();

            dataGridView1.Enabled = false;

            label30.Enabled = false;

            label31.Enabled = false;

            button3.Enabled = false;

            textBox2.Enabled = false;

            textBox1.Enabled = false;

            chart2.Titles.Add("Cyclometric Complexity");

            chart1.Titles.Add("Halstead Metrics");

            chart4.Titles.Add("Organizational Metrics");

        }

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

        {

            folderBrowserDialog1.ShowDialog();

            selectedFolder = folderBrowserDialog1.SelectedPath;

                if(selectedFolder != string.Empty)

            {

                tabControl1.Visible = true;

                button1.Enabled = false;

                button1.Enabled = false;

                panel2.Enabled = true;

                panel3.Enabled = true;

                dataGridView1.Enabled = true;

                    }

            Files = getFile(selectedFolder);

        }

        private List<FileDetail> getFile(string selectedFolderPath)

        {

            List<FileDetail> f = null;

            string[] files = Directory.GetFiles(selectedFolderPath, "\*.java", SearchOption.AllDirectories);

            f = new List<FileDetail>();

            DataTable dt = new DataTable();

            dt.Columns.Add("Name");

            dt.Columns.Add("Directory");

            foreach (var file in files)

            {

                FileInfo info = new FileInfo(file);

                DataRow dr = dt.NewRow();

                dr[0] = ""+info.Name;

                dr[1] = ""+info.DirectoryName;

                dt.Rows.Add(dr);

            }

            dataGridView1.DataSource = dt;

            return f;

        }

        private List<string> getFileList(string selectedFolderPath)

        {

            List<string> filePaths = null;

            string[] files = Directory.GetFiles(selectedFolderPath, "\*.java", SearchOption.AllDirectories);

            filePaths = new List<string>();

            foreach (var file in files)

            {

                FileInfo info = new FileInfo(file);

                filePaths.Add( info.DirectoryName+"//"+info.Name);

            }

            return filePaths;

        }

        private void button2\_Click(object sender, EventArgs e)

        {

            openFileDialog1.ShowDialog();

        }

        private void Form1\_Load(object sender, EventArgs e)

        {

        }

        private void listBox1\_SelectedIndexChanged(object sender, EventArgs e)

        {

        }

        private static Tuple<string[],int[],int> CalculateComplexity(string Path)

        {

            string line;

            int numberOfMethod=0;

            int[] MethodComplexity = new int[9999];

            string[] MethodName = new string[9999];

            Boolean lineIsMethod = false;

            int methodCounter = 0;

            string prevLine = "";

            System.IO.StreamReader str=null;

            try {

                 str = new System.IO.StreamReader(@"" +Path);

            }

            catch(Exception ex)

            {

                MessageBox.Show(ex.Message);

            }

            while ((line = str.ReadLine()) != null)

            {

                if (string.IsNullOrEmpty(line))

                    continue;

                if (line.Contains("void ") && line.Contains("(") && line.Contains(")") && !line.Contains("=") && !prevLine.Contains("new") && !line.Contains("for") ||

                    line.Contains("int ")  && line.Contains("(") && line.Contains(")") && !line.Contains("=") && !prevLine.Contains("new") && !line.Contains("for") ||

                    line.Contains("float ") && line.Contains("(") && line.Contains(")") && !line.Contains("=") && !prevLine.Contains("new") && !line.Contains("for") ||

                    line.Contains("string ") && line.Contains("(") && line.Contains(")") && !line.Contains("=") && !prevLine.Contains("new") && !line.Contains("for") ||

                    line.Contains("char ") && line.Contains("(") && line.Contains(")") && !line.Contains("=") && !prevLine.Contains("new") && !line.Contains("for")

                    )

                {

                    numberOfMethod++;

                    lineIsMethod = true;

                    methodCounter++;

                    if (line.Contains("protected"))

                    line= line.Replace("protected", "");

                        line = line.Replace("public", "");

                        line = line.Replace("private", "");

                        line = line.Replace("static", "");

                        line = line.Replace("final", "");

                        line = line.Replace("protected", "");

                    MethodComplexity[methodCounter]++;

                    line = line.Replace("void", "");

                    line = line.Replace("int", "");

                    line = line.Replace("float", "");

                    line = line.Replace("char", "");

                    line = line.Replace("string", "");

                    line = line.Replace("Boolean", "");

                    line = line.Replace("boolean", "");

                    line = line.Replace("String", "");

                    line = line.Replace("List<>", "");

                    line = line.Replace("BigInteger", "");

                    line = line.Replace("async", "");

                    line = line.Replace("synchronized", "");

                    var x = line.Split('(');

                    MethodName[methodCounter] = x[0];

                }

                if (lineIsMethod)

                {

                    if (line.Contains("if") || line.Contains("else") || line.Contains("case") || line.Contains("default"))

                    {

                        MethodComplexity[methodCounter]++;

                    }

                    if (line.Contains("for") || line.Contains("while") || line.Contains(" do ") || line.Contains("break") || line.Contains("continue"))

                    {

                        MethodComplexity[methodCounter]++;

                    }

                    if (line.Contains("&&") || line.Contains("||") || line.Contains("?"))

                    {

                        MethodComplexity[methodCounter]++;

                    }

                    if (line.Contains("catch") || line.Contains("finally") || line.Contains("throw") || line.Contains("throws"))

                    {

                        MethodComplexity[methodCounter]++;

                    }

                }

                prevLine = line;

}

            return Tuple.Create(MethodName,MethodComplexity, numberOfMethod);

        }

        private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

        {

        }

        private Tuple<double, double, double, double , double> CalculateHalstead2(int Operand, int n1, int Operator, int n2)

        {

            double ProgramLenght = 0;

            double SizeOfVocabulary = 0;

            double ProgramVolume = 0;

            double DifficultyLevel = 0;

            double ProgramLevel = 0;

            try {

             ProgramLenght =Operand+Operator;

             SizeOfVocabulary=n1+n2;

             ProgramVolume= ProgramLenght\*(Math.Log(SizeOfVocabulary, 2));

             DifficultyLevel = (n1/2)\*(Operator / n2) ;

             ProgramLevel = 1 / DifficultyLevel;

            }

            catch(Exception ex)

            {

                MessageBox.Show(ex.Message);

            }

            return Tuple.Create(ProgramLenght, SizeOfVocabulary, ProgramVolume, DifficultyLevel, ProgramLevel);

        }

        private Tuple<int,int,int,int> CalculateHalstead1(string filePath)

        {

          List<string> PostPath=  doPreProcessing(filePath);

            int Operater = 0, n1 = 0, Operand = 0, n2 = 0 ;

            List<string> l = new List<string>();

            List<string> Operads = new List<string>();

            Boolean[] UniqueOperands= new Boolean[150];

            Boolean method = false ;

            foreach(string l1 in PostPath)

