SOFTWARE METRICS TOOL

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Rajat Singh

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SOFTWARE METRICS TOOL

By

Rajat Singh

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Kenneth Magel
Chair

Anne Denton

Benton Duncan

Approved:

November 27, 2018
Date

Brian Slator
Department Chair

ABSTRACT

In the current world software applications are becoming more important with each passing day. They are present in all walks of life and it's difficult to imagine a world without them. It's a part of every known industry whether it be manufacturing, healthcare, financing. They are available on all forms in personal laptops, mobiles, and tablets.

However, there is another challenging task to figure out the quality of the software. There are multiple measures available in the form of software metrics. The objective of this thesis is to present an extensible software for calculating software metrics. This tool proposed is a web application which calculates metrics and statistics for the source code files provided. This tool also provides an ability to the user to extend the tool by adding a metric to the tool.

ACKNOWLEDGMENTS

This research would not have been possible without the constant support and guidance of Dr. Kenneth Magel. He has been an extremely valuable resource during my entire Masters Degree program.

I would also like to thank my committee members Dr. Anne Denton and Dr. Benton Duncan for their time and support on my research. I am very thankful to the Computer Science Department, especially Jane Dickerson and Carol Huber for their co-operation and help.

Finally, I sincerely thank my parents and family for their unconditional support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Software metrics play an important part in ensuring software developed is of high quality. Software metrics tell us how this particular software matches with the requirements, is there any problem with the design of the software?

In software development one of the important goals is to improve the process and reduce the effort and input needed for software development. Another point would be to ensure that the software lives up to industry standards (1). To ensure this we can use software metrics. With software metrics we can compare how the software has changed over versions or how a change impacts the software. The same can be done with other facets related to software development like software processes and managing a software project.

In the real world we have many software metrics and any software project might need a different combination of software metrics. So how can we select which software metrics to provide in a tool? The idea is to provide a tool which is not too heavy in terms of size and not too difficult to maintain. But that tool would be useful for software development

This tool provides an extensible way to calculate software metrics. This tool has the ability to calculate some of the software metrics like number of executable statements, number of declarations, in-degree, out-degree etc. This tool provides an API which can be used to implement the code to calculate other software metrics. This tool provides the facility so that new code can be included as part of the application or tool. The tool is implemented in C# and uses Roslyn API (.Net Compiler API) to parse the language. The tool (web application) calculates the metrics data for C# code only.

BACKGROUND AND RELATED WORK

Definition and classification of software metric

According to the standard definition we can say that a software metric is a measure of some property of a piece of software or its specifications (2). The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

There can be multiple ways in which software metrics can be defined.

- Software metrics provide a measure of the software produced and the process used for software development (3).
- Software metrics provides a quantitative value in terms of attributes of a software or a software process or software resources (4).
- If we go by standard definition “standard of software Quality Metrics Methodology”, software metrics is a function, with input as the software data, and output is a value which could decide on how the given attribute affect the software (5).

According to ‘software measurement guidebook’ from the university of Carnegie-Mellon function of metrics can be explained by 3 points:

- to have a deep understanding of software development
- to fulfill appropriate management of software
- to improve the software design procedure

Software metrics can be primarily classified into three areas (3)

a) Product metrics
b) Process metrics
c) Project metrics

Product metrics are concerned with product characteristics like the complexity of the project, quality of the project, the size of the project, design features and performance of the project.

Process metrics collect metrics related with the process used in software development. They are used for improving software development processes. Here software development means all the activities pertaining to software development lifecycle e.g. software development, maintenance etc.

Project metrics involve metrics from the project management side. This includes staffing for the project over software lifecycle, cost of the project, scheduling of the project, number of software developers in a software project.

Literature review

SourceMonitor

SourceMonitor is a metrics tool provided by Campwood Software (6). Its executable file is provided in the form of free software for internal use. SourceMonitor is written using C++ programming language and it calculates the software metrics in a single pass through the code. In terms of language support, it supports C, C++, C#, VB.NET, Java, Delphi and Visual Basic. It provides features such as the ability to save metrics in checkpoints so that metrics for different software projects can be compared. It also provides a feature to display metrics in the form of tables and charts using Kiviat diagrams. It allows the user to download the results in the form of XML or comma-separated-value form. It supports metrics like cyclomatic complexity, total number of code files, number of classes, average number of methods per class, and statements per method (7). This software can be operated in windows GUI or inside scripts using XML commands.

One of the limitations of this software is that it only supports close to 12 metrics. Its source code is not available for modification so adding more metrics is not possible. Another limitation is it provides a windows based installer so it needs to be downloaded and installed on every machine before using it.

SLOCCount

SLOCCount is a software for counting physical source line of code (SLOC) in large programs (8). It uses Park's framework (9) with some minor modifications to calculate the physical number of lines (10). It supports languages like C, C++, C#, COBOL, FORTRAN, Haskell, Java, Perl, and Python etc. It supports close to 30 different languages (10). SLOCCount is available in the form prebuilt binary for Unix and Unix like systems. For windows, it can be configured using Cygwin. It also has report generating tools

for collects the metrics generated for programs. It also provides an estimate of development effort and schedule given the lines of code computed using Basic COCOMO model (10).

This software needs to be downloaded and configured on a machine before running it. Also metrics provided are limited to the physical source line of code.

Source Code Metrics

Source Code Metrics is a commercial software which can be used to calculate source code metrics and perform quality analysis on the software code (11). It supports C, ANSI C++, C# and Java source code for calculating metrics and performing an analysis. It can be used to calculate a large number function, class, namespace/package, file, project, and baseline metrics (12). It can provide reports in multiple formats like html, xml, csv, and text. It also supports code quality analysis which can be used to satisfy ISO9001, TickIT, SEI Levels 3 and 4 compliance.

Source code Metrics being a commercial software is neither free nor open source. It provides an evaluation version for students which supports up to 20 source files for metrics calculation. Although it supports a large set of metrics the user cannot modify/update/add metrics as per user requirements.

NDepend

NDepend is a visual studio extension for static code analysis (13). It can be used for .Net managed code. It has features like calculating code metrics for the code, dependency analysis using dependency graphs and dependency matrix (14). It supports comparing two different versions of the compiled code library. It has the ability to query code over LINQ using CQLinq (15). There are around 200 default queries and rules provided and users can add their custom rules too. It provides the capability to combine a set of rules to be checked before sending the code to production, this is called as a quality gate.

This tool is only available for .NET managed code. It is only available as a visual studio extension. The tool is not free but can be used for 14 days on trial. This is a proprietary tool so custom additions are limited to code rules.

SLOC

SLOC is a tool written in coffee script which can be used to calculate metrics related to source lines of code (16). SLOC can be used to calculate software metrics like physical lines, source lines of code, lines with comments , lines with single line comments, line with block line comments and lines with TODO's. It supports large number of languages like C, C++, C#, Go, Groovy, Java, Python, PHP, R etc. It can be installed using either node or bower. It can be used directly in a html file by using the javascript file in the html. It's a command line tool and the results are shown on the command window. There are multiple adapters to use it with Grunt, Gulp, Atom, Jenkins, Istanbbul and Codemetrics. This tool is available under MIT license so it can be used and modified/customized by user.

The limitation of this tool is that it supports few metrics related to lines of code.

McCabe IQ

McCabe IQ is an advanced static analysis and visualization tool. It helps in analyzing the vulnerabilities and complexity in the code. It can be used to visualize the architecture of the software to determine complex areas of code base which could lead to bugs and security vulnerabilities. It calculates a large number of software metrics related to complexity (cyclomatic complexity, actual complexity, module design complexity, essential complexity), data related software metrics (data complexity metric, data reference metric, tested data complexity metric), object oriented software metrics (metrics related to encapsulation, polymorphism, quality) and Halstead software metrics (program length, program volume, programming time) (17). It also has features to track changes in the software over a period of time, compare code to locate redundant code and track data use within code.It supports programming languages like Ada, C, C++, C#, Java, Perl etc.

This tool is intended for enterprise users. It's a commercial software and is not available for adding more features or functionalities by end user.

Semantic Design

Semantic Design provides a variety of tools for static code analysis, source code metrics, source code style check and analyzing source code and architecture (18). As far as source code metrics is considered they provide separate tools for different languages (19).They support languages like C#, Java,

Cobol, VB.Net. For this literature review we will consider the tool for calculating C# source code metrics tool. It provides metrics like lines or source code, lines of comment, number of files, number of methods, Halstead metrics, cyclomatic complexity, max loop depth, max nested conditional depth etc. (20). This tool is based on 'DMS software reengineering toolkit' which is a proprietary tool of Sematic Design. In simple terms DMS can be defined as an extremely generalized compiler with its own parser, semantic analyzers, compiler data structure modification engines (21). In terms of reports it can generate report in the form of text or xml as needed by end user.

It's a commercial tool meant for enterprise users. So it's not available for free to use or to modify/update. It has an evaluation version which can be used to some extent but it limits the results unless a license is purchased. Also being a proprietary software its source code is not available in public domain.

EZ – Metrix

EZ-Metrix is a commercial source code counting tool. It can be used to calculate metrics like total number of lines in the code, comment lines, blank lines, and source code lines. User can add more rules to calculate custom information related to source code lines (22). Its' a web based tool, so end user logs in to his/her account submits code and gets the results. The results can be viewed in text, excel or graph form. It supports more than 85 languages such as C, C++, C#, Java, PHP, and Python etc. It also has ability to provide a comparison between different versions of code. This can be used to calculate source code metrics of modified code between versions. In this tool the source code measurements are done using Physical Source Statements Algorithm (PSS) (22) and comparison measures are performed using Levenshtein-Distance algorithm (23).

