SOFTWARE METRICS TOOL

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

By

Vijay Reddy Reddy

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

Major Department:
Computer Science

July 2016

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SOFTWARE METRICS TOOL

By

Vijay Reddy Reddy

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

Kendall Nygard

Bernhardt Saini-Eidukat

Approved:

August 4, 2016
Date

Brian M. Slator
Department Chair

ABSTRACT

A software metric is the measurement of a particular characteristic of a software program. These metrics are very useful for optimizing the performance of the software, managing resources, debugging, schedule planning, quality assurance and periodic calculation of software metrics increases the success rate of any software project. Therefore, a software metrics tool is developed in this research which provides an extensible way to calculate software metrics. The tool developed is a web-based application which has the ability to calculate some of the important software metrics and statistics as well. It also provides a functionality to the user to add additional metrics and statistics to the application. This software metrics tool is a language-dependent tool which calculates metrics and statistics for only C# source files.

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this disquisition, a great many people have contributed to its production. I owe my gratitude to all those people who have made this possible and because of whom my graduate experience has been one that I will cherish forever. I would like to extend my deepest gratitude to my advisor Dr. Kenneth Magel for introducing me with this topic and giving an opportunity to work under his guidance. The door of Dr. Magel was always open whenever I ran into any trouble spot or had any questions in my research or writing. I am very thankful to Mr. Rajat Singh whose help and effort was the biggest support for my research.

I am very grateful to my committee members Dr. Kenneth Magel, Dr. Kendall Nygard and Dr. Bernhardt Saini-Eidukat for their valuable time and suggestions regarding my thesis.

Finally, I would like to thank my parents and my friends, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

A software metric, according to definition [4] [5], is a standard of measure of a degree to which a software system or process possesses some property. In simple words, it can be defined as the measurement of a particular characteristic of a program's performance or efficiency. The fundamental goal is to obtain objective, reproducible and quantifiable measurements that can be useful for quality assurance, performance optimization, debugging, schedule planning, cost estimation and proper management of workforce and other resources.

Software metrics can provide a wide range of information about a software system as each metric looks at a different aspect of the software. These include measuring the quality of the software which basically is done by calculating some metrics using the source code, estimating the cost of the software project that includes maintenance, research and typical costs associated with a project, defining the schedule of the project by looking at the functionality and also at the documents produced, measuring the size and complexity of the project based on the code or at the macro-level of the project and the project's dependency on other projects.

Software metrics are basically categorized into product, process and project/resource metrics. Product metrics deal with the characteristics of the product like size (can be Lines of Code (LOC), duration of tests, etc.), complexity, performance, design features and quality level, and can be used at any stage of software development i.e., from requirements phase to deployment phase. Product metrics can be sub-divided into the following categories: Design metrics which are useful means for improving the quality of the software and are computed from the requirements or design documents and they are not dependent on the syntax or data structure, Object-oriented metrics which are particularly used to identify the error proneness of different classes in the source code, predict maintenance efforts and error rate, and Communication

metrics which deal with the communication artifacts like electronic e-mails, weekly project meetings, intra-team (interactions within a team) and inter-team (interactions between different teams) communications. Process metrics can be used to measure and improve the process of software development and maintenance. Examples include the type of methodology used, the effectiveness of defect removal during development, the pattern of testing defects and the response time of the fix process. Project metrics deal with the characteristics of the overall project like the cost, schedule, time, number of software developers needed and productivity [7].

Along with the advantages, there are some limitations too for the use of software metrics in software development process. The main concern about software metrics is that there is no standard definition for most of them and there is always a debate on what they mean and which of them matter in a particular situation [2]. The results provided by different software metric tools are sometimes inconsistent and they only compute a selected number of metrics. Therefore, different tools are needed for different software components which makes it not only hard to use but also often unusable. There are some other arguments which point out that simple software metrics can cause more harm than good and they can have harmful effects on developers due to stress and anxiety, while some practitioners argue that software metrics have become an integral part of the software development process and they have a positive impact on developers' performance. But, as DeMarco stated in [3], 'You can't control what you can't measure', it is always better to do some quantification rather than doing nothing. According to the Standish Chaos Report (2015) [6], for the years from 2011 to 2015, on an average, only 29% of the projects were successful (i.e. on time, on budget with a satisfactory result) out of the total 50,000 projects that they have studied from around the world. This shows that there is still a lot of work to be done to achieve higher success rates in software development projects. A good way to

increase the success rate of any software project is to calculate different software metrics which in turn can be very useful to predict defects in code and to determine code quality.

The software metrics tool developed in this research provides an extensible way to calculate software metrics. This tool is basically a web-based application which calculates some of the important software metrics like the number of executable statements, In-degree, Out-degree, number of declarations, number of parameters, number of inherited data items, number of inherited data items actually used, number of inherited methods, number of inherited methods actually used, and Cyclomatic complexity, and also has the ability to compute statistics like the average value, maximum value and the range of each metric in the application. In addition, it provides an ability to the user to add additional metrics and statistical calculations at any time to the application by writing a new module and uploading it through the API provided by the tool. The tool is developed in Microsoft Visual Studio using C# programming language and the Roslyn API, a .Net Compiler API, to parse the language. This is a language-dependent tool which calculates the software metrics and statistics for C# source files only.

The ability of this tool to dynamically allow addition of new modules for metrics and statistical calculations and the ability to calculate different metrics and statistics online without the need to download or install any software makes it unique and easy to use when compared to different software metric tools available in the market.