            {

                var line = l1;

                if (line.Contains(" void") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" int") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" float") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" double") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" char") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" string") && line.Contains("(") && line.Contains(")") && !line.Contains(";") ||

                    line.Contains(" List") && line.Contains("(") && line.Contains(")") && !line.Contains(";")) {

                    method = true;

                    continue;

                }

                if (line.Contains("break")) { Operater++; UniqueOperands[0]=true; line =line= line.Replace("break", " ");  }

                if (line.Contains("case")) { Operater++; UniqueOperands[1] = true; line = line.Replace("case", " "); }

                if (line.Contains("continue")) { Operater++; UniqueOperands[2] = true; line = line.Replace("continue", " "); }

                if (line.Contains("default")) { Operater++; UniqueOperands[3] = true; line = line.Replace("default", " "); }

                if (line.Contains("if")) { Operater++; UniqueOperands[4] = true; line = line.Replace("if", " "); }

                if (line.Contains("else")) { Operater++; UniqueOperands[5] = true; line = line.Replace("else", " "); }

                if (line.Contains("for")) {

                    Operater++; UniqueOperands[6] = true; line = line.Replace("for", " "); }

                if (line.Contains("goto")) { Operater++; UniqueOperands[7] = true; line = line.Replace("goto", " "); }

                if (line.Contains("new")) { Operater++; UniqueOperands[8] = true; line = line.Replace("new", " "); }

                if (line.Contains("return")) { Operater++; UniqueOperands[9] = true; line = line.Replace("return", " "); }

                if (line.Contains("operator")) { Operater++; UniqueOperands[10] = true; line = line.Replace("operator", " "); }

                if (line.Contains("private")) { Operater++; UniqueOperands[11] = true; line = line.Replace("private", " "); }

                if (line.Contains("protected")) { Operater++; UniqueOperands[12] = true; line = line.Replace("protected", " "); }

                if (line.Contains("public")) { Operater++; UniqueOperands[13] = true; line = line.Replace("public", " "); }

                if (line.Contains("protected")) { Operater++; UniqueOperands[14] = true; line = line.Replace("protected", " "); }

                if (line.Contains("sizeof")) { Operater++; UniqueOperands[15] = true; line = line.Replace("sizeof", " "); }

                if (line.Contains("struct")) { Operater++; UniqueOperands[16] = true; line = line.Replace("switch", " "); }

                if (line.Contains("switch")) { Operater++; UniqueOperands[17] = true; line = line.Replace("switch", " "); }

                if (line.Contains("union")) { Operater++; UniqueOperands[18] = true; line = line.Replace("union", " "); }

                if (line.Contains("while")) { Operater++; UniqueOperands[19] = true; line = line.Replace("while", " "); }

                if (line.Contains("this")) { Operater++; UniqueOperands[20] = true; line = line.Replace("this", " "); }

                if (line.Contains("namespace")) { Operater++; UniqueOperands[21] = true; line = line.Replace("namespace", " "); }

                if (line.Contains("using")) { Operater++; UniqueOperands[22] = true; line = line.Replace("using", " "); }

                if (line.Contains("try")) { Operater++; UniqueOperands[23] = true; line = line.Replace("try", " "); }

                if (line.Contains("catch")) { Operater++; UniqueOperands[24] = true; line = line.Replace("catch", " "); }

                if (line.Contains("throw")) { Operater++; UniqueOperands[25] = true; line = line.Replace("throw", " ");}

                if (line.Contains("throws")) { Operater++; UniqueOperands[26] = true; line = line.Replace("throws", " "); }

                if (line.Contains("finally")) { Operater++; UniqueOperands[27] = true; line = line.Replace("finally", " "); }

                if (line.Contains("strictfp")) { Operater++; UniqueOperands[28] = true; line = line.Replace("strictfp", " "); }

                if (line.Contains("instanceof")) { Operater++; UniqueOperands[29] = true; line = line.Replace("instanceof", " "); }

                if (line.Contains("interface")) { Operater++; UniqueOperands[30] = true; line = line.Replace("interface", " "); }

                if (line.Contains("extends")) { Operater++; UniqueOperands[31] = true; line = line.Replace("extends", " "); }

                if (line.Contains("implements")) { Operater++; UniqueOperands[32] = true; line = line.Replace("implements", " "); }

                if (line.Contains("abstract")) { Operater++; UniqueOperands[33] = true; line = line.Replace("abstract", " "); }

                if (line.Contains("concrete")) { Operater++; UniqueOperands[34] = true; line = line.Replace("concrete", " "); }

                if (line.Contains("const\_cast")) { Operater++; UniqueOperands[35] = true; line = line.Replace("const\_cast", " "); }

                if (line.Contains("static\_cast")) { Operater++; UniqueOperands[36] = true; line = line.Replace("static\_cast", " "); }

                if (line.Contains("dynamic\_cast")) { Operater++; UniqueOperands[37] = true; line = line.Replace("dynamic\_cast", " "); }

                if (line.Contains("reinterpret\_cast")) { Operater++; UniqueOperands[38] = true; line = line.Replace("reinterpret\_cast", " "); }

                if (line.Contains("typeid")) { Operater++; UniqueOperands[39] = true; line = line.Replace("break", " "); }

                if (line.Contains("explicit")) { Operater++; UniqueOperands[40] = true; line = line.Replace("typeid", " "); }

                if (line.Contains("true")) { Operater++; UniqueOperands[41] = true; line = line.Replace("true", " "); }

                if (line.Contains("false")) { Operater++; UniqueOperands[42] = true; line = line.Replace("false", " "); }

                if (line.Contains("typename")) { Operater++; UniqueOperands[43] = true; line = line.Replace("typename", " "); }

                if (line.Contains("explicit")) { Operater++; UniqueOperands[44] = true; line = line.Replace("explicit", " "); }

                if (line.Contains("auto")) { Operater++; UniqueOperands[45] = true; line = line.Replace("auto", " "); }

                if (line.Contains("extern")) { Operater++; UniqueOperands[46] = true; line = line.Replace("extern", " "); }

                if (line.Contains("register")) { Operater++; UniqueOperands[47] = true; line = line.Replace("register", " "); }

                if (line.Contains("static")) { Operater++; UniqueOperands[48] = true; line = line.Replace("static", " "); }

                if (line.Contains("typedef")) { Operater++; UniqueOperands[49] = true; line = line.Replace("typedef", " "); }

                if (line.Contains("virtual")) { Operater++; UniqueOperands[50] = true; line = line.Replace("virtual", " "); }

                if (line.Contains("mutable")) { Operater++; UniqueOperands[51] = true; line = line.Replace("mutable", " "); }

                if (line.Contains("inline")) { Operater++; UniqueOperands[52] = true; line = line.Replace("inline", " "); }

                if (line.Contains("const")) { Operater++; UniqueOperands[53] = true; line = line.Replace("const", " "); }

                if (line.Contains("friend")) { Operater++; UniqueOperands[54] = true; line = line.Replace("friend", " "); }

                if (line.Contains("volatile")) { Operater++; UniqueOperands[55] = true; line = line.Replace("volatile", " "); }