This is a commercial software so an evaluation copy can be used for free for a duration of 30 days. The software metrics are limited to metrics related to source code counting metrics. User can add more rules to get different metrics but it's limited to source code related metrics only.

LOCMetrics

LOCMetrics is a freeware tool which can be used to calculate various software metrics (24). It supports calculating software metrics for languages like C#, C++, Java and SQL. It provide the ability to

calculate metrics like total lines of code, blank lines of code, comment lines of code, logical source lines of code, MCCabe VG complexity, lines with both code and comments, number of comment words and physical executable source lines of code. It's available as an installer for windows operating system. It has a GUI interface or it can be run using command line. For the output it shows the result as a histogram or on the command line.

Though it's a freeware tool but the source code is not available. So there is no possibility of updating or customizing the tool. User is limited to the metrics and the implementation provided in the tool by default.

Visual Studio Code Metrics

Visual Studio provides a feature to help in evaluating the software quality during development (25). This feature provides code metrics for solution or projects. The metrics provided by Visual Studio include maintainability index which measure the ease with which the software can be maintained (it lies between 0 and 100), cyclomatic complexity which measures structural complexity, depth of inheritance which measure the number of class definitions that extend up to root class definition , class coupling and lines of code. This feature also provide ability to filter the results. It also allows user to export the results in excel format or copy to clipboard.

Using this feature requires a license for professional or ultimate version of visual studio to use this feature.

Learning from literature review

In the above literature review we have reviewed software metric tools and application that are available to use currently. One thing that we learned from this review is that most of the software metric tools are require installation and configuration on each machine before using them. Most of them are only available for use for 30 days and beyond that require a subscription or a license to use it further. There is also limitation in terms of capability to customize or update the tool as per user requirements.

Proposed tool

The software metrics tool that we are proposing in this research is a web application hosted on a web server. The users are provided a web ui where they can upload there source code files and users

are presented with the code metrics and statistics on the source code provided. Users can also add their own implementations to calculate additional software metrics. This tool is available for free to all the users. Hence these features make this proposed tool more beneficial to end user. It add some new unique feature and is easy to use.

Need for software metrics

The software development process is an abstract concept. How can we analyze it? How can we quantify it? This job is done via software metrics. Software development lifecycle contains of various steps like requirement gathering, designing, coding, testing and maintenance (26). All of these steps can be analyzed and improved with the help of software metrics.

There are a number of reasons for using software metrics for an application

- Software metrics can be used as a decision making tool during software life cycle. To decide when one stage is complete and it's time to start next stage.
- They provide a standard and scientific method for evaluating a software.
- They can be used to forecast the status or health of a project.
- Metrics can be used to measure reliability of a software.
- Metrics can be used to measure software quality and complexity.

In other words as DeMarco stated 'You cannot control what you cannot measure'. To improve the understanding of what is going on during software development process, to control what is happening and improve the processes and products, software metrics are needed.

Benefits gained from using metrics:-

- Better overall understanding of the product, processes and resources.
- Ability to understand the status of product and resources during software development lifecycle.
- It provides with information which helps in driving a project better and in correct direction. It provides with information as to how much the changes impact the product or in case of processes how much impact the changes had.

- Helps in estimation of various software development timelines. By comparing the current metrics data with historic data people can estimate the time required for activity or a phase of software development lifecycle.

How to improve a software development lifecycle:-

We want to improve the software product or process or better manage the software resources. The first step would be to quantitatively identify the current product or process. Next we suggest or implement an alternative which can improve the product or process. Then measure the affect this alternative had on the product or process. If the product or process actually became better we can continue going forward. This approach is also known as Shewart-Deming Cycle (27).We can use the same alternative to improve similar products or services. But for this whole process of improvement one of the essential condition is to quantitatively measure the attributes of a product or process or usage of software resources. This can be done by software metrics. So software metrics plays an important part in improving software development lifecycle.

Roslyn API

For most of the programmers compilers are kind of a black box. Programmers write some code and then compile it using the compiler. Compiler does some magic at its end and programmer gets the object files or assemblies as the output. Programmer has no idea what happens inside the compiler. We have a number of use cases where this information will either improve the performance of existing tools or simplify their implementation by a large margin. Some of the examples of such use cases would be Intellisense, refactoring and intelligent renaming in case of integrated development environments.

Roslyn API exposes the information about our code that compilers have. The .NET Compiler Platform (“Roslyn”) exposes the C# and Visual Basic compiler’s code analysis to you as a consumer by providing an API layer that mirrors a traditional compiler pipeline (28).

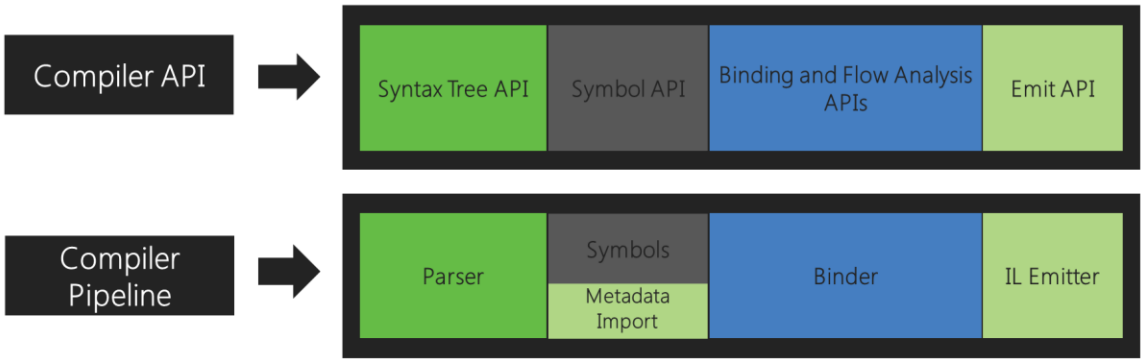

Figure 1 – Roslyn compiler - phases (28)

Corresponding to each of the compiler phases, an object model is surfaced that allows access to the information at that phase. The parsing phase is exposed using a syntax tree, the declaration phase as a hierarchical symbol table and binding phase as a model that exposes the result of semantic analysis. Some of the Roslyn API components are –

Semantic model

A semantic model contains all the semantic information of a single source file. It contains following information

- How variable flow in and out of regions of source.
- All diagnostics, which are errors and warnings.
- The resultant type of any expression

Syntax tree

Syntax tree is literally a tree data structure, where non terminal structural elements parent other elements. It has three key attributes

- It holds all the source information.
- From any syntax node it is possible to get the text representation of the sub tree rooted at that node.
- Syntax trees are immutable and thread safe

Each syntax tree is primarily made up of

- Syntax nodes -They represent syntactic constructs such as declarations, clauses and expressions.

- Syntax tokens – These are the terminals of the language grammar, representing the smallest fragments of the code. They consist of keywords, identifiers, literals and punctuation.
- Syntax Trivia – These consist part of source text like whitespace, comments and preprocessor directives.

CompilationUnitSyntax

CompilationUnitSyntax is the parent of all the syntax nodes inside a source file. In other words we can say that if we convert any source file to a Syntax tree using Roslyn then the Node which includes all other language elements is known as CompilationUnitSyntax.

NamespaceDeclarationSyntax

NamespaceDeclarationSyntax is a syntax node corresponding to namespace declaration in a source file. Similar to how a namespace declaration contains components like class declaration, method declaration etc. NamespaceDeclarationSyntax also contains syntax nodes for these syntactic constructs.

ClassDeclarationSyntax

ClassDeclarationSyntax is similar to NamespaceDeclarationSyntax but differs in its scope since it only contains class level language components.

MethodDeclarationSyntax

MethodDeclarationSyntax is also similar to NamespaceDeclarationSyntax but differs in its scope since it only contains method level language components.

FieldDeclarationSyntax

FieldDeclarationSyntax is a syntax node which contains information related to a declared field.

SOFTWARE METRICS TOOL

Compilers are complex and monolithic entities. Although there have been developments in the field on lexer and parser. Most of the compilers and parsers have their source code available for developers to peruse. But still it's very difficult to developers to use that code and start modifying it to add/update features. A case in point being javac i.e. java compiler which relies on hand coded LALR parser. OpenJDK community has been working on providing a java compiler using tools like ANTLR , lex and yacc. Though these tools provides lots of benefits to users but they lack important features like modularity and code reuse which are essential in projects.

So a good solution would be to provide information regarding language elements to the user. Even with this approach it would be a good idea if we can capture the information in the same manner it is used for execution or by the standard language compiler. It would be most helpful if we can collect most of the information that is available to language compiler. To collect information in this way the standard approach would be to use tools like ANTLR, lex and yacc. But there are some shortcomings of this approach. Firstly you have to replicate the behavior of lexer, parser and compiler components and that in itself is a non-trivial task. Secondly with each release and update in the programming language we need to update the code for compiler components. So unless we want this tool to be limited to some specific release of the programming language we need to update codebase on a regular basis. Thirdly even after we do all this work there is a question of verifying whether we are replicating the behavior of compiler in a reasonable manner or not. This is also a task that needs to be repeated with every update in codebase.