LITERATURE REVIEW

There are number of tools available on the internet for calculating different software metrics for projects written in different programming languages. This chapter gives a brief introduction and specifies the limitations of some of the software metric tools that are related to the tool developed in this research. As mentioned in the previous chapter, the software metrics tool developed in this research calculates the metrics and statistics for C# source files only. Hence, the criteria on which the related tools are selected is that they should be able to calculate the software metrics for C# language. It is to be noted that some of the tools discussed here are also capable of calculating metrics for other programming languages. The list of the tools that can be used to calculate metrics for C# programming language include SLOCCount, SourceMonitor, NDepend, Code Counter Pro, SLOC Metrics, Resource Standard Metrics, SD C# Metrics, EZ – Metrix, McCabe IQ, CodeMetrics, and Visual Studio Code Metrics. A brief description for each of these tools is given in the following paragraphs.

List of Software Metric Tools

SLOCCount

SLOCCount [8] is a suite of programs for counting physical source lines of code in potentially large software systems. This tool was developed by David A. Wheeler, originally to count SLOC in a GNU/Linux distribution, but it can be used for counting the SLOC of arbitrary software systems. This tool has the ability to take a large list of files and then automatically categorize them using a number of different heuristics. It has some report-generating tools to collect the data generated and present it in several formats according to the requirement. This

tool also has the ability to automatically estimate the effort, time and money required to develop the software depending on the count of lines of code.

Since SLOCCount is an open source free software released under the GNU General Public License (GPL), it can be downloaded for free and modified according to the needs of the user. It can count physical SLOC for a wide number of languages like Ada, Assembly, C, C++, C#, Java etc. and can run on GNU/Linux, Apple Mac OS X and Windows operating systems.

However, the problem with this tool is that it only calculates the lines of code (LOC) metric and relies on that to estimate the time, effort, and money required to develop the software. It is not a web-based application and hence, a user need to download and install it before using or editing the tool and though the tool is capable of running on different operating systems, it is not easy to install or use it on Windows and is also much slower on Windows platform. Another limitation is that the tool does not support the calculation of statistics for the metrics in a given application.

SourceMonitor

SourceMonitor [9] is a freeware program that collects software metrics in a fast, single pass through the source files. It lets the user see how much code they have inside the source code and makes it easy to identify the relative complexity of different code modules. SourceMonitor is written using C++ programming language and can measure metrics which includes method and function level metrics for source code written in C, C++, and C #, Java, Delphi or HTML. This program can be operated within a standard Windows GUI or by using XML commands inside the scripts. It can display and print the results of metrics in tables and charts and also has the ability to export the results to XML (Extensible Markup Language) or CSV (comma separated value) files for further processing with other tools.

The problem with this tool is that it is not a web-based application and hence a user need to download and install the software before using it. The other issue is that the users are not permitted to modify or customize the software according to their needs even though the tool is free to download and use.

NDepend

NDepend [10] is a static analysis tool which is available as an extension for Visual Studio for .net managed code. It currently supports 82 code metrics which includes Lines of Code, Cyclomatic Complexity, Coupling, Nesting Depth, Rank, etc. and has more than 150 default code rules to check against best practices. It allows trend monitoring by displaying trend charts and visualizations for different code metrics. It has the ability to detect dependency cycles which can be eliminated to achieve higher code maintainability and can compare two versions of the code base to check the modifications and to warn the user if there are any incompatible changes in the code. It can generate reports with different details and analysis of the source code that can be used to keep track of the development process.

This is not a web-based application. The NDepend extension has to be downloaded and then installed for Visual Studio. This tool is not a freeware, but has a 14-day free trial version available. There is no way for the user to modify or customize the tool to add new modules that can calculate other metrics or statistics which are not originally calculated by the tool.

Code Counter Pro

Code Counter Pro [11] is a standalone program that counts all the programming lines, SLOC (source lines of code) and KLOC (thousands of lines of code) automatically, calculates the productivity of the software development team, measures Function Points through Backfiring and measures the percentage of comments in the source code. It can handle several types of

source code including that of C, C++, Java, C#, HTML, XML, and Pascal etc. It displays the result in a sortable grid that can be sorted by total lines, total comments, total blank lines, or total source code only lines depending on the requirement of the user and can then be saved to XML, HTML (HyperText Markup Language), CSV and XLS (spreadsheet file format) files directly from the program.

It is not a freeware tool but has a trial version available. It mainly concentrates on the lines of code metric, doesn't have the ability to calculate other metrics related to the source code and the users are not permitted to modify and customize the tool. It is not a web-based application and therefore, a user need to download, install and then use the user interface or the command-line interface to run the tool.

SLOC Metrics

SLOC Metrics [12] [13] is an application that measures the size of the source code based on the Physical Source Lines of Code metric recommended by the Software Engineering Institute at Carnegie Mellon University. According to the definition of this metric, the source lines that are included in the count are the executable statements, declarations, and/or compiler directives, excluding comments and blank lines from the count. It supports different programming languages including C, C++, C#, Java, HTML, etc. and the users are allowed to add additional definitions for other file types. The results generated can be formatted into a HTML, CSV or an XML file format which makes it easier to share the results and analyze the source code.

A user needs to download the application, install it and then can run it as a console application or by using the Windows-based graphical user interface. The full version of this tool is not free, however, a trial version is provided for the users.