                if (line.Contains("transient")) { Operater++; UniqueOperands[56] = true; line = line.Replace("transient", " "); }

                if (line.Contains("final")) { Operater++; UniqueOperands[57] = true; line = line.Replace("final", " "); }

                if (line.Contains("!=")) { Operater++; UniqueOperands[58] = true; line = line.Replace("break", " "); }

                if (line.Contains("%")) { Operater++; UniqueOperands[59] = true; line = line.Replace("%", ""); }

                if (line.Contains("%=")) { Operater++; UniqueOperands[60] = true; line = line.Replace("%=", " "); }

                if (line.Contains("&")) { Operater++; UniqueOperands[61] = true; line = line.Replace("&", " "); }

                if (line.Contains("&&")) { Operater++; UniqueOperands[62] = true; line = line.Replace("&&", " "); }

                if (line.Contains("||")) { Operater++; UniqueOperands[63] = true; line = line.Replace("||", " "); }

                if (line.Contains("(")) { Operater++; UniqueOperands[64] = true; line = line.Replace("(", " "); }

                if (line.Contains(")")) { Operater++; UniqueOperands[65] = true; line = line.Replace(")", " "); }

                if (line.Contains("{")) { Operater++; UniqueOperands[66] = true; line = line.Replace("{", " "); }

                if (line.Contains("}")) {

                    if (method)

                    {

                        line = line.Replace("}", " ");

                        method = false;

                    }

                    else

                    {

                        Operater++; UniqueOperands[67] = true; line = line.Replace("}", " ");

                    }

                                        }

                if (line.Contains("[")) { Operater++; UniqueOperands[68] = true; line = line.Replace("[", " "); }

                if (line.Contains("]")) { Operater++; UniqueOperands[69] = true; line = line.Replace("]", " "); }

                if (line.Contains("\*=")) { Operater++; UniqueOperands[71] = true; line = line.Replace("\*=", " "); }

                if (line.Contains("\*")) { Operater++; UniqueOperands[70] = true; line = line.Replace("\*", " "); }

                if (line.Contains("++")) { Operater++; UniqueOperands[73] = true; line = line.Replace("++", " "); }

                if (line.Contains("+=")) { Operater++; UniqueOperands[74] = true; line = line.Replace("+=", " "); }

                if (line.Contains("+")) { Operater++; UniqueOperands[72] = true; line = line.Replace("+", " "); }

                if (line.Contains(",")) { Operater++; UniqueOperands[75] = true; line = line.Replace(",", " "); }

                if (line.Contains("--")) { Operater++; UniqueOperands[77] = true; line = line.Replace("--", " "); }

                if (line.Contains("-")) { Operater++; UniqueOperands[76] = true; line = line.Replace("-", " "); }

                if (line.Contains("-=->")) { Operater++; UniqueOperands[78] = true; line = line.Replace("-=->", " "); }

                if (line.Contains(".")) { Operater++; UniqueOperands[79] = true; line = line.Replace(".", " "); }

                if (line.Contains("...")) { Operater++; UniqueOperands[80] = true; line = line.Replace("...", " "); }

                if (line.Contains("/")) { Operater++; UniqueOperands[81] = true; line = line.Replace("/", " "); }

                if (line.Contains("/=")) { Operater++; UniqueOperands[82] = true; line = line.Replace("/=", " "); }

                if (line.Contains("::")) { Operater++; UniqueOperands[83] = true; line = line.Replace("::", " "); }

                else if (line.Contains(":")) { Operater++; UniqueOperands[84] = true; line = line.Replace(":", " "); }

                if (line.Contains("<<=")) { Operater++; UniqueOperands[87] = true; line = line.Replace("<<=", " "); }

                if (line.Contains("<<")) { Operater++; UniqueOperands[86] = true; line = line.Replace("<<", " "); }

                if (line.Contains("<=")) { Operater++; UniqueOperands[88] = true; line = line.Replace("<=", " "); }

                if (line.Contains("<")) { Operater++; UniqueOperands[85] = true; line = line.Replace("<", " "); }

                if (line.Contains("==")) { Operater++; UniqueOperands[89] = true; line = line.Replace("==", " "); }

                else   if (line.Contains("=") &&

                            !line.Contains("<=") &&

                            !line.Contains(">=") &&

                            !line.Contains("!=") &&

                            !line.Contains(">>>=") &&

                            !line.Contains(">>=") &&

                            !line.Contains("<<<=") &&

                            !line.Contains("<=") &&

                            !line.Contains("<<=") &&

                            !line.Contains("==")&&

                            !line.Contains("-=->") &&

                            !line.Contains("\*=") &&

                            !line.Contains("/=") &&

                            !line.Contains(">>=>>>=") &&

                            !line.Contains("^=") &&

                            !line.Contains("|=") &&

                            !line.Contains("=&"))

                            { Operater++; UniqueOperands[90] = true; line = line.Replace("=", " "); }

                if (line.Contains(">>=>>>=")) { Operater++; UniqueOperands[95] = true; line = line.Replace(">>=>>>=", " "); }

                if (line.Contains(">>>")) { Operater++; UniqueOperands[94] = true; line = line.Replace(">>>", " "); }

                if (line.Contains(">>")) { Operater++; UniqueOperands[93] = true; line = line.Replace(">>", " "); }

                if (line.Contains(">=")) { Operater++; UniqueOperands[92] = true; line = line.Replace(">=", " "); }

                if (line.Contains(">")) { Operater++; UniqueOperands[91] = true; line = line.Replace(">", " "); }

                if (line.Contains("?")) { Operater++; UniqueOperands[96] = true; line = line.Replace("?", " "); }

                if (line.Contains("^=")) { Operater++; UniqueOperands[98] = true; line = line.Replace("^=", " "); }

                if (line.Contains("^")) { Operater++; UniqueOperands[97] = true; line = line.Replace("^", " "); }

                if (line.Contains("|=")) { Operater++; UniqueOperands[100] = true; line = line.Replace("|=", " "); }

                if (line.Contains("|")) { Operater++; UniqueOperands[99] = true; line = line.Replace("|", " "); }

                if (line.Contains("~")) { Operater++; UniqueOperands[101] = true; line = line.Replace("~", " "); }

                if (line.Contains(";")) { Operater++; UniqueOperands[102] = true; line = line.Replace(";", " "); }

                if (line.Contains("“")) { Operater++; UniqueOperands[104] = true; line = line.Replace("“", " "); }

                if (line.Contains("'")) { Operater++; UniqueOperands[105] = true; line =line.Replace("'", " "); }

                if (line.Contains("=&")) { Operater++; UniqueOperands[103] = true; line = line.Replace("=&", " "); }

                if (line.Contains("int")) { Operater++; UniqueOperands[106] = true;line= line.Replace("int", " "); }

                if (line.Contains("float")) { Operater++; UniqueOperands[107] = true; line = line.Replace("float", " "); }

                if (line.Contains("double")) { Operater++; UniqueOperands[108] = true; line = line.Replace("double", " "); }

                if (line.Contains("char")) { Operater++; UniqueOperands[109] = true; line = line.Replace("char", " "); }

                if (line.Contains("string")) { Operater++; UniqueOperands[110] = true; line = line.Replace("string", " "); }

                else   if (line.Contains("String")) { Operater++; UniqueOperands[111] = true; line = line.Replace("String", " "); }

                if (line.Contains("boolean")) { Operater++; UniqueOperands[112] = true; line = line.Replace("boolean", " "); }

                else if (line.Contains("Boolean")) { Operater++; UniqueOperands[113] = true; line = line.Replace("Boolean", " "); }

                l.Add( line);