Roslyn or .Net compiler framework provides us with hooks into the .Net compiler such that we can collect the information that is available to compiler. Since we are collecting this information from the compiler we don't need to verify whether the same information is being processed by the compiler or not. Also in case of update in programming language(C#), the most that we need to do will be to update the version of Roslyn being used and make some minor changes in the code in case if there were any changes in standard Roslyn API.

So our approach is to use Roslyn API and provide users with information regarding most of language elements/components. The information provided is in two forms, it is provided in a form so that

user can use this information to directly i.e. metrics data is provided to the user or the information is provided in the form so that user doesn't need the knowledge about Roslyn API or how the language components interact with each other ?. User is provided information in the form of primitive data types or in the form of some custom classes. Since the number of custom classes is very small it won't create any obstacles for the user. In case the user wants to get more information or wants to do something that's not provided in the current version of this tool, we also expose the language component in a simplified manner. By simplified manner we mean that we have tried to reduce the complexity to the user by providing language components in Roslyn API form also.

Definitions used for calculating software metrics

1. Lines of Code – For calculating this software metric, we have used Hewlett-Packard's definition i.e. A line of code is a non-commented source statement: any statement in the program except for comments and blank lines (Grady and Caswell, 1987).
2. Lines of comments – All the number of lines of comments provided in the code.
3. Number of declaration – It counts the number of declarations of following code items
    a. Namespace
    b. Class
    c. Method
    d. Variable (at various levels)
4. Cyclomatic complexity – We used Ndepend tools definition to calculate cyclomatic complexity i.e. count the number of
    a. If statements
    b. While statements
    c. For statements
    d. For each statements
    e. Case statements(within switch statement)
    f. Default statements(within switch statement)
    g. Continue statement
    h. Goto statement

i. “&&” operator usage

j. “||” operator usage

k. Catch statement

l. Ternary operator

m. Method declarations

n. Class declarations

o. “??” operator

5. Number of parameters (29) – Count the total number of parameters (or arguments to methods) passed in different methods in the source file.
6. Out-degree – Total number of function calls in the source file to functions outside of current source file.
7. In-degree – Total number of calls to the functions inside this source file. It measures how much this class is used by other classes of the system.
8. Number of inherited data-items – Total number of data-items inherited by the source file (class).
9. Number of inherited data-items actually used - Total number of data-items inherited and used by the source file (class).
10. Number of inherited methods – Total number of methods inherited by the class.

Extensibility

Extensibility can be defined as the ability to extend a system. This extension might come as an addition of new functionality or modifying existing functionality (30). An extensible system might be defined as a system on which there is a minimal impact in terms of internal structure and data flow when a new functionality is added or existing one is updated. Designing software for extensibility assumes that not everything in a software can be designed in advance. It provides some framework using which the code can be adapted/updated or added. It helps in evolving the software which makes the software more useful for the end user and increases the life of the software too.

Extensibility can be broadly classified into below forms. This classification is based on what is changed in the software and how it is changed (31).

White Box extensibility

In white box extensibility the software can be extended by modifying or adding source code. It is the most flexible or least restrictive type of extensibility. Now based on how software is extended we can break it down into two more parts.

Open Box extensibility

Open box extensibility performs update in the source code in an invasive manner. The assumption here is that the source code for the software is available and user has the ability (license) to update it. Also since it directly updates existing code there is a possibility that it might introduced bugs in already working and tested part of the code. Based on how it is implemented it can be broken into an invasive approach where source code is updated and then binary updated accordingly. In other approach user creates a copy with updated code and creates another binary with updated logic. The latter approach is easy to implement but difficult to maintain.

Glass Box extensibility

Glass Box extensibility also updates the software but instead of updated existing code it allows user to add new functionality or features. The assumption here is that the user can see existing code but cannot modify it. User only has the capability to add extensions to existing source code. The user writes extensions to existing code in new files and adds them to the existing software leaving old source code untouched.

Gray Box extensibility

As we can see in in glass box functionality to extend the software the user does not necessarily need access to current source code. So in Gray box extensibility user has access to documentation of the existing source code i.e. specialization interface. This interface provides abstractions which can be refined or extended and how they interact with current system. Using this documentation user can extend the software.

Black Box extensibility

In the approach the user/programmer has no information about the implantation or architecture of the current software. This type of extension system are deployed and built using only interface

specification. It assumes that while building the current software it was designed such that all the possible extension scenarios were taken into consideration. This is the reason black box extensibility is more limited than white box extensibility.

This tool uses glass box extensibility. The methods available to user when extending the software are described in API documentation in the next section. This tool provides a web interface. User can select the code in the form of C# file and upload this web interface. On submission internally the project configuration files are updated. To reload the application so that new code can become usable a visual basic script is called. This visual basic script calls a batch file which restarts the IIS server and reloads the web application.

Extensibility in the tool

We have provided the capability to extend the tool i.e. user is able to upload user specific logic to the tool. Now apart from providing the ability for the user to extend the code we also provide api which can be used by user to write custom code. The idea behind such approach is to make the implementation easier for the end user. It helps the user in two ways one user can use the api to reduce the effort to code and learn about Roslyn, second this api can be used to implement code for a metric related to requirement which is not generic and not available which available tools. This new implementation could be some new metric or change in implementation due to user specific requirement.

Use of API

In this tool we have provided api which is also documented. This api provides pre- implemented methods which use Roslyn and custom classes. It is supposed to help user in both cases whether user understands Roslyn or not. Also it reduces the amount of coding that needs to be done.

Let go from easier examples to complex ones. Consider the case where user wants to get the program for the current program. If user implements it using Roslyn api it will take around 12 lines to implement whereas using the api it can be done in one method call.

```
public List<TypeDeclarationSyntax> getProgram(NamespaceDeclarationSyntax namespaceObj)
{
```

```
            List<TypeDeclarationSyntax> progs = new List<TypeDeclarationSyntax>();
            foreach (MemberDeclarationSyntax prog in (namespaceObj.Members))
            {
                if (prog.GetType().Name == "ClassDeclarationSyntax" || prog.GetType().Name ==
"InterfaceDeclarationSyntax")
                {
                    progs.Add((TypeDeclarationSyntax)prog);
                }
            }
            return progs;
        }
```

Apart from that we also provide another method to fetch program which returns a custom node to the user. This node stores information on whether this program is a class or interface, its name, its content, comments before and after the program.

Similarly for fetching methods in the program if implemented from beginning it will take around 11 lines on the other hand using api it's just one method call. For method also we provide a separate method which provides a simplified method object to the user.

```
         public List<CustomMethod> getMethodNodes(TypeDeclarationSyntax program)
        {
            IEnumerable<MethodDeclarationSyntax> methodsTemp =
program.DescendantNodes().OfType<MethodDeclarationSyntax>();
            List<CustomMethod> methodNodes = new List<CustomMethod>();

            foreach (MethodDeclarationSyntax method in methodsTemp)
            {
                CustomMethod node = new CustomMethod();
                node.Type = method.GetType().Name;
```

```
                    node.ReturnType = method.ReturnType.ToString();
                    node.Content = method.GetText().ToString();
                    node.Name = method.Identifier.Text;
                    node.DataItems = getMethodLevelDeclNode(method);
                    foreach (ParameterSyntax param in method.ParameterList.Parameters)
                    {
                        node.ParamTypes.Add(param.Type.ToString());
                    }
                    foreach (SyntaxTrivia syntaxTrivia in method.GetLeadingTrivia())
                    {
                        if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                            node.CommentBefore.Add(syntaxTrivia.ToFullString());
                    }

                    foreach (SyntaxTrivia syntaxTrivia in method.GetTrailingTrivia())
                    {
                        if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                            node.CommentAfter.Add(syntaxTrivia.ToFullString());
                    }
                    methodNodes.Add(node);
                }
                return methodNodes;
            }
```

Now if we move to other scenarios like getting the lines of code in a program. If implemented from beginning it will take around 38 lines to implement as compared to a method call using the provided api. In case there is a need to analyze the inheritance relationship between the programs there is a method

which creates a dictionary of custom class objects with inheritance relationship included. If implemented from beginning it will take around 85 lines of code (which doesn't include the code for custom object). Apart from this there are few utility methods are also provided like method of listing C# files in a location. The implemented part is around 43 lines long so using the api would same time re-coding 43 lines and can be replaced with a method call.