Resource Standard Metrics

Resource Standard Metrics [14] is a commercial, source code metrics and quality analysis tool which is fast, flexible and an easy-to-use tool in the measurement of code quality and metrics. It supports any operating system while providing a standard method to analyze source code written in C, C++, C# and Java. It calculates a number of functional and object oriented metrics, has the ability to do that by function, class, file, and project, and has no limitations to the file length or the number of files. It has the ability to create an inheritance tree directly from the code, analyze source code for code style enforcement and can analyze class inheritance by depth and derivation. It can create reports in HTML with hyperlinks to the code, XML, CSV format for direct input to Microsoft Excel spreadsheets and in text format for screen, print and importing purposes.

It is not a freeware or an open-source software, but has a free version available for student projects and evaluation purposes with a restriction that it only supports metrics for up to 20 source files. This is not a web-based application and a user is not permitted to modify or add new modules to the application.

Semantic Designs C# Metrics

Semantic Designs C# Metrics [15] is a member of SD's family of language-specific metrics tools which include tools for COBOL, Java, Logix5000 and VBScript. It provides standard metrics down to the level of methods or procedures and summary rollups of larger units such as classes, files and directories. Some of the metrics calculated by this tool at module level includes source lines of code (SLOC), comment lines, number of methods, Cyclomatic complexity, maximum loop nesting, maximum conditional nesting, decision density, Software

Engineering Institute (SEI) maintainability index, class and file counts, etc. It can generate reports in text or XML formats depending on the requirement of the user.

It is not a freeware or a web-based tool. An evaluation copy is available for download, but with some limitations such as only a part of the results are displayed until a license is bought. A user is not permitted to customize the tool to add code for calculating new metrics or to calculate statistics using different metrics.

EZ – Metrix

EZ – Metrix [16] is a web-based source code counting utility which combines a consistent measurement of source code across different programming languages, platforms and operating systems with the portability of a hosted application. It supports more than 80 programming languages including Ada, Assembly, BASIC, C, C++, C#, Delphi, HTML, Java, JavaScript, jsp, Pascal, PHP, SQL, Python, Ruby etc. The metrics that are calculated for each file by this tool are Source Lines of Code, Comments and Blank Lines. It has the ability to compare two versions of a file or a group of files to determine the differences between them and generate a report with the results. Reports are generally stored on the server and can be displayed on the screen or can even be downloaded or exported to a Microsoft Excel compatible file format. It uses a flexible and configurable rules engine which defines how to differentiate lines of code from comments and blank lines and which can be modified according to the specific needs of the user.

Even though the tool is web-based, it still needs an application to be downloaded into the local computer that a user is using before calculating the metrics. This is not a free tool except for a 30-day free evaluation license. This tool calculates only the metrics related to the lines of code and does not have the ability to calculate other method-related or object oriented metrics.

McCabe IQ

McCabe IQ [17] is a quality management suite used to analyze the quality and test coverage of different applications. It uses advanced software metrics to identify, objectively measure and report on the complexity and quality of the code at the application and enterprise level. It is available in three editions, each suiting to a particular group or team in the software development process and are modified to suit their specific needs. The first is the Developers' edition which performs static analysis of the code to measure software quality, visualizes the architecture, and highlights the complex blocks of the code which makes it easier to identify bugs and security vulnerabilities, second is the Test Team edition which helps in providing complete test/code coverage and in measuring the time and resources needed to ensure a well-tested application, and the last one is the Enterprise edition which provides the functionalities of both the Developers' and Test Team editions along with other reporting and test data collection capabilities. It supports different programming languages including Ada, C, C++, C#, Java, COBOL, Perl, etc.

This is not a web-based or a free tool and is mainly used by large organizations because of its pricing. A user is not permitted to modify or customize the tool to calculate additional metrics or statistics.

CodeMetrics

CodeMetrics [18] is an add-in for Reflector, a .net decompiler, which analyzes and computes several code quality metrics on the assemblies. It calculates different metrics including Cyclomatic complexity, source lines of code, comments, number of local variables in a method, etc. and the results can be saved to a file. This tool and its source code are freely available to the users, but it is not a web-based tool.

Visual Studio Code Metrics

It [19] is a set of software measures that help teams to identify risks and track progress during software development process. Using these metrics, developers can identify the parts of the code that needs some modifications or simplifications and to know which of them are to be more thoroughly tested. The different software metrics calculated by this tool includes Maintainability Index which is a value calculated to represent the relative ease of maintaining the code ranging between 0 and 100, Cyclomatic Complexity which measures the structural complexity of the code, Depth of Inheritance which measures the class hierarchy, Class Coupling which measures the interdependencies between different modules of the code, and Lines of Code which is the approximate number of lines in the code. This tool is in-built in the Microsoft Visual Studio Ultimate and Premium editions.

Limitations of the Existing Applications

The software metric tools and applications available in the market are not completely web-based and hence the users need to download and install the application before using it for calculating the software metrics. Most of these applications are standard tools where users are not permitted to customize the tool or allowed to add new modules to calculate the metrics which are not included in the tool. The other problem with these tools is that most of them are not free to use and give only a small 30 day trail period for the users.

Proposed Tool and its Advantages

The software metrics tool proposed in this research is a web-based application to which users can upload the source files and calculate different software metrics and statistics on the

uploaded files. The users are provided with the ability to add new modules to the application to calculate additional metrics or statistics which are not originally calculated by the tool and this tool is totally free for all the users. Hence, these advantages makes the tool reliable, easy to use and unique when compared to the tools available in the market.

ARCHITECTURE

The Software metrics tool is an application that can be accessed by the users through a web browser. The architecture of the tool is a typical three-layered architecture which consists of presentation, business and data layers. Each layer is a logical grouping of software components decomposed from the design of the application. These layers are very useful in differentiating the functionalities of different components which makes it easier to create a design that supports reusability of components, helps in maximizing the maintainability of the code and to optimize the way the tool works under different circumstances [23].