            }

            foreach (var x in l)

            {

                string[] st = x.Split(' ');

                foreach(string s in st)

                {

                    if(!string.IsNullOrEmpty(s)&&!s.Equals("\t"))

                    Operads.Add(s);

                }

            }

            var g = Operads.Select(k => k).Distinct();

            Operand = Operads.Count();

            n2 = g.Count();

            for (int j = 0; j < UniqueOperands.Length; j++)

            {

                if (UniqueOperands[j])

                {

                      n1++;

                }

            }

            return Tuple.Create(Operater,n1,Operand,n2 );

        }

        private static List<string> doPreProcessing(string Path)

        {

            System.IO.StreamReader str = new System.IO.StreamReader(@"" + Path);

            string line = "";

            List<string> List = new List<string>();

            while ((line = str.ReadLine()) != null)

            {

                if (line.Contains("/\*") && line.Contains("\*/"))

                {

                    var c = line.Split(new string[] { "/\*" }, StringSplitOptions.None);

                    line = c[0];

                }

                if (line.Contains("&&") || line.Contains("|") || line.Contains("||") || line.Contains("&") || line.Contains(";") || line.Contains(","))

                {

                    line = line.Replace("||", "@@");

                    line = line.Replace("|", "@");

                    var x = Regex.Split(line, "(@@|&&|&|@|;|,)");

                    string z = "";

                    foreach (var y in x)

                    {

                        z = y;

                        if (y.Contains("@@"))

                            z = y.Replace("@@", "||");

                        if (y.Contains("@"))

                            z = y.Replace("@", "|");

                        string[] xy = z.Split(new string[] { "//" }, StringSplitOptions.None);

                        z = xy[0];

                        List.Add(z);

                    }

                }

                else

                {

                    string[] m;

                    Boolean BComment = false;

                    if (line.Contains("/\*") && line.Contains("\*/"))

                    {

                        var x1 = line.Split(new string[] { "/\*" }, StringSplitOptions.None);

                        var x2 = line.Split(new string[] { "\*/" }, StringSplitOptions.None);

                        List.Add(x1[0]);

                        List.Add(x2[1]);

                    }

                    if (line.Contains("/\*") && !line.Contains("\*/"))

                    {

                        m = line.Split(new string[] { "/\*" }, StringSplitOptions.None);

                        List.Add(m[0]);

                        while (!BComment)

                        {

                            var l = str.ReadLine();

                            if (l.Contains("\*/"))

                            {

                                var za = l.Split(new string[] { "\*/" }, StringSplitOptions.None);

                                List.Add(za[1]);

                                BComment = true;

                            }

                        }

                    }

                    else

                    {

                        string[] xy = line.Split(new string[] { "//" }, StringSplitOptions.None);

                        line = xy[0];

                        List.Add(line);

                    }

                }

            }

            foreach (string li in List)

            {

                Console.WriteLine(li);

            }

            tempCount++;

            return List;

        }

        private void label2\_Click(object sender, EventArgs e)

        {

        }

        private void label3\_Click(object sender, EventArgs e)

        {

        }

        private void dataGridView2\_CellContentClick(object sender, DataGridViewCellEventArgs e)

        {

        }

        private void label23\_Click(object sender, EventArgs e)

        {

        }

        public static async

        Task<Github>

        getGitInfo(string username, string reponame)

        {

            Github git = await RequestGitInfo(username, reponame);

            return git;

        }

        public static async Task<Github> RequestGitInfo(string username, string reponame)

        {

            var client = new GitHubClient(new Octokit.ProductHeaderValue("abc"));

            var basicAuth = new Credentials("cicada33012018", "area522018"); // NOTE: not real credentials

            client.Credentials = basicAuth;

            var Commits = await client

                    .Repository.Commit.GetAll(username, reponame);

            var Contributors = await client

                    .Repository.GetAllContributors(username, reponame);

            return new Github(Contributors, Commits);

        }

        private void textBox1\_TextChanged(object sender, EventArgs e)

        {

        }

        public  async void button3\_Click(object sender, EventArgs e)

        {

            var x = textBox1.Text;

            var y = textBox2.Text;

            try

            {

                var GitInfo = await getGitInfo(x, y);

                panel9.Visible = true;

                panel9.Enabled = true;

                label25.Text = GitInfo.Contributors1.Count.ToString();

                label29.Text = GitInfo.Commits1.Count.ToString();

                label56.Text = GitInfo.Contributors1.Count.ToString();

                label54.Text = GitInfo.Commits1.Count.ToString();

                DataTable dt = new DataTable();

                dt.Columns.Add("Contributor");

                dt.Columns.Add("Commits");

                foreach (var f in GitInfo.Contributors1)

                {

                    DataRow dr = dt.NewRow();

                    dr[0] = "" + f.Login;

                    dr[1] = "" + f.Contributions;

                    chart4.Series["Series1"].Points.AddXY(dr[0],dr[1]);

                    dt.Rows.Add(dr);

                }

                dataGridView3.DataSource = dt;

                dataGridView5.DataSource = dt;

                dataGridView3.Columns[0].Width = 170;// The id column

                dataGridView3.Columns[1].Width = 70;

                dataGridView5.Columns[0].Width = 170;// The id column

                dataGridView5.Columns[1].Width = 70;

            }

            catch(Exception ex)

            {

                MessageBox.Show(ex.Message);

            }

        }

        private void checkBox1\_CheckedChanged(object sender, EventArgs e)

        {

            if (checkBox1.Checked)

            {

                label30.Enabled = true;

                label31.Enabled = true;

                button3.Enabled = true;

                textBox2.Enabled = true;

                textBox1.Enabled = true;

                panel9.Enabled = true;

            }

            else

            {

                label30.Enabled = false;

                label31.Enabled = false;

                button3.Enabled = false;

                textBox2.Enabled = false;

                textBox1.Enabled = false;

                panel9.Enabled = false;

            }

        }

        private void button2\_Click\_1(object sender, EventArgs e)

        {

            foreach (var series in chart2.Series)

            {

                series.Points.Clear();

            }

            foreach (var series in chart1.Series)

            {

                series.Points.Clear();

            }

            foreach (var series in chart3.Series)

            {

                series.Points.Clear();