Implementation of custom code

Let's start with the assumption that an end user wants to add custom code the tool. The first step for that would be to implement the code. Consider the hypothetical metric where user wants to find out how many comments are provided before method declaration (say as a method documentation metric). If you implement it using Roslyn

```
static void Main()
    {
        String lines = "";
        if (File.Exists(filepath))
            lines = System.IO.File.ReadAllText(filepath);

        SyntaxTree tree = CSharpSyntaxTree.ParseText(lines);
        CodeSemantics semantics = new CodeSemantics();
        var root = (CompilationUnitSyntax)tree.GetRoot();
        var namespace1 =
root.DescendantNodes().OfType<NamespaceDeclarationSyntax>().First();
        IEnumerable<ClassDeclarationSyntax> progs =
namespace1.DescendantNodes().OfType<ClassDeclarationSyntax>();
        List<CustomMethod> methodNodes = new List<CustomMethod>();

        foreach (ClassDeclarationSyntax progName in progs)
        {
```

```
                    IEnumerable<MethodDeclarationSyntax> methodsTempProg =
progName.DescendantNodes().OfType<MethodDeclarationSyntax>();
                    methodNodes.AddRange(getMethodNodes(methodsTempProg));
                }

                IEnumerable<InterfaceDeclarationSyntax> interfaces =
namespace1.DescendantNodes().OfType<InterfaceDeclarationSyntax>();
                foreach (InterfaceDeclarationSyntax progName in interfaces)
                {
                    IEnumerable<MethodDeclarationSyntax> methodsTempInt =
progName.DescendantNodes().OfType<MethodDeclarationSyntax>();
                    methodNodes.AddRange(getMethodNodes(methodsTempInt));
                }
                Console.WriteLine(methodNodes.Count);
            }
            public static List<CustomMethod>
getMethodNodes(IEnumerable<MethodDeclarationSyntax> methodsTemp)
            {
                List<CustomMethod> methodNodes = new List<CustomMethod>();

                foreach (MethodDeclarationSyntax method in methodsTemp)
                {
                    CustomMethod node = new CustomMethod();
                    node.Type = method.GetType().Name;
                    node.ReturnType = method.ReturnType.ToString();
                    node.Content = method.GetText().ToString();
                    node.Name = method.Identifier.Text;
                    node.DataItems = getMethodLevelDeclNode(method);
```

```
                    foreach (ParameterSyntax param in method.ParameterList.Parameters)
                    {
                        node.ParamTypes.Add(param.Type.ToString());
                    }
                    foreach (SyntaxTrivia syntaxTrivia in method.GetLeadingTrivia())
                    {
                        if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                            node.CommentBefore.Add(syntaxTrivia.ToFullString());
                    }

                    foreach (SyntaxTrivia syntaxTrivia in method.GetTrailingTrivia())
                    {
                        if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                            node.CommentAfter.Add(syntaxTrivia.ToFullString());
                    }
                    methodNodes.Add(node);
                }
                return methodNodes;
            }

            public static List<Node> getMethodLevelDeclNode(MethodDeclarationSyntax method)
            {
                IEnumerable<LocalDeclarationStatementSyntax> fieldsTemp =
method.DescendantNodes().OfType<LocalDeclarationStatementSyntax>();
                List<Node> fields = new List<Node>();
                foreach (LocalDeclarationStatementSyntax field in fieldsTemp)
```

```
            {
                Node node = new Node();
                node.Type = field.GetType().Name;
                node.Content = field.GetText().ToString();
                foreach (SyntaxTrivia syntaxTrivia in field.GetLeadingTrivia())
                {
                    if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                        node.CommentBefore.Add(syntaxTrivia.ToFullString());
                }

                foreach (SyntaxTrivia syntaxTrivia in field.GetTrailingTrivia())
                {
                    if (syntaxTrivia != null && syntaxTrivia.ToFullString() != "" &&
syntaxTrivia.ToFullString() != Environment.NewLine)
                        node.CommentAfter.Add(syntaxTrivia.ToFullString());
                }
                fields.Add(node);
            }
            return fields;
        }
```

Now if we implement the same using api provided. We can see a big change in the complexity as well as number of lines of code.

```
        static void Main()
        {
            String lines = Utility.readContent(filePath);
            CodeSemantics semantics = new CodeSemantics();
            var root = (CompilationUnitSyntax)CSharpSyntaxTree.ParseText(lines).GetRoot();
```

```
            List<TypeDeclarationSyntax> progs =
semantics.getProgram(semantics.getNamespace(root));
            List<CustomMethod> methodNodes = new List<CustomMethod>();
            foreach (TypeDeclarationSyntax prog in progs)
            {
                methodNodes.AddRange(semantics.getMethodNodes(prog));
            }
            int count = 0;
            foreach (CustomMethod met in methodNodes)
            {
                count = count + met.CommentBefore.Count;
            }
            Console.WriteLine(count);
        }
```

Evaluation of the tool

Software metrics tool is a web appliction which is based on .Net technology. For UI it uses asp.net and for server code it uses C# language. It runs on a IIS server and database is based off LocalDB. So to customize or extend it the user can use a visual studio express which is available for free. It has features such as ability to calculate metrics and statistics. Abillity to add custom code metric and statistics. The metrics provided by default are number of lines of code, number of lines of comment, declaration count, cyclomatic complexity, number of parameters, in-degree, out-degree, number of inherited methods and data items. The statistics provided are average, max and range. The features of the tool and how to use it are explained below.

The figure below shows the home page of the application. The home page contains ability to browse and upload a file which contains custom implementation. It provides ability to provide location of the project which user would like to calculate metrics for.It also provides ability to select metrics and statistics to be calculated.

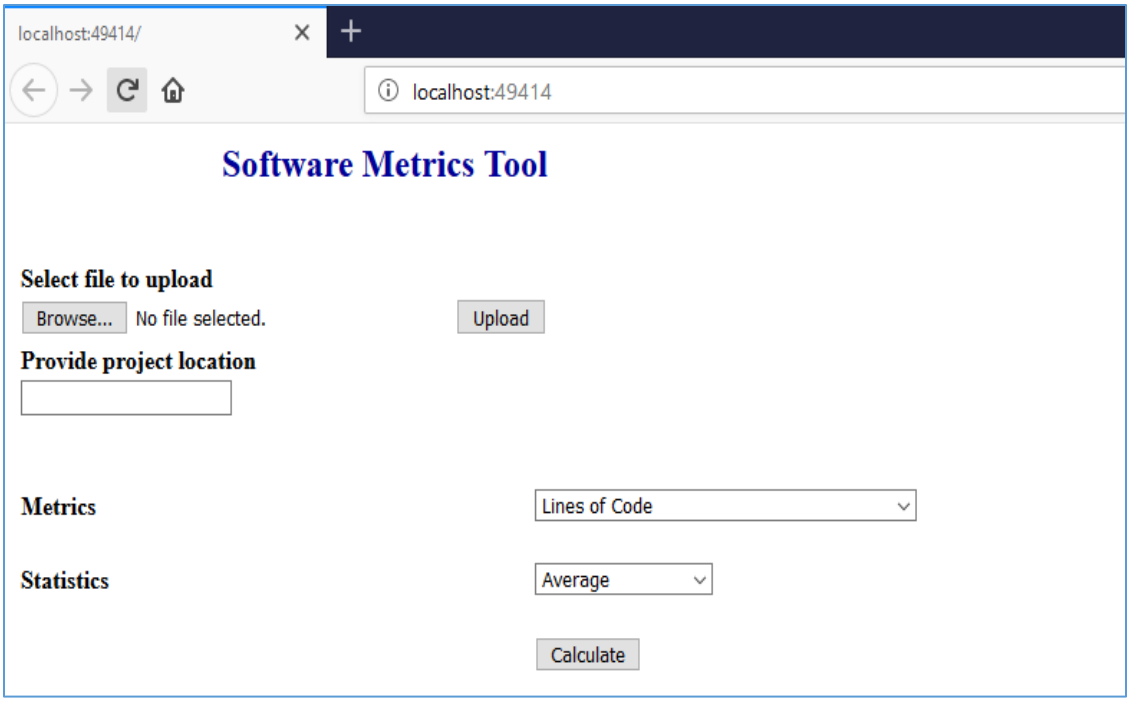

Figure 2 – Home page of the tool

The figure below show the drop down for metrics. It shows the default set of metrics provided by the tool initially. If user adds their own metric then it will also show up in the same drop down.