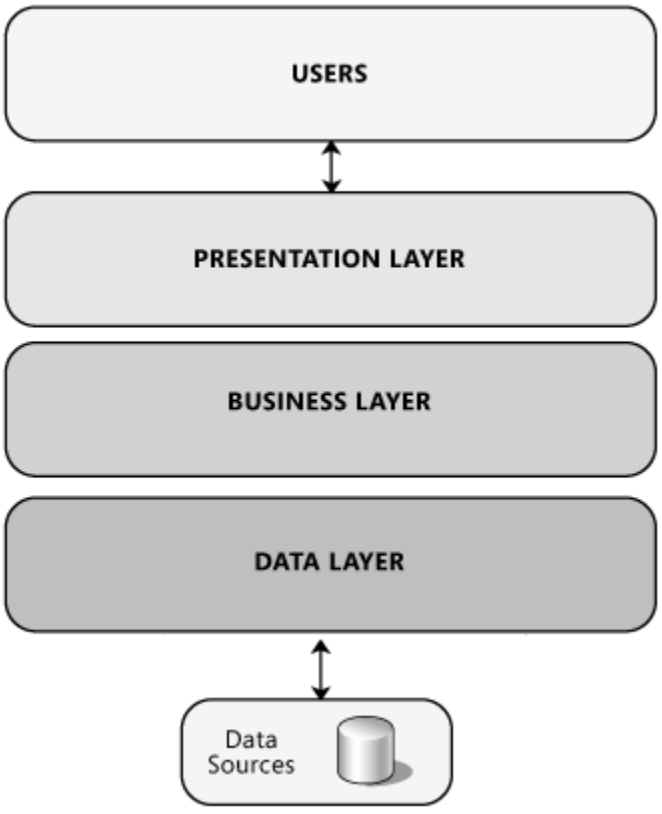

Figure 1: The layered architecture view of the tool

Layered Design of the Tool

The functionality of each layer in the three-layered design shown in Figure 1 is explained below.

Presentation Layer

This layer contains the components that implement and display the user interface and provides components for user input and for displaying the output. The presentation layer for this tool will have a simple user interface through which users can upload the source files for which the metrics and statistics are to be calculated, select the software metrics to be calculated, select the statistics to be calculated and view the results of the calculated metrics and statistics. A user will have access to add new modules to the tool for calculating new metrics or statistics by uploading a source file through the web user interface to the application.

Business Layer

The business layer typically implements the core functionality of the software application. It contains application logic which deals with the retrieval and processing of data, managing the user inputs and ensuring data consistency and validity. After the user uploads the data using the presentation layer components, the data is passed to the business layer where the source files are verified first and processed according to the requirement of the user by interacting with the other workflow components.

Data Layer

This layer contains the data access components which has the logic required to access the underlying data source. The data source for this tool is Microsoft SQL database which stores the information on the metrics and statistics calculated.

System Design

A Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing and documenting the artifacts of a software system. The UML mostly uses graphical notations to express the design of a software project which in turn helps to explore potential designs and validate the architectural design of the software. According to the UML specification, Structure diagrams and Behavior diagrams are the two major types of UML diagrams.

Structure diagrams show the static structure of the system and its components on different abstraction and implementation levels and their relation with each other whereas Behavior diagrams show the dynamic behavior of the objects in a system which can be described as a series of changes to the system over time [24].

The Use-case diagram, a type of Behavior diagram, for the software metrics tool is shown in the figure below. This diagram describes a set of actions (use cases) that this tool (subject) can perform in collaboration with one or more external users of the tool (actors) to provide some meaningful output to the users of the tool.

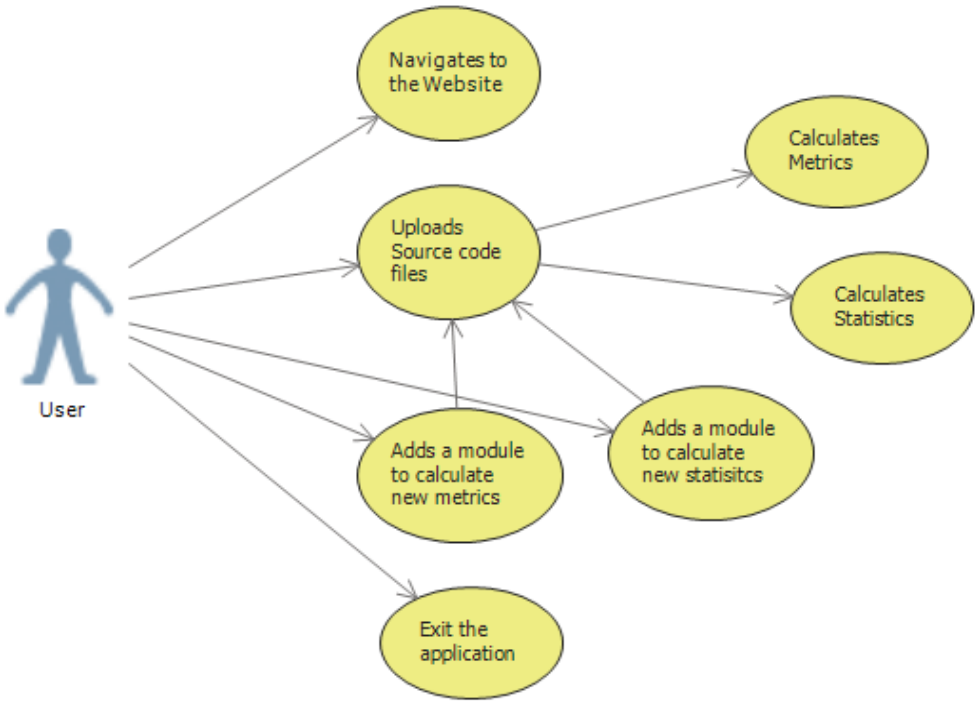

Figure 2: The UML Use-case diagram of the tool

The Class diagram, a type of Structure diagram, for the software metrics tool is shown in the figure below. The purpose of this diagram is to show the structure of the application as related classes and interfaces and to show the relationships between them.