            }

            //Initialization of all Metrices

            // Cyclometric Complexity

            Tuple<string[], int[], int> tpl;

            List<string> methodName = new List<string>();

            List<int> methodCompelxity = new List<int>();

            int  noOfMethod = 0 ;

            //Halstead

            Tuple<int, int, int, int> H\_M1=null;

            Tuple < double, double, double, double, double> H\_M2 = null;

            int Operators = 0, Operands = 0, UniqueOperators = 0, UniqueOperands = 0;

            double ProgramLenght=0, SizeOfVocabulary = 0, ProgramVolume = 0, DifficultyLevel = 0, ProgramLevel = 0;

            // LOC

            int loc = 0, sloc = 0, bloc = 0, ncloc = 0, cloc=0;

            Tuple<int, int, int, int, int> LOC=null;

            /\*\*\*\*\*\*\*\*\* Cyclometric Complexity \*\*\*\*\*\*\*\*/

            List<string>  fileList = getFileList(selectedFolder);

            foreach (var file in fileList)

            {

                tpl = CalculateComplexity(file);

                methodName.AddRange(tpl.Item1.Where(x=>!string.IsNullOrEmpty(x)));

                methodCompelxity.AddRange(tpl.Item2.Where( x=> x>0));

                noOfMethod += tpl.Item3;

                H\_M1 = CalculateHalstead1(file);

                Operators += H\_M1.Item1;

                Operands += H\_M1.Item3;

                UniqueOperators += H\_M1.Item2;

                UniqueOperands += H\_M1.Item4;

                H\_M2 = CalculateHalstead2(H\_M1.Item1, H\_M1.Item2, H\_M1.Item3, H\_M1.Item4);

                ProgramLenght += H\_M2.Item1;

                SizeOfVocabulary += H\_M2.Item2;

                ProgramVolume += H\_M2.Item3;

                DifficultyLevel += H\_M2.Item4;

                ProgramLevel += H\_M2.Item5;

                LOC= CalculateLOC(file);

                loc += LOC.Item1;

                cloc += LOC.Item2;

                bloc += LOC.Item3;

                sloc += LOC.Item4;

                ncloc += LOC.Item5;

            }

            DataTable dt = new DataTable();

            dt.Columns.Add("Methods");

            dt.Columns.Add("Complexity");

            dt.Columns.Add("ComplexityLVL");

            for (int i = 1; i < methodName.Count; i++)

            {

              DataRow dr = dt.NewRow();

                dr[0] = methodName.ElementAt(i);

                if (methodCompelxity.ElementAt(i) > 0)

                {

                    dr[1] = methodCompelxity.ElementAt(i);

                    int val = methodCompelxity.ElementAt(i);

                    if (val < 5)

                        dr[2] = "Good";

                    else if (val >= 5 && val <= 10)

                        dr[2] = "OK";

                    else

                        dr[2] = "Too Complex";

                }

             if (!string.IsNullOrEmpty(dr[0].ToString()))

                  chart2.Series["s1"].Points.AddXY("" + dr[0], "" + dr[1]);

                dt.Rows.Add(dr);

            }

            label23.Text = ""+ noOfMethod;

            label20.Text = ""+ noOfMethod;

            dataGridView2.DataSource = dt;

            dataGridView4.DataSource = dt;

            /\*\*\*\*\*\*\*\*\* Halstead Metrics \*\*\*\*\*\*\*\*/

            label11.Text = "" + Operators;

            label12.Text = "" + Operands;

            label13.Text = "" + UniqueOperators;

            label14.Text = "" + UniqueOperands;

            label15.Text = "" + ProgramLenght;

            label16.Text = "" + SizeOfVocabulary;

            label17.Text = "" + ProgramVolume.ToString("#####.##");

            label18.Text = "" + DifficultyLevel;

            label19.Text = "" + ProgramLevel.ToString("#####.##");

            label41.Text = "" + Operators;

            label40.Text = "" + Operands;

            label39.Text = "" + UniqueOperators;

            label37.Text = "" + UniqueOperands;

            label36.Text = "" + ProgramLenght;

            label35.Text = "" + SizeOfVocabulary;

            label34.Text = "" + ProgramVolume.ToString("#####.##");

            label33.Text = "" + DifficultyLevel;

            label32.Text = "" + ProgramLevel.ToString("#####.##");

            chart1.Series["Series1"].Points.AddXY("Operators", Operators);

            chart1.Series["Series1"].Points.AddXY("Operands", Operands);

            chart1.Series["Series1"].Points.AddXY("Unique Operators", UniqueOperators);

            chart1.Series["Series1"].Points.AddXY("Unique Operands", UniqueOperands);

            chart3.Series["Series1"].Points.AddXY("Program Lenght", ProgramLenght);

            chart3.Series["Series1"].Points.AddXY("Size of Vocabulary", SizeOfVocabulary);

            chart3.Series["Series1"].Points.AddXY("Program Volume ", ProgramVolume);

            try {

            progressBar1.Minimum = 0;

            progressBar1.Maximum = 300;

            progressBar1.Value = (int)DifficultyLevel;

            progressBar2.Minimum = 0;

            progressBar2.Maximum = 100;

            progressBar2.Value = (int)(ProgramLevel \* 100);

            }

            catch(Exception ex)

            {

                MessageBox.Show(ex.Message);

            }

            /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*     LOC     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

            label68.Text = "" + loc;

            label62.Text = "" + cloc;

            label61.Text = "" + bloc;

            label60.Text = "" + sloc;

            label59.Text = "" + ncloc;

            label93.Text = "" + loc;

            label85.Text = "" + loc;

            label84.Text = "" + cloc;

            label83.Text = "" + bloc;

            label82.Text = "" + sloc;

            label81.Text = "" + ncloc;

            progressBar3.Maximum = loc;

            progressBar4.Maximum = loc;

            progressBar5.Maximum = loc;

            progressBar6.Maximum = loc;

            progressBar3.Minimum = 0;

            progressBar4.Minimum = 0;

            progressBar5.Minimum = 0;

            progressBar6.Minimum = 0;

            try

            {

                progressBar3.Value = cloc;

                progressBar4.Value = bloc;

                progressBar5.Value = sloc;

                progressBar6.Value = ncloc;

            }

            catch (Exception ex)

            {

                MessageBox.Show("" + ex.Message);

            }

        }

        private void chart2\_Click(object sender, EventArgs e)

        {

        }

        private void chart4\_Click(object sender, EventArgs e)

        {

        }

        private void dataGridView2\_CellContentClick\_1(object sender, DataGridViewCellEventArgs e)

        {

        }

        private void dataGridView1\_CellClick(object sender, DataGridViewCellEventArgs e)

        {

            foreach (var series in chart2.Series)

            {

                series.Points.Clear();

            }

            foreach (var series in chart1.Series)

            {

                series.Points.Clear();

            }

            foreach (var series in chart3.Series)

            {

                series.Points.Clear();