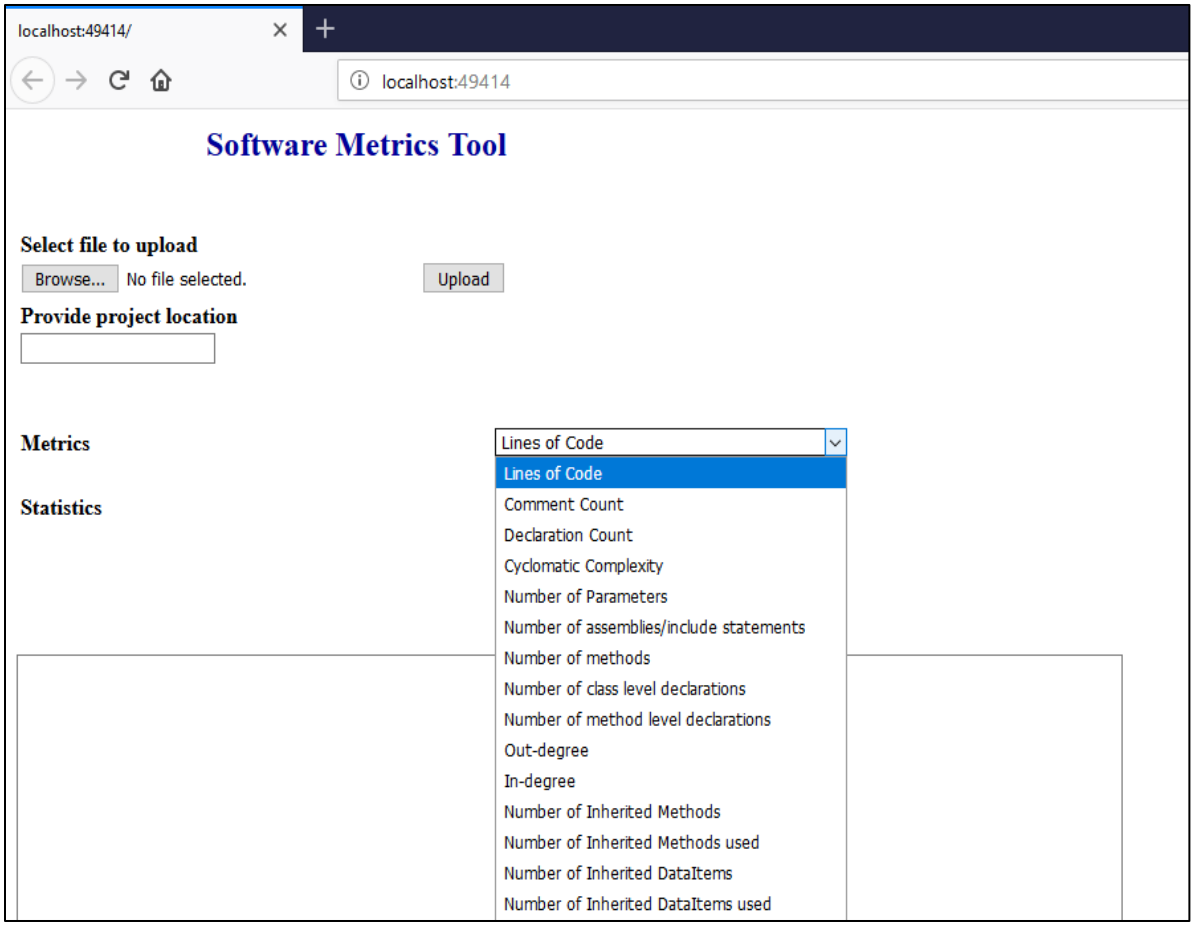

Figure 3 – Drop down list of metrics

The second drop down shows the default statistics provided with the tool. Similar to code metrics if user add their own statistic it will show up in the drop down.

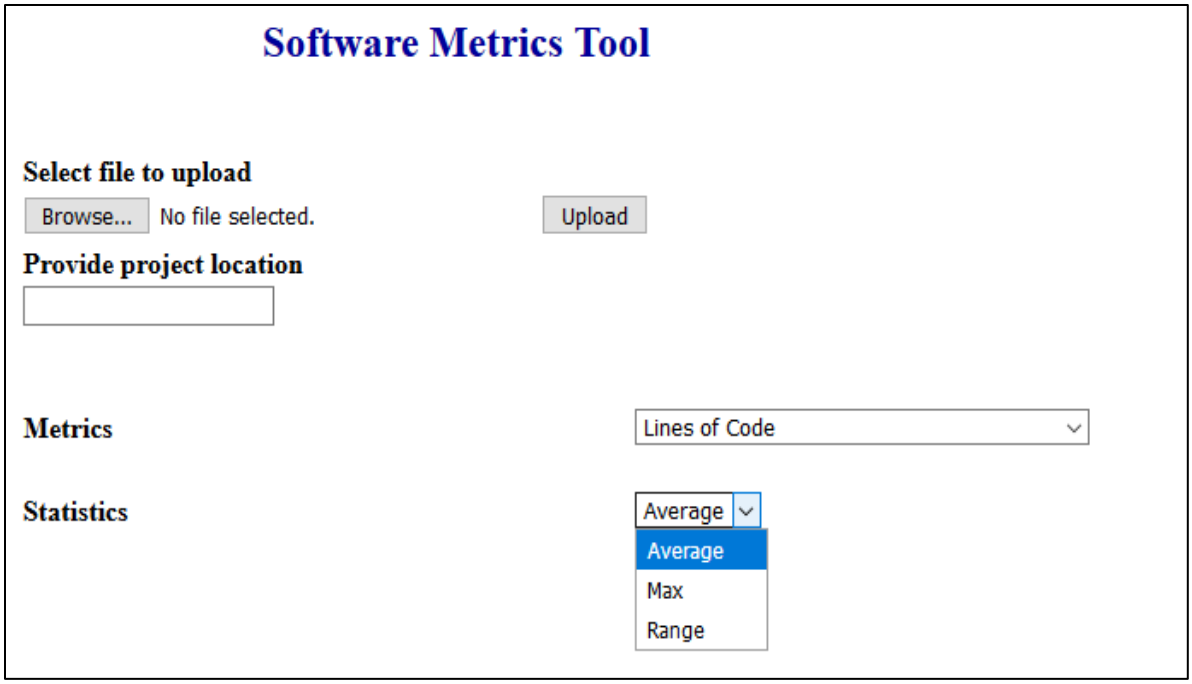

Figure 4 – Statistics drop down list

To upload custom metric or statistic user should click on first browse and then upload. The thing to note here is user can upload only C# code.

Figure 5 – File select and file upload button

On clicking the browse button a file upload window shows up. User can then select a C# file (with .cs extension).

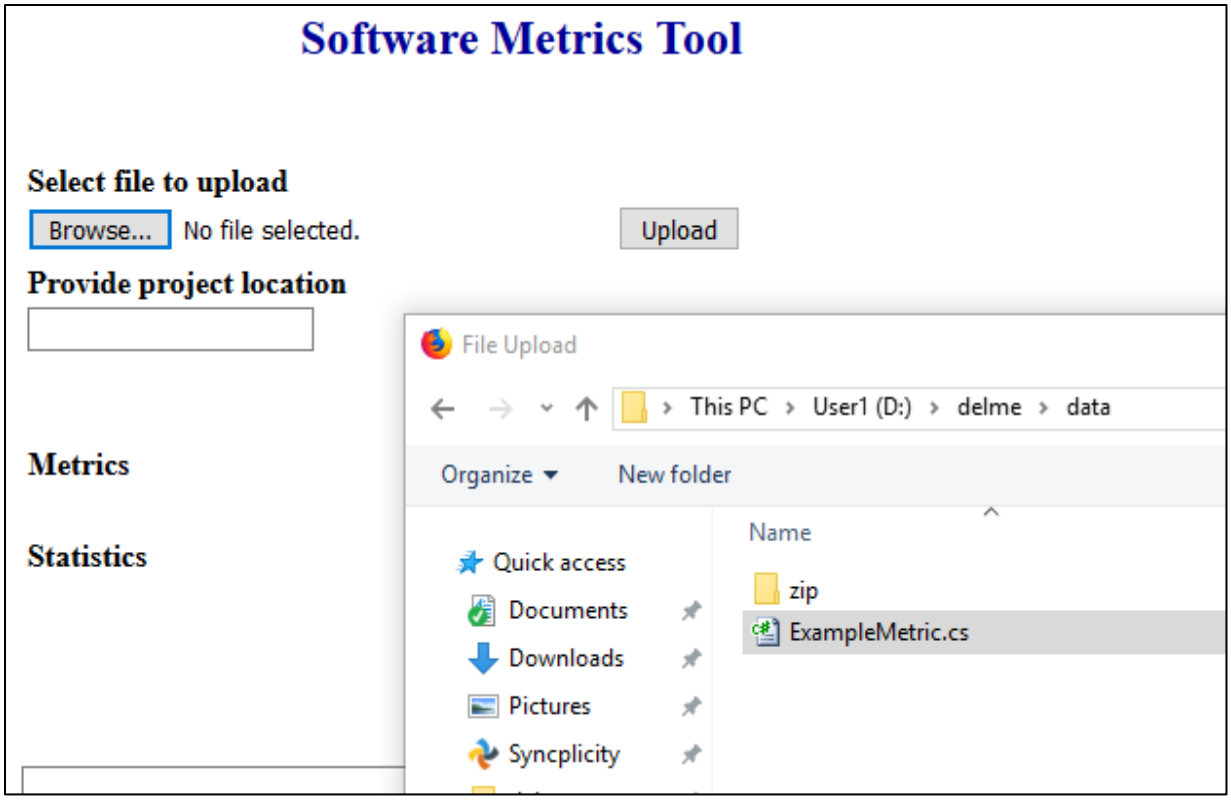

Figure 6 – Selecting files using browse button

Suppose a user selects another file like a groovy file then as you can see in the figure below and error message comes up on the screen. Similarly if user clicks on upload button without select a file error message is shown on the home page.

Figure 7 – Validation for non C# files

When user selects a file from file upload window, file name shows up on the page and when clicked on upload a new pop up comes up. In this window user is prompted to select whether the uploaded code contains a metric or a statistic. There is one more option of others in case user just wants to execute the statistics on this file.

Figure 8 – Selecting a custom file for upload

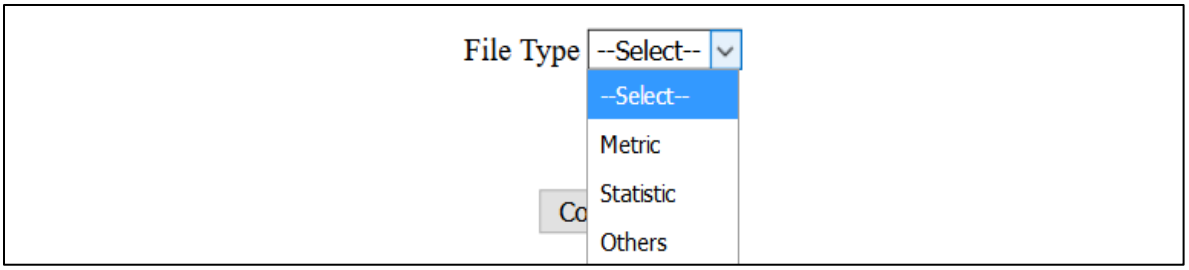

Figure 9 - Selecting the type of custom logic being uploaded in pop up window

Once use selects a value from file type. User is prompted to provide a name for the metric or statistic depending upon what was selected in previous drop down. Here metric name field is mandatory. Once user clicks on confirm button, this code gets uploaded to the web application. From now onwards the user can see the uploaded metric (or statistic) on the drop down menu of metric (or statistic).