The UML Use-case diagram and the Class diagram are generated using the Microsoft Visual Studio 2013.

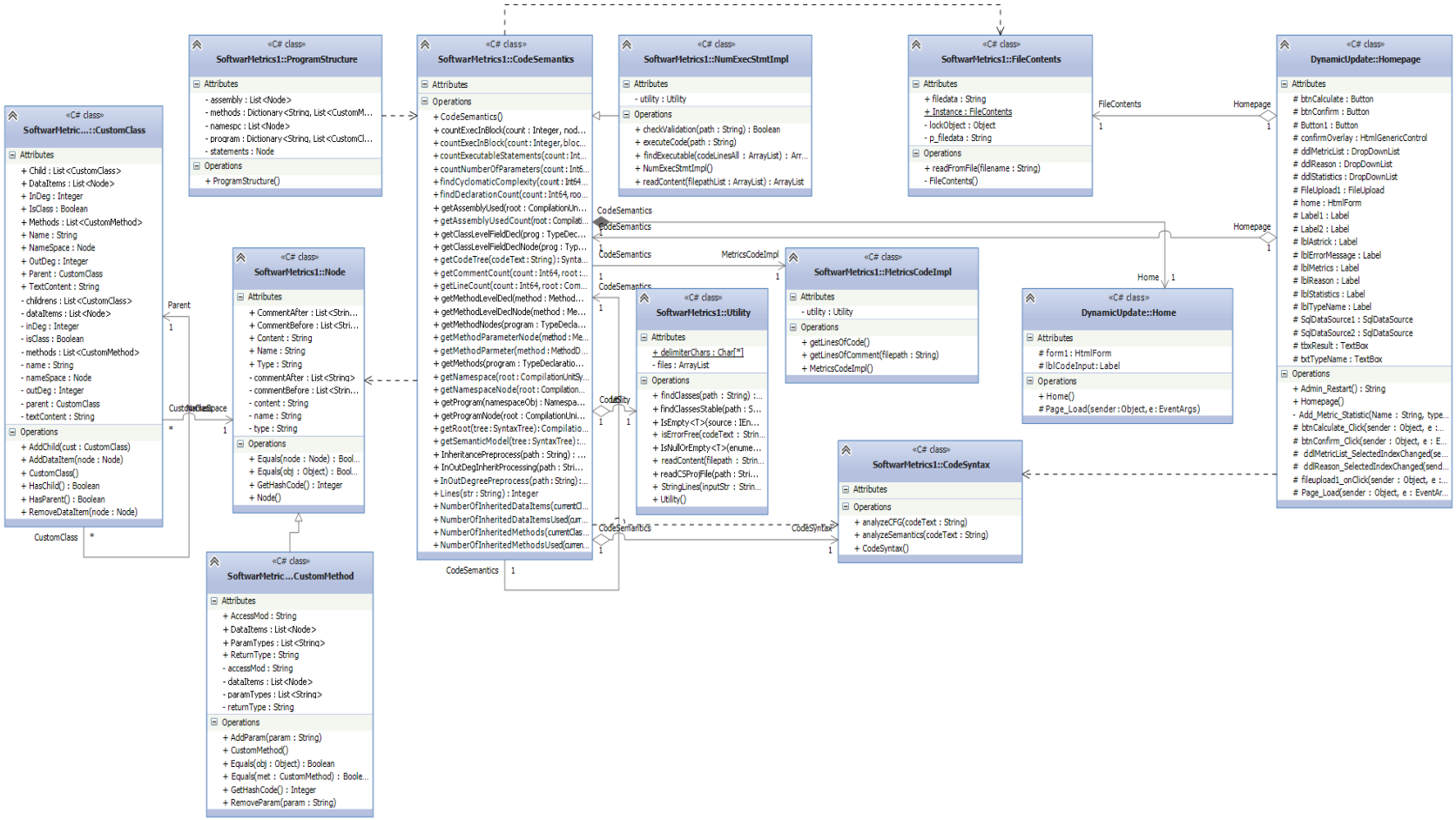

Figure 3: The UML Class diagram of the tool

The figure below shows the class dependency graph of the software metrics tool.