            }

            var FileName = dataGridView1.SelectedCells[0].Value;

            var FileDirectory = dataGridView1.SelectedCells[1].Value;

            string FilePath = FileDirectory + "/" + FileName;

            int z = 0;

            Tuple<string[], int[], int> tpl = CalculateComplexity(FilePath);

            DataTable dt = new DataTable();

            dt.Columns.Add("Methods");

            dt.Columns.Add("Complexity");

            dt.Columns.Add("ComplexityLVL");

            for (int i = 1; i < tpl.Item1.Length; i++)

            {

                DataRow dr = dt.NewRow();

                dr[0] = tpl.Item1[i];

                if (tpl.Item2[i] > 0)

                {

                    dr[1] = tpl.Item2[i];

                    int val = tpl.Item2[i];

                    if (val < 5)

                        dr[2] = "Good";

                    else if (val >= 5 && val <= 10)

                        dr[2] = "OK";

                    else

                        dr[2] = "Too Complex";

                }

                if (!string.IsNullOrEmpty(dr[0].ToString()))

                    chart2.Series["s1"].Points.AddXY("" + dr[0], "" + dr[1]);

                dt.Rows.Add(dr);

            }

            Tuple<int,int,int,int,int> LOC = CalculateLOC(FilePath);

            label68.Text = "" + LOC.Item1;

            label62.Text = "" + LOC.Item2;

            label61.Text = "" + LOC.Item3;

            label60.Text = "" + LOC.Item4;

            label59.Text = "" + LOC.Item5;

            label93.Text = "" + LOC.Item1;

            label85.Text = "" + LOC.Item1;

            label84.Text = "" + LOC.Item2;

            label83.Text = "" + LOC.Item3;

            label82.Text = "" + LOC.Item4;

            label81.Text = "" + LOC.Item5;

            progressBar3.Maximum = LOC.Item1;

            progressBar4.Maximum = LOC.Item1;

            progressBar5.Maximum = LOC.Item1;

            progressBar6.Maximum = LOC.Item1;

            progressBar3.Minimum = 0;

            progressBar4.Minimum = 0;

            progressBar5.Minimum = 0;

            progressBar6.Minimum = 0;

            try {

            progressBar3.Value = LOC.Item2;

            progressBar4.Value = LOC.Item3;

          progressBar5.Value = LOC.Item4;

            progressBar6.Value = LOC.Item5;

            }

            catch (Exception ex)

            {

                MessageBox.Show(""+ex.Message);

            }

            label23.Text = tpl.Item3.ToString();

            label20.Text = tpl.Item3.ToString();

            dataGridView2.DataSource = dt;

            dataGridView4.DataSource = dt;

            dataGridView2.Columns[0].Width = 130;

            dataGridView2.Columns[1].Width = 60;

            dataGridView2.Columns[2].Width = 90;

            dataGridView4.Columns[0].Width = 130;

            dataGridView4.Columns[1].Width = 50;

            H\_Matrix1 = CalculateHalstead1(FilePath);

            H\_Matrix2 = CalculateHalstead2(H\_Matrix1.Item1, H\_Matrix1.Item2, H\_Matrix1.Item3, H\_Matrix1.Item4);

            label11.Text = "" + H\_Matrix1.Item1;

            label12.Text = "" + H\_Matrix1.Item3;

            label13.Text = "" + H\_Matrix1.Item2;

            label14.Text = "" + H\_Matrix1.Item4;

            label15.Text = "" + H\_Matrix2.Item1;

            label16.Text = "" + H\_Matrix2.Item2;

            label17.Text = "" + H\_Matrix2.Item3.ToString("#####.##");

            label18.Text = "" + H\_Matrix2.Item4;

            label19.Text = "" + H\_Matrix2.Item5.ToString("#####.##");

            label41.Text = "" + H\_Matrix1.Item1;

            label40.Text = "" + H\_Matrix1.Item3;

            label39.Text = "" + H\_Matrix1.Item2;

            label37.Text = "" + H\_Matrix1.Item4;

            label36.Text = "" + H\_Matrix2.Item1;

            label35.Text = "" + H\_Matrix2.Item2;

            label34.Text = "" + H\_Matrix2.Item3.ToString("#####.##");

            label33.Text = "" + H\_Matrix2.Item4;

            label32.Text = "" + H\_Matrix2.Item5.ToString("#####.##");

            chart1.Series["Series1"].Points.AddXY("Operators", H\_Matrix1.Item1);

            chart1.Series["Series1"].Points.AddXY("Operands", H\_Matrix1.Item3);

            chart1.Series["Series1"].Points.AddXY("Unique Operators", H\_Matrix1.Item2);

            chart1.Series["Series1"].Points.AddXY("Unique Operands", H\_Matrix1.Item4);

            chart3.Series["Series1"].Points.AddXY("Program Lenght", H\_Matrix2.Item1);

            chart3.Series["Series1"].Points.AddXY("Size of Vocabulary", H\_Matrix2.Item2);

            chart3.Series["Series1"].Points.AddXY("Program Volume ", H\_Matrix2.Item3);

            progressBar1.Minimum = 0;

            progressBar1.Maximum = 100;

            progressBar1.Value = (int) H\_Matrix2.Item4;

            progressBar2.Minimum = 0;

            progressBar2.Maximum = 100;

            progressBar2.Value = (int)(H\_Matrix2.Item5\*100);

        }

        private Tuple<int, int, int, int, int> CalculateLOC(string path)

        {

                  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*       LOC     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

            using (StreamReader sr = File.OpenText(path))

            {

                string t = "";

                while ((t = sr.ReadLine()) != null)

                {

                    loc++;

                    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*     BLOC    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                    abc = t.Length;

                    if (abc < 1)

                    {

                        bloc++;

                    }

                    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      SLOC      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                    if (t.Contains('/'))

                    {

                        string vv = t;

                        tc = vv.Split('/');

                        int bb = tc[0].Length;

                        string tv = tc[0];

                        if (bb > 0)

                        {

                            bd = bb - 1;

                        }

                        char st = '\*';

                        st.ToString();

                        try

                        {

                            if (tc[0] != null && tc[0].Length > 0 && !tv[bd].Equals('\*'))

                            {

                                tsloc++;

                            }

                        }

                        catch (Exception ex)

                        {

                            MessageBox.Show(ex.Message);

                        }

                    }

                    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      CLOC       \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                    foreach (var c in t)

                    {

                        e++;

                        int ff = cloc;

                        if (c == ('/') && star == 'e')

                        {

                            star = 'w';

                            check = 'x';

                        }

                        else if (c == ('/') && ex == '/' && star == 'w' && check == 'x' && f == 0)

                        {

                            star = 'r';

                            check = 'y';

                            f = 3;

                        }

                        if (c == ('/') && star == 'y' && check == 'e')

                        {

                            cloc--;

                            star = 'w';

                            check = 'x';

                        }

                        else if (c == ('\*') && ex == '/' && star == 'w' && check == 'x' && f == 0)

                        {

                            f = 2;

                            check = 'y';

                            star = 'r';