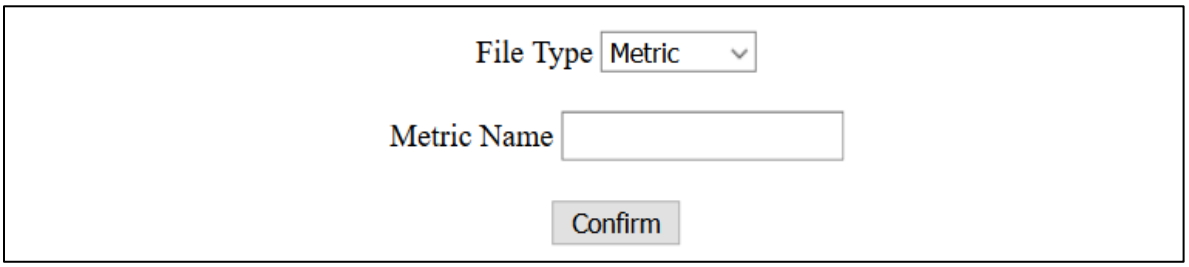

Figure 10 – Text box for adding name of the custom metric

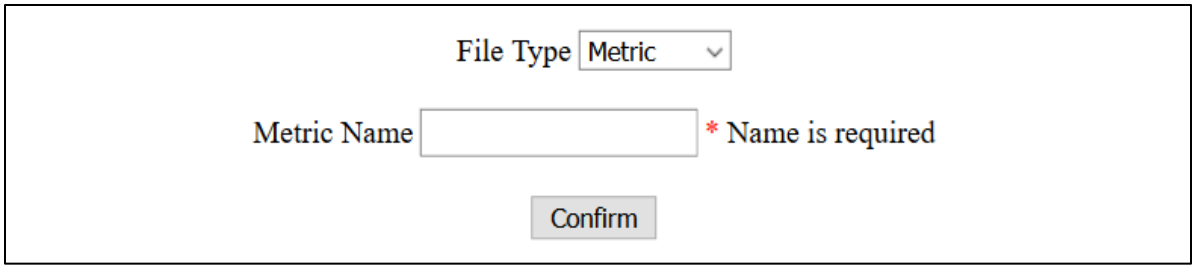

Figure 11 – Validation for custom metric name

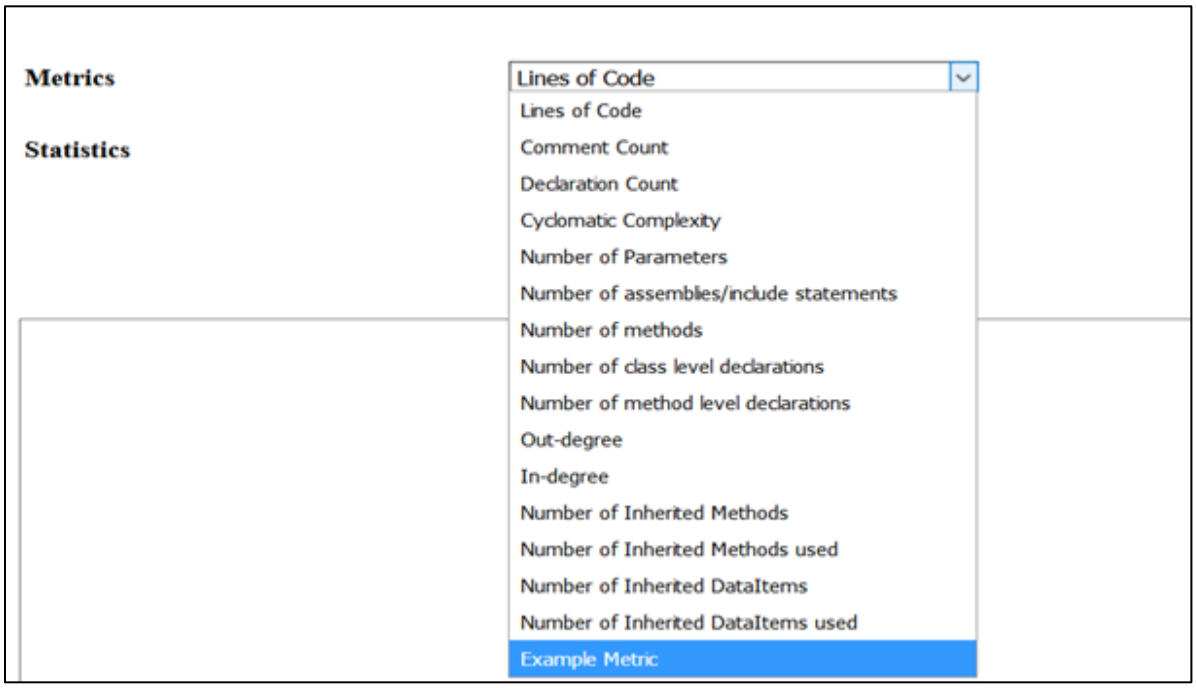

Figure 12 – Custom metric in metric drop down list

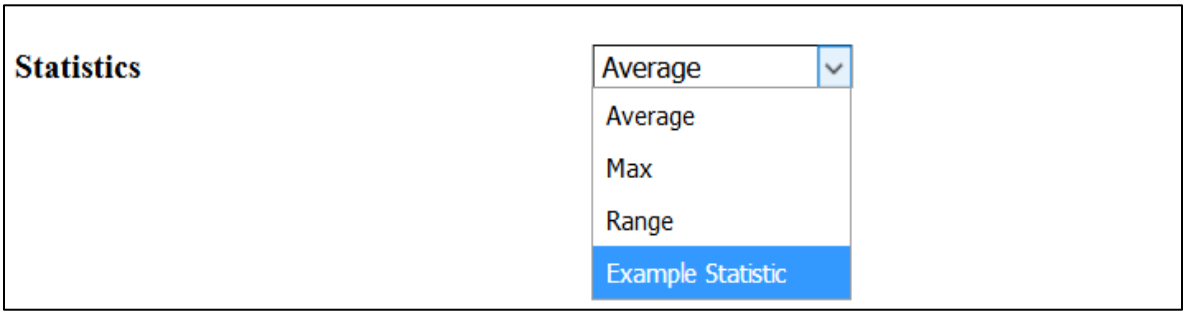

Figure 13 – Custom statistic in statistics drop down list

Also to calculate different statistics and metrics for a number of files or a project. First provide the project location under the text box and then click on ‘Calculate’ button. User can see the results displayed

in the text box at the bottom of the page. This text box is scrollable so user can scroll through in case there are more files.

There is a ‘Clear Content’ button next to result box. It helps in resetting the page. It clears up the results pane and the project location entered and file selected for upload (if any).

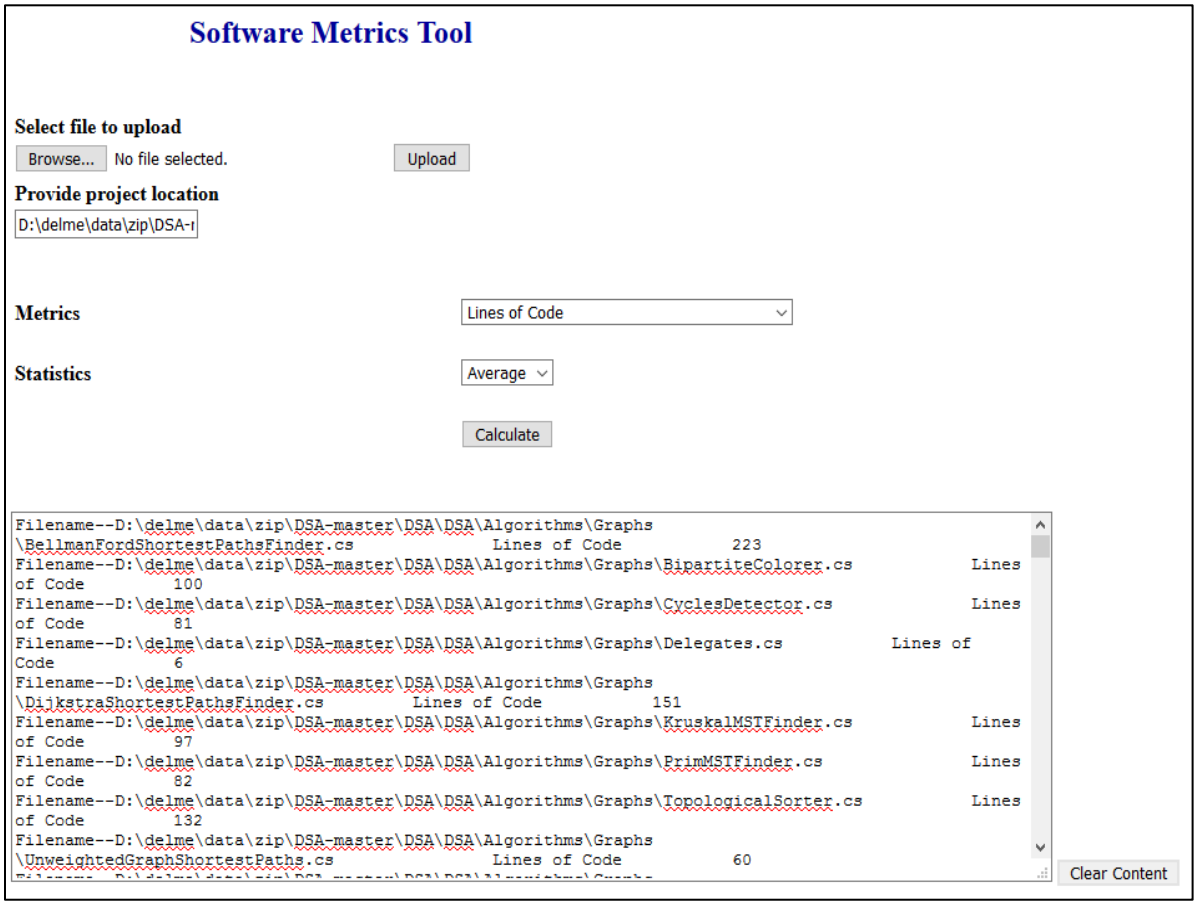

Figure 14 – Metric results

Testing of the tool

This web application was tested using multiple open source projects available online. All the projects used for evaluation are C# projects which are open source and available for download for free (32) (33). For testing this tool was used to calculate metrics and statistics selected randomly for any project from the group. To vary the project size the projects with different number of C# files were selected, maximum number of files in a project were 245. Covering all the projects included in evaluation there were a total of 1777 C# source code files that were part of this evaluation. The name of the projects and number of files it contained are listed below.