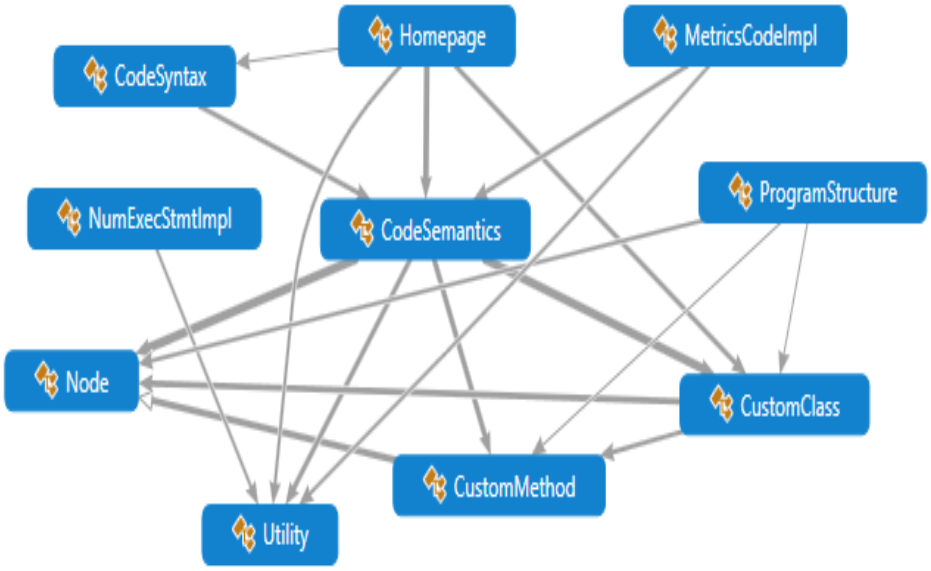

Figure 4: Class dependency graph of the tool

Software Development Methodology

A software development methodology is typically a framework that is used to structure, plan and control the process of developing a software application. The development methodology that we have used for this application is Scrum, which is an agile software development methodology that focuses on developing a software in short iterations and reevaluating the project priorities at the end of every iteration. The main emphasis is on developing a working application with some progress compared to the previous iteration.

This development method was chosen because it improves the overall quality of the application by enabling regular testing of the application and giving an early insight if an issue arises in the code. It also ensures that the changes are taken care at any point in the development process without the need to make major changes to the application.

The important thing I have learnt using this method is that it requires a lot of commitment from everyone involved because each iteration has new tasks to be completed in a small period of time as well as some backlog tasks that were not completed in the previous iteration. It is also more enjoyable as it requires active involvement, cooperation and working on new tasks in every iteration.

RESEARCH APPROACH

The architecture of the software metrics tool developed in this research is designed in such a way that it provides a simple web-based user interface in the frontend for the users to select the software metrics and statistics that are to be calculated and provides a functionality to add or upload the source files which can be used to calculate the metrics and statistics or can be specified as modules that are to be added to the tool for calculating new metrics or statistics. The backend uses the Roslyn API which is a .net compiler API to parse the source files and facilitates the calculation of the required metrics from the source code. A database is also connected to the application which keeps track of the newly added modules to the tool. The software metrics application is developed in Microsoft Visual Studio using C# programming language. It is designed to be a language-dependent tool which accepts and calculates metrics and statistics for C# source files only.

The first important step in designing a tool that calculates software metrics is researching the different definitions available for the metrics and deciding on an appropriate definition for implementation. The definitions of different software metrics that are implemented in this tool are described in the following lines.

Software Metrics

Lines of Code

For this metric, the most widely accepted definition is used which defines an LOC (Lines of Code) as a count of all the statements in a program except for comments and blank lines. This count includes all the lines in a program including headers, declarations, executable and non-

executable statements. According to Grady and Caswell, this definition is defined by Hewlett-Packard [20].

Lines of Comments

This metric calculates the count of all the comments in a given source code file. In C# programming language, a single line comment is preceded by a double forward slash, an XML tag comment is preceded by three forward slashes and a multiple line comment is enclosed in between /* and */.

Number of Declarations

A declaration, in general specifies the property of an identifier and is commonly used for classes, variables, constants and methods. This metric calculates the total number of declarations which include declarations of namespace, classes, methods and variables present at different levels in a program.

Number of Class Level Declarations

This metric calculates the total number of declarations at a class level for a given source file.

Number of Method Level Declarations

This metric calculates the total number of declarations at a method level for a given source file.

Cyclomatic Complexity

This metric uses the NDepend tool's definition [21] which defines Cyclomatic Complexity as the sum of if, while, for, foreach, case, default, continue, goto, method declarations, class declarations, AND operator (&&), OR operator (||), catch, ternary operator (?:), and ?? operator expressions found in the code. The else, do, switch, try, using, throw,

finally, return, object creation, method call, and field access expressions are not counted for this metric.

Number of Parameters

This metric calculates the total number of parameters or arguments passed to different methods in the source file.

In-degree

This metric measures how much a class is used by other classes in the project. It is the total number of calls to the functions or methods inside a class.

Out-degree

This metric is defined as the total number of function calls from a class to the functions present in other class.

Number of Inherited Data-items

This metric calculates the total number of data-items inherited by the class.

Number of Inherited Data-items Actually Used

This metric calculates the total number of data-items used out of all the items inherited by the class.

Number of Inherited Methods

This metric calculates the total number of methods inherited by a class.

Number of Inherited Methods Actually Used

This metric calculates the number of methods used out of all the methods inherited by the class.

Number of Include Statements

This metric calculates the total number of include statements in a given source file.

Statistics

The definitions of different statistics implemented in this tool are described in the following lines.

Average

This statistic uses the standard definition of Average which defines it as the sum of a list of numbers divided by the total number of numbers in the list. In this tool, the average value is calculated by adding up the results of a specific metric for each file and dividing it by the total number of files in the project.

Max

This statistic displays the maximum value of a specified software metric from the list of all the results for a given project or a set of source files.

Range

The range of a set of data is defined as the difference between the largest and smallest values. The range of a software metric in a given project is calculated by subtracting the lowest value of all the results for that metric from the highest value.