                        }

                        else if (c == ('\*') && ex == '\*' && star == 'r' && check == 'y' && f == 0)

                        {

                            f = 2;

                            check = 'y';

                            star = 'r';

                        }

                        else if (c == ('/') && star == 'r' && check == 'y' && f == 2)

                        {

                            star = 'y';

                            check = 'e';

                            f = 0;

                            cloc++;

                        }

                        else if (c == '\*' && ex == '/' && star == 'y' && check == 'e' && f == 0)

                        {

                            cloc--;

                            check = 'y';

                            star = 'r';

                            f = 2;

                        }

                        else if (c == ('\*') && star == 'r' && check == 'y' && f == 2)

                        {

                            f = 2;

                            check = 'y';

                            star = 'r';

                        }

                        else if (c == ('\*') && star == 'w' && check == 'x' && f == 2)

                        {

                            f = 2;

                            check = 'y';

                            star = 'r';

                        }

                        else if (c == ('/') && ex == '\*' && star == 'r' && check == 'y' && f == 2)

                        {

                            f = 0;

                            star = 'k';

                            check = 'g';

                            cloc++;

                        }

                        else if (c == ('\*') && star == 'e' && check == 'z' && f == 2)

                        {

                            f = 2;

                            star = '\*';

                            check = 'y';

                        }

                        else if (c == ('/') && star == '\*' && check == 'y' && f == 2)

                        {

                            f = 0;

                            star = 'k';

                            check = 'g';

                            cloc++;

                        }

                        else if (c == '/' && star == 'k')

                        {

                            f = 0;

                            star = 'e';

                            check = 'g';

                        }

                        else if (c == ('\*') && ex == '\*' && star == 'e' && check == 'g')

                        {

                            f = 2;

                            star = 'g';

                            check = 'e';

                            cloc--;

                        }

                        else if (c == ('/') && ex == '\*' && star == 'e' && check == 'g')

                        {

                            cloc++;

                            star = 'h';

                            check = 'i';

                        }

                        else if (c == '/' || c == '\*')

                        {

                            ex = c;

                        }

                    }

                    e = -1;

                    star = 'e';

                    check = 'z';

                    if (f == 2 && t.Length > 0)

                    {

                        cloc++;

                    }

                    else if (f == 3 && t.Length > 0)

                    {

                        f = 0;

                        cloc++;

                    }

                }

                sloc = tsloc + (loc - (bloc + cloc));

                /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      NCLOC      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                ncloc = tsloc + (loc - cloc);

                return Tuple.Create(loc, cloc, bloc, sloc, ncloc);

            }

        }

        private void tabPage3\_Click(object sender, EventArgs e)

        {

        }

        private void button4\_Click(object sender, EventArgs e)

        {

            tabControl1.Visible = false;

            button1.Enabled = true;

        }

        private void label82\_Click(object sender, EventArgs e)

        {

        }

        private void label55\_Click(object sender, EventArgs e)