- Customer Relationship Management System – This project had 245 C# files.
- School Management System – This project contained 183 source code files.
- Pharmacy Information Management System – Number of C# files were 168 for this project.
- Gym Management System – This project had 123 C# source code files.
- Point of Scale – There were 117 source code files in this project.
- Employees Management System - There were 91 C# files in this project.
- Kiddie Care – This project had 88 C# files.
- Recruitment Process System - This project contained 88 source code files.
- RTA Information System - Number of C# files were 82 for this project.
- Network Load Balancing System - This project had 66 C# source code files.
- Peer to Peer File Sharing System - There were 56 C# files in this project.
- Dental Management System - There were 49 source code files in this project.
- Banking System - This project had 46 C# files.
- Blood Inventory Management System - This project had 43 C# source code files.
- Online Discuss Forum - This project had 42 C# source code files.
- Grade Book Application - This project contained 38 C# source code files.
- Riverside Parts Master Project – There were 38 C# source code files in this project.
- Project Management System - This project had 36 C# source code files.
- Database Enterprise Manager - This project contained 31 C# source code files.
- Sales and Inventory System - This project had 27 C# source code files.
- Knowledge Processing Application - This project had 26 C# source code files.
- Data Hiding Application - This project had 22 C# source code files.
- Network Monitoring System - Number of C# files were 19 for this project.
- Password Generator - This project contained 16 source code files.
- Count-down Timer - There were 12 C# source code files in this project.
- Battery Information Application - There were 10 C# files in this project.

- OOP Login - Number of C# files were 9 for this project.
- Calculator- This project had 6 C# files.

The tool executed as per expectations on all the projects above and was used to successfully calculate different statistics and metrics.

Apart from the above projects the tool was tested using projects from github too.

- SimplCommerce (34) – This project has 607 c-sharp files spread across 36 modules. The project was tested on individual module basis.
- DSA (35) - There were around 100 C# files in this project for implementation and around 80 C# files for unit testing.
- SeeGit (36) – There were around 33 c sharp code files in this project. Apart from that there were 5 unit test files.
- Fluent command line parser (37) – There were around 150 C# code files. They were spread in actual code, examples and unit test.
- NJsonSchema (38) – There were around 300 source code files spread across multiple modules in this project.

API Documentation (Methods)

Table 1 – findClasses method API

Name	findClasses
Description	Find all the relevant classes at the path provided(path can be directory or a file)
Parameters	String path
Returns	ArrayList

Table 2 – readContent method API

Name	readContent
Description	Read all the content inside a class/code body
Parameters	String filepath
Returns	String

Table 3 – getCodeTree method API

Name	getCodeTree
Description	Generate a syntax tree based on the code provided
Parameters	String codeText
Returns	SyntaxTree(refer appendix)

Table 4 – getRoot method API

Name	getRoot
Description	Get the root node of the tree
Parameters	SyntaxTree tree
Returns	CompilationUnitSyntax

Table 5 – getSemanticModel method API

Name	getSemanticModel
Description	Get the semantic model(refer appendix) containing all the semantic information of the tree
Parameters	SyntaxTree tree
Returns	SemanticModel

Table 6 – getAssemblyUsed method API

Name	getAssemblyUsed
Description	Get the assemblies used in the source file in terms of custom Node class
Parameters	CompilationUnitSyntax root
Returns	List<Node>

Table 7 – getAssemblyUsedCount method API

Name	getAssemblyUsedCount
Description	Get the number of assemblies used in the source file
Parameters	CompilationUnitSyntax root
Returns	Int

Table 8 – getNamespace method API

Name	getNamespace
Description	Get the namespace node in terms of Roslyn API object
Parameters	CompilationUnitSyntax root
Returns	NamespaceDeclarationSyntax

Table 9 – getNamespaceNode method API

Name	getNamespaceNode
Description	Get the namespace node in terms of custom Node object
Parameters	CompilationUnitSyntax root
Returns	Node

Table 10 – getProgram method API

Name	getProgram
Description	Get a list of classes declared in the source file in terms of a list of Roslyn API objects
Parameters	NamespaceDeclarationSyntax namespace1
Returns	List<ClassDeclarationSyntax>

Table 11 – getProgramNode method API

Name	getProgramNode
Description	Get a list of classes declared in the source file in terms of a list of custom node objects
Parameters	CompilationUnitSyntax root
Returns	List<Node>

Table 12 – getMethods method API

Name	getMethods
Description	Get a list of methods declared in the a class in terms of a list of Roslyn API objects
Parameters	ClassDeclarationSyntax program
Returns	List<MethodDeclarationSyntax>

Table 13 – getMethodNodes method API

Name	getMethodNodes
Description	Get a list of methods declared in the a class in terms of a list of custom node objects
Parameters	ClassDeclarationSyntax program
Returns	List<CustomMethod>

Table 14 – getClassLevelFieldDecl method API

Name	getClassLevelFieldDecl
Description	Get a list of variable declarations in a class in terms of a list of Roslyn API object
Parameters	ClassDeclarationSyntax prog
Returns	List<FieldDeclarationSyntax>

Table 15 – getClassLevelFieldDeclNode method API

Name	getClassLevelFieldDeclNode
Description	Get a list of variable declarations in a class in terms of a list of custom node objects
Parameters	ClassDeclarationSyntax prog
Returns	List<Node>

Table 16 – getMethodLevelDecl method API

Name	getMethodLevelDecl
Description	Get a list of variable declarations in a class in terms of a list of Roslyn API objects
Parameters	MethodDeclarationSyntax method
Returns	List<FieldDeclarationSyntax>

Table 17 – getMethodLevelDeclNode method API

Name	getMethodLevelDeclNode
Description	Get a list of variable declarations in a method in terms of a list of custom node objects
Parameters	MethodDeclarationSyntax method
Returns	List<Node>

Table 18 – getLineCount method API

Name	getLineCount
Description	Get the lines of code in a class(definition used to calculate it is provided earlier)
Parameters	Int64 count, CompilationUnitSyntax root
Returns	Int64

Table 19 – getCommentCount method API

Name	getCommentCount
Description	Get the lines of comments in a class
Parameters	Int64 count, CompilationUnitSyntax root
Returns	Int64

Table 20 – Lines method API

Name	Lines
Description	Count the number of lines in the string provided
Parameters	String s
Returns	Long

Table 21 – findDeclarationCount method API

Name	findDeclarationCount
Description	Count the number of declaration in a class
Parameters	Int64 count, CompilationUnitSyntax root
Returns	Int64

Table 22 – countNumberOfParameters method API

Name	countNumberOfParameters
Description	Count the total number of parameters in a class
Parameters	Int64 count, CompilationUnitSyntax root
Returns	Int64

Table 23 – findCyclomaticComplexity method API

Name	findCyclomaticComplexity
Description	Calculate the cyclomatic complexity for the class
Parameters	Int64 count, CompilationUnitSyntax root
Returns	Int64

Table 24 – countExecutableStatements method API

Name	countExecutableStatements
Description	Count the number of executable statements in the class
Parameters	Int32 count, CompilationUnitSyntax root
Returns	Int32

Table 25 – NumberOfInheritedDataItemsUsed method API

Name	NumberOfInheritedDataItemsUsed
Description	Number of methods inherited from parent used in the class
Parameters	CustomClass currentClass
Returns	Int

Table 26 – NumberOfInheritedDataItems method API

Name	NumberOfInheritedDataItems
Description	Number of data items inherited from parent
Parameters	CustomClass currentClass
Returns	Int

Table 27 – NumberOfInheritedMethodsUsed method API

Name	NumberOfInheritedMethodsUsed
Description	Number of methods inherited from parent used in the class
Parameters	CustomClass currentClass
Returns	Int

Table 28 – NumberOfInheritedMethods method API

Name	NumberOfInheritedMethods
Description	Number of methods inherited from parent
Parameters	CustomClass currentClass
Returns	Int

Table 29 – InheritancePreprocess method API

Name	InheritancePreprocess
Description	Calculate the inheritance relationship between the files given
Parameters	String path
Returns	Dictionary with key as class name and value as custom class object of that class

Table 30 – InOutDegreePreprocess method API

Name	InOutDegreePreprocess
Description	calculate the in-degree and out-degree value for lists of files(does not include inheritance effect)
Parameters	String path
Returns	Returns a hashtable which contains two hashtables one for in-degree and other for out-degree. Each inner hashtable has class name as key and in/out degree as value.