The next step after finalizing the software metric definitions is to find appropriate programming language and researching different available API's that can help in implementing the software metrics. The programming language chosen for implementing the tool is C# which makes use of the Roslyn API to facilitate the calculation of different software metrics. A brief description of the Roslyn project is given in the following paragraphs.

Roslyn APIs

A compiler generally builds up a deeper understanding of the code it is processing, but that knowledge is not available to anyone. For this reason, Microsoft has developed the Roslyn project [22] which shares the information compilers have about the code with the end users, hence making them services instead of just opaque source-code-in and object-code-out translators.

The Roslyn project exposes the C# and Visual Basic compiler's code analysis to the users by providing an API layer that mirrors a traditional compiler pipeline. There are four important phases in a compiler pipeline and the first of those is parse phase where source code is tokenized and parsed into a syntax tree that follows the language grammar, next is the declaration phase where declarations from source and imported metadata are analyzed to form named symbols, next phase is the bind phase where identifiers in the code are matched to symbols, and the last phase is the emit phase where all the information built up by the compiler is emitted as an assembly code. The figure below illustrates the different phases of a compiler in the particular order of their execution from left to right.

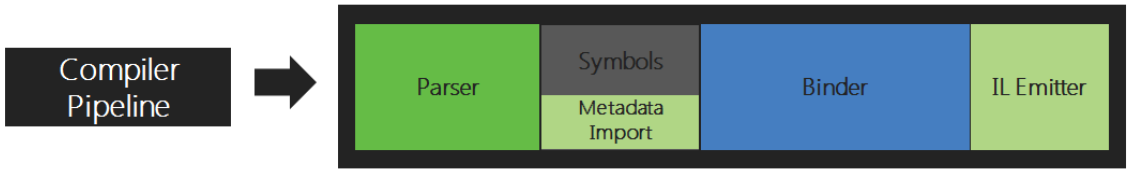

Figure 5: Phases in a compiler pipeline. Source: The Roslyn project documentation [22]

For each of the different phases in the compiler pipeline, Roslyn surfaces an object model that allows access to the information at that phase. For instance, the parsing phase is surfaced or exposed as a syntax tree, the declaration phase is exposed as a hierarchical symbol tree, the binding phase is exposed as APIs that show the results of compiler's semantic analysis and

finally the emit phase is exposed as an API that produces Intermediate Language (IL) bytecodes. The figure below illustrates the different APIs provided for different phases of a compiler.

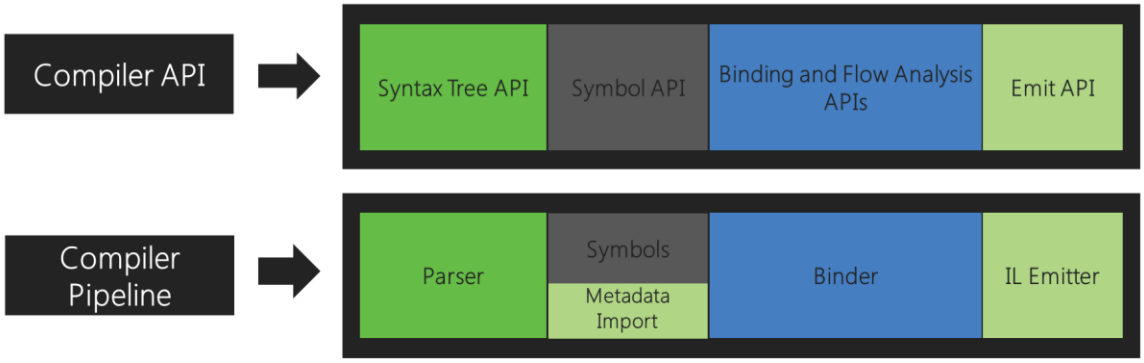

Figure 6: APIs corresponding to different phases of a compiler. Source: The Roslyn project documentation [22]

The most important data structure exposed by the Roslyn APIs is the syntax tree which represents the lexical and syntactic structure of the source code. The syntax tree is the primary structure used for code analysis, binding, refactoring and code generation as it contains every piece of information regarding the source code.

The final step in the research is to develop the tool using the metric definitions and the Roslyn API. Initially, a console-based application is developed in C# which calculates the different metrics and statistics for a given set of source files. It was initially developed as a console-based application to minimize the complexity of development and also because it is easy to test and debug a console-based application compared to a web-based one. Later, it is converted to a web-based application with a simple User interface allowing users to select the metrics and statistics to be computed, providing the functionality to add or upload source files, and also the ability to add new modules to the tool.

Extensibility

Extensibility in software engineering can be defined as the ability of a software application or a program to allow users to expand or add new functionalities to it. The fundamental goal of extensibility is to provide the ability to change while minimizing the impact on the existing system i.e. it should not be necessary to change the original source code while adding a new functionality to the application. Since software applications need to evolve with time and new requirements arise frequently, extensibility enables developers or users to expand or add to the software's capabilities and facilitates systematic reuse [25].

The importance of extensibility is growing over years because adaptable software components are very necessary as they have to fit into different scenarios that the users are using them for. Therefore, building an independently extensible software system is very important and also a challenge for the developers.

Classification of Extensibility Mechanisms

There are three different types of software extensibility based on the way the software components are allowed to change. They are White-box extensibility, Black-box extensibility and Gray-box extensibility [25] [26].

White-Box Extensibility

In this type of extensibility, a software system can be extended by modifying or adding to the source code. It is the most flexible form of extensibility and is sub-divided into two types: Open-box extensibility and Glass-box extensibility.