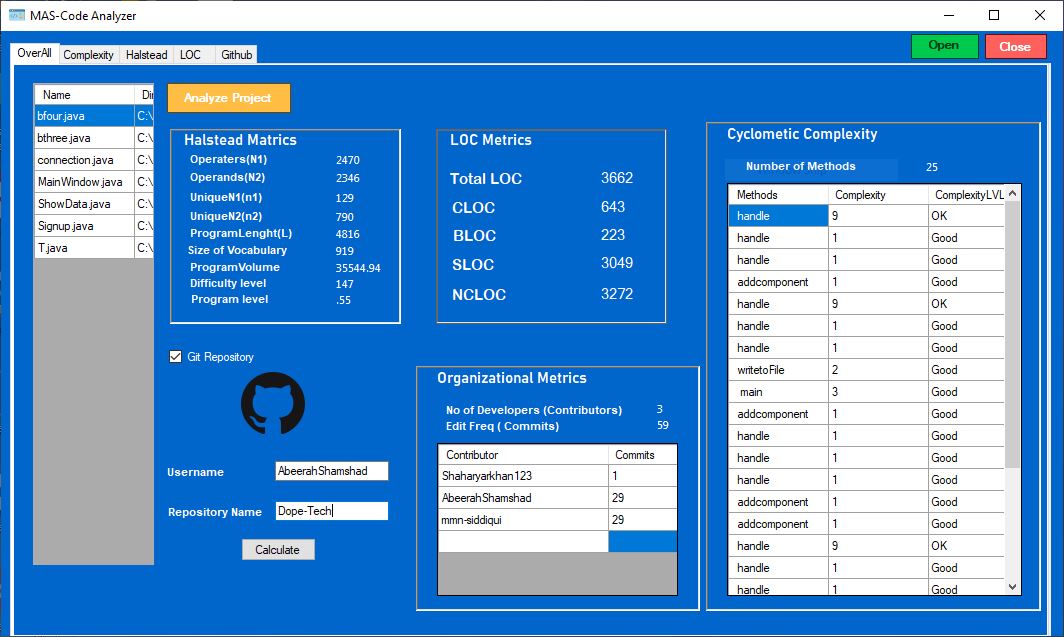
        {

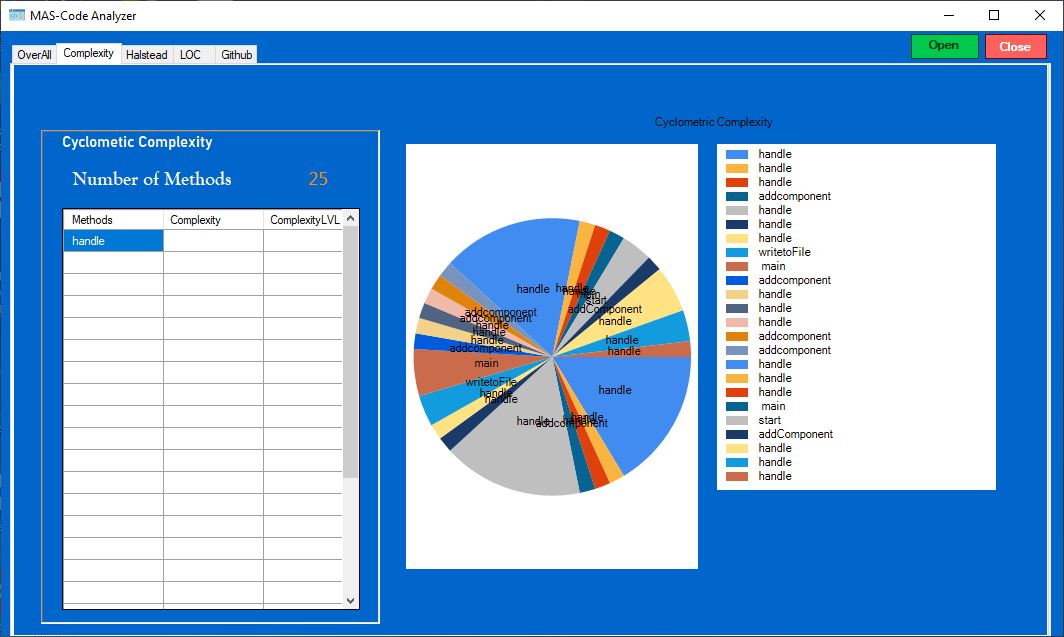
        }

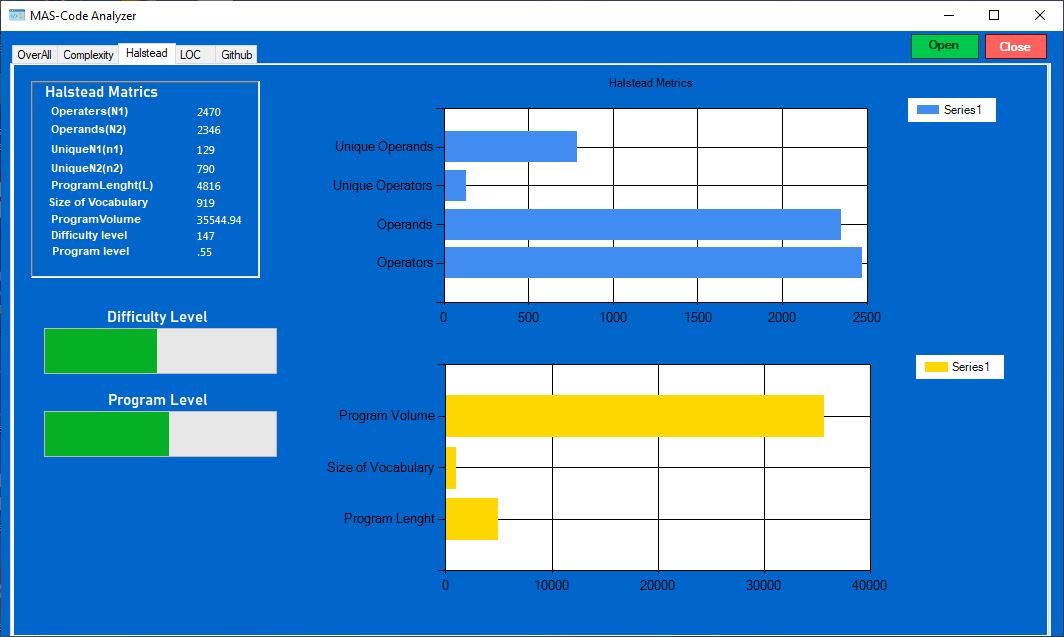
    }

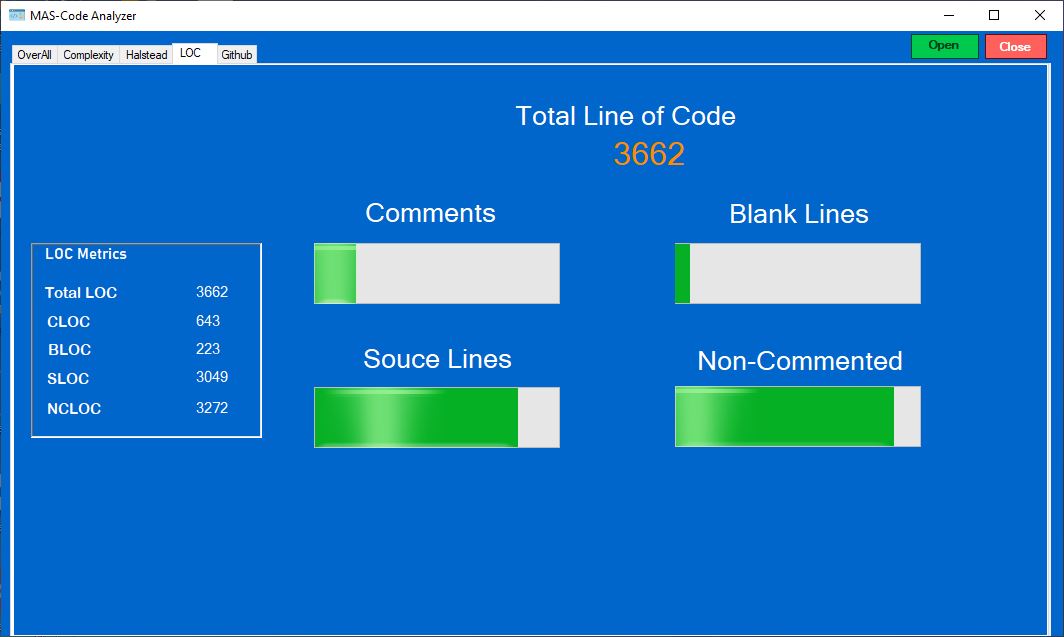
}

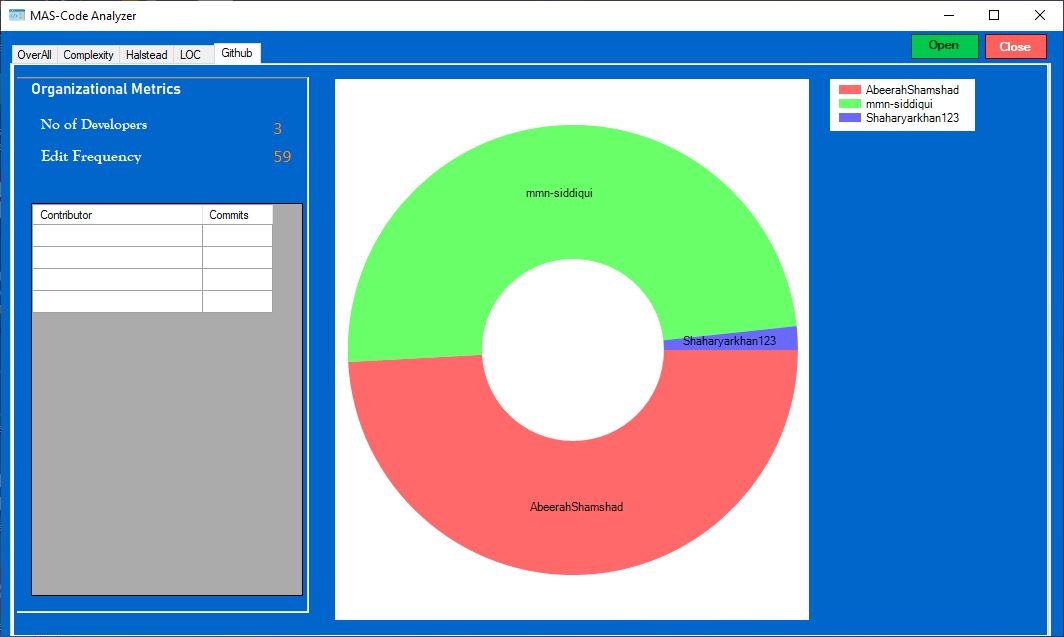
**Output of Project:**

****

****

****

****

****