Table 31 – InOutDegInheritProcessing method API

Name	InOutDegInheritProcessing
Description	Update the in-degree and out-degree values by parents effect into children
Parameters	CustomClass currentClass
Returns	A dictionary with key as class name and value as custom class object of that class

CONCLUSION

The idea behind this software was to capture data for software metrics as per definitions used and then compare them with the expected value. Here we got those definitions from sources in the references. The definitions used are also explained in earlier sections. There are multiple tools available online like NDepend, SLOC, Semantic Design, SourceMontior, McCabe IQ which can be used to calculate code metrics. For tools like SoureMonitor, SLOCCount, SourceCode Metrics, NDepend, Semantic Design limitation for them is that they have to be installed on each physical machine where user is expecting to use it. Also another limitation with tools like SourceCode Metrics, SLOC,LOC Metrics, SLOCCount is that they either have limited functionality or features available (in terms of code metrics that can be calculated by the tool) . Limitation with tools like SourceCode Metrics, NDepend, McCabe IQ, Semantic Design, EZ metrix is that they are commercial applications which are available for free only for a limited time.

The proposed tool tries to resolved most of these problems. This tool is a web application so it's not required to be installed in each physical machine. It can be used to calculate code metrics like cyclomatic complexity, metrics related to inheritance, lines of code, lines of comment, number of parameters etc. It can also be used to calculate statistics for these code metrics. This tool also provides a capability to extend the tool by adding custom metrics and statistics. It provides user the ability to add a new metrics or a different implementation of an existing metric to uploaded and added to the tool. This feature is not available in other tools. Some of them like NDepend and EZ Metrix allow user to add rules but that's not the same as ability to add new metrics.

This software considered the structure of a program as a basis. The structure here refers to how program is understood/read by the compiler. We got access to that information via Roslyn api. This ensured that we are reading the program as it's supposed to be read(by the compiler).The process of creating the software initially started with adding basic language features and testing whether the software was able to collect the information reliably. Then add some complexity or language feature in test code and modify the software to run some tests again. After multiple cycles we introduced complex features like inheritance and fan-in and fan-out (39).

This tool can be enhanced to calculate code metrics for multiple languages. In its current format it supports files with .cs extension only. There can also be improvements in the area of user interface, capability to create and save reports on per user basis. In its current format it supports basic validation in the ui it can be extended to make the application more stable and secure.

REFERENCES

1. Andrew Meneely, Ben Smith, and Laurie Williams,. Validating Software Metrics: A Spectrum of Philosophies. *ACM Transactions on Software Engineering and Methodology.* November 2012.

2. Software Metrics. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Software_metric.

3. Mills, Everald E. *SEI Curriculam Module.* s.l. : Software Engineering Institute, 1988.

4. *The Research on Software Metrics and Software Complexity Metrics.* Tu Honglei, Sun Wei, Zhang Yanan. 2009, International Forum on Computer Science-Technology and Applications.

5. Zuse, Horst. *A Framework of Software Measurement.* Berlin : Walter de Gruyter & Co, 1998.

6. *SourceMonitor - home page.* [Online] http://www.campwoodsw.com/sourcemonitor.html.

7. *SourceMonitor - overview.* [Online] https://www.safaribooksonline.com/library/view/windows-developer-power/0596527543/ch09s03.html.

8. *SLOCCount - home.* [Online] https://www.dwheeler.com/sloccount.

9. Park, Robert E. *Software Size Measurement: A Framework for Counting Source Statements.* s.l. : Software Engineering Institute, 1992.

10. *SLOCCount - architecture.* [Online] https://www.dwheeler.com/sloccount/sloccount.html.

11. *Source code metrics.* [Online] https://msquaredtechnologies.com.

12. *Source Code Metrics - metrics.* [Online] (https://msquaredtechnologies.com/Code-Metrics.html.

13. *NDepend- home page.* [Online] https://www.ndepend.com/.

14. *NDepend - features.* [Online] https://en.wikipedia.org/wiki/NDepend).

15. *NDepend -CQLinq feature.* [Online] https://www.ndepend.com/features/cqlinq#CQL.

16. *SLOC tool.* [Online] https://github.com/flosse/sloc.

17. *McCabe IQ - software metrics.* [Online] http://www.mccabe.com/iq_research_metrics.htm.

18. *Semantic Design- tools.* [Online] http://www.semdesigns.com/Products/Tools.html.

19. *Semantic Design - metric tools.* [Online] http://www.semdesigns.com/Products/Metrics/index.html?Home=Tools.

20. *Semantic Design - C# metrics.* [Online] http://www.semdesigns.com/Products/Metrics/CSharpMetrics.txt.

21. *Semantic Design - DMS toolkit.* [Online] http://www.semdesigns.com/Products/DMS/DMSToolkit.html.

22. *EZ - Metrix overview.* [Online] http://www.jamesheiresconsulting.com/scc/downloads/EZ-Metrix%20User%20Guide.pdf.

23. *EZ - Metrix home.* [Online] http://www.jamesheiresconsulting.com/Products.htm.

24. *LOC Metrics - home.* [Online] http://www.locmetrics.com/.

25. *Visual Studio Code Metrics.* [Online] https://msdn.microsoft.com/en-us/library/bb385914.aspx.

26. Alan J. Perlis, Frederick G. Sayward and Mary Shaw. *Software Metrics.* 1981.

27. *Shewart-Deming Cycle.* [Online] https://en.wikipedia.org/wiki/PDCA.

28. Karen NG, Matt Warren, Peter Golde, Anders Hejlsberg. *The Roslyn Project.* s.l. : Microsoft, 2012.

29. Software metrics- number of parameters. *realsearchgroup.org.* [Online] http://realsearchgroup.org/SEMaterials/tutorials/metrics/.

30. Extensibility. [Online] https://en.wikipedia.org/wiki/Extensibility.

31. Zenger, Matthias. *Programming language - Abstractions for Extensible Software Components(Doctral Thesis).* s.l. : Swiss Federal Institute of Technology, Lausanne, Switzerland, 2004.

32. *C# open source projects.* [Online] https://www.sourcecodester.com/c-sharp .

33. *C# projects.* [Online] http://1000projects.org/c-projects-and-source-code-free-download.html.

34. SimplCommerce. [Online] https://github.com/simplcommerce/SimplCommerce.

35. DSA. [Online] https://github.com/abdonkov/DSA.

36. SeeGit. [Online] https://github.com/Haacked/SeeGit.

37. fluent-command-line-parser. [Online] https://github.com/fclp/fluent-command-line-parser.

38. NJsonSchema. [Online] https://github.com/RSuter/NJsonSchema.

39. Software Metrics - Fan out. *mit.edu.* [Online] http://sunnyday.mit.edu/16.355/classnotes-metrics.pdf.

40. Pfleeger, Fenton E. Norman and Shari Lawrence. *Software Metrics: A rigorous and practical approach.* 1997.

41. *An Empirical Study of Social Networks Metrics in Object-Oriented Software.* Giulio Concas, Michele Marchesi, Alessandro Murgia, and Roberto Tonelli. 2010, Advances in Software Engineering.

42. Michael Unterkalmsteiner, Tony Gorschek, A.K.M. Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt,. Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review. *IEEE Transactions on software engineering , Vol 38.* March/April 2012.

APPENDIX. DEFINITIONS OF TERMS USED

Node

It is a custom object which is used to represent any node in the code tree. It is used to provide user relevant information without any knowledge of Roslyn API. It contains following fields.

- type (String) - Type of the node. Eg ClassDeclarationSyntax, MethodDeclarationSyntax.
- content (String) - Content of the node in String format(includes all the internal nodes).
- commentBefore (List of string) - Trivia provided before this node
- commentAfter (List of string) - Trivia provided after this node

CustomClass

It is a custom object which is used to represent any class in the code base. This custom object provides user necessary information regarding a class without going through Roslyn API. It contains following fields.

- parent (CustomClass) – Contains object of parent class if any.
- childrens (List of CustomClass) – Contains list of children classes if any.
- methods (List of CustomMethods) – Contains a list of CustomMethod objects.
- namespace (Node) - Contains a custom Node object for namespace of this class.
- isClass (boolean) – Contains a Boolean value which can be used to separate a class from interface.
- outDeg(int) – Contains the out-degree for this class.
- inDeg(int) – Contains the in-degree for this class.
- name (string) – Contains the name of the class.
- textContent (string) – Contains the text of the class(basically all the code enclosed by this class). This field is provided in case user wants to use Roslyn API to get more data about this class.

CustomMethod

It is a custom object which extends Node custom class. Along with the data provided by Node object it also provides the data corresponding to a method in a class. It contains following fields.

- accessMod (String) – Contains the access modifier information of the method.
- returnType (String) - Contains the return type of the method.

- paramTypes (List of string) – Contains a list of parameters for this method.
- dataItems (List of Node) - list of custom node objects containing information about data items in the method.

Data types used (in API documentation)

- Int / Int32 – This datatype is stored as a four byte integer or we can say it is a signed 32 bit integer. Its range varies from -2,147,483,648 to 2,147,483,647 inclusive.
- Int64 – This datatype is stored as a 8 byte integer or we can say it is a signed 64 bit integer. Its range varies from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.