- *Open-Box Extensibility*: In this type of extensibility, changes are directly made to the original source code. For this, the entire source code should be available for the users and the license of the application should permit source code modifications. The main

disadvantage of this extensibility is that if the source code or the way a system works is not well understood, then changing the source code may become an error-prone and tedious activity.

- *Glass-Box Extensibility*: In this type of extensibility, the source code is available to the users, but they are not allowed to modify it directly. Users can use the source code, develop their extensions separately in a way that it does not affect the original system. It is easy to understand and maintain new modules or extensions as well as the original system as they are cleanly separated using this extensibility. The other advantage of glass-box extensibility is that it is less likely that the extension process introduces bugs in the original system because the changes are not directly implemented in the source code. Object-oriented application frameworks use inheritance and dynamic binding features to achieve glass-box extensibility.

Black-Box Extensibility

In this type of extensibility, the internal details about the system's architecture and implementation are not available to the users, instead they have to use the interface specification for extending the system. Black-box extensions are typically made through system configuration applications or through the use of application-specific scripting languages and in the case of object-oriented frameworks, it is achieved by defining interfaces for components that can be plugged in to the framework.

Gray-Box Extensibility

This extensibility represents a compromise between a pure white-box and a pure black-box approach as it does not rely fully on the exposure of source code. Users are provided with a

more abstract documentation of the system's specialization interface which provides the internal details about the implementation and extension of the system.

Approach for Adding a Metric/Statistic to the Application

The extensibility provided by this tool is a type of glass-box extensibility where the users have the ability to write their own modules to calculate metrics or statistics by using the methods and classes that are already present in the source code of the application. Modifications are not permitted directly in the source code, instead users need to upload the new source file to the tool using the web user-interface of the tool.

There are many useful methods defined in the original source code that can be reused by the user to calculate new metrics or statistics. Some of the important methods available to the user are defined below:

- *findClasses:* This method finds all the relevant classes at the given path and returns the result in an ArrayList.
- *getCodeTree:* This method generates a syntax tree based on the code provided and returns it.
- *getRoot:* This method finds the root node of the syntax tree and returns it.
- *getSemanticModel:* This method returns a semantic model which contains all the information about the source file.
- *getAssemblyUsed:* It returns a list containing the assemblies used in the source file in terms of custom Node class.
- *getNamespaceNode:* This method returns the namespace node in terms of custom Node object.

- *getProgram:* It returns the list of classes declared in the source file in terms of a list of Roslyn API objects.
- *getProgramNode:* This method returns the list of classes declared in the source file in terms of a list of custom node objects.
- *getMethods:* It returns the list of methods declared in a class in terms of a list of Roslyn API objects.
- *getMethodNodes:* It returns the list of methods declared in a class in terms of a list of custom node objects.
- *getClassLevelFieldDecl:* It returns a list of variable declarations in a class in terms of a list of Roslyn API objects.
- *getClassLevelFieldDeclNode:* It returns a list of variable declarations in a class in terms of a list of custom node objects.
- *getMethodLevelDecl:* This method returns a list of variable declarations in a class in terms of a list of Roslyn API objects.
- *getMethodLevelDeclNode:* This method returns a list of variable declarations in a class in terms of a list of custom node objects.
- *getLineCount:* This method calculates the lines of code in a class according to its definition.
- *getCommentCount:* This method calculates the total lines of comments in a class.
- *findDeclarationCount:* This method calculates the total number of declarations in a class.
- *findCyclomaticComplexity:* This method calculates the Cyclomatic complexity for the class.

Using these methods, a user can write a new module for calculating other metrics/statistics which are not calculated by the tool, upload the new module and use it for calculating the respective metric/statistic. For example, if a user wants the tool to calculate the number of blank lines in a source file, a new method can be written which makes use of the getLineCount and getCommentCount methods to get those values and then subtracting them from the total number of lines in the file gives the result of number of blank lines metric. In this way, a number of useful metrics or statistics can be developed and added to the tool according to the requirements of the user.

After writing or developing a new module, it can be uploaded to the tool using the 'upload' option in the web user-interface of the tool. The tool only accepts and uploads the files with .cs extension. After a file is selected to upload, the user is asked to select the file type from the drop-down list with options: Metric, Statistic and Others. If the file selected contains code to calculate a new metric, the 'Metric' option has to be selected, if it contains code to calculate a new statistic, the 'Statistic' option has to be selected and if it is a file for which the metrics have to calculated, the 'Others' option has to be selected. If a 'Metric' or 'Statistic' option is selected, then it prompts the user to enter the name of the metric or statistic respectively and this name field is a required field which throws an error if a user continues without entering a name for the metric or statistic. After entering the name of the metric or statistic, the name is added to the respective drop-down list in the homepage of the application and file is uploaded to the source code. Then, the user can use the new metric/statistic by just selecting the metric/statistic name from their respective drop-down list and clicking the 'Calculate' button.

RESEARCH EVALUATION

The software metrics tool developed in this research is a web-based application which has the ability to calculate some of the standard software metrics like the number of executable statements, in-degree, out-degree, number of parameters, number of declarations, number of inherited data items, number of inherited data items actually used, Cyclomatic complexity, etc., computes the statistics like average, maximum and range of each metric in the source code and provides an option to the user to add additional metrics and statistical calculations at any time to the application by uploading a new module through the API provided by the tool. The web user interface (UI) and the different operations supported by the tool are demonstrated in the following paragraphs.

The following figure is the screenshot of the user interface for the software metrics tool. It is the homepage of the application supporting the functionality to upload source files, select metrics and statistics from their respective drop-down lists, and an output area to show the results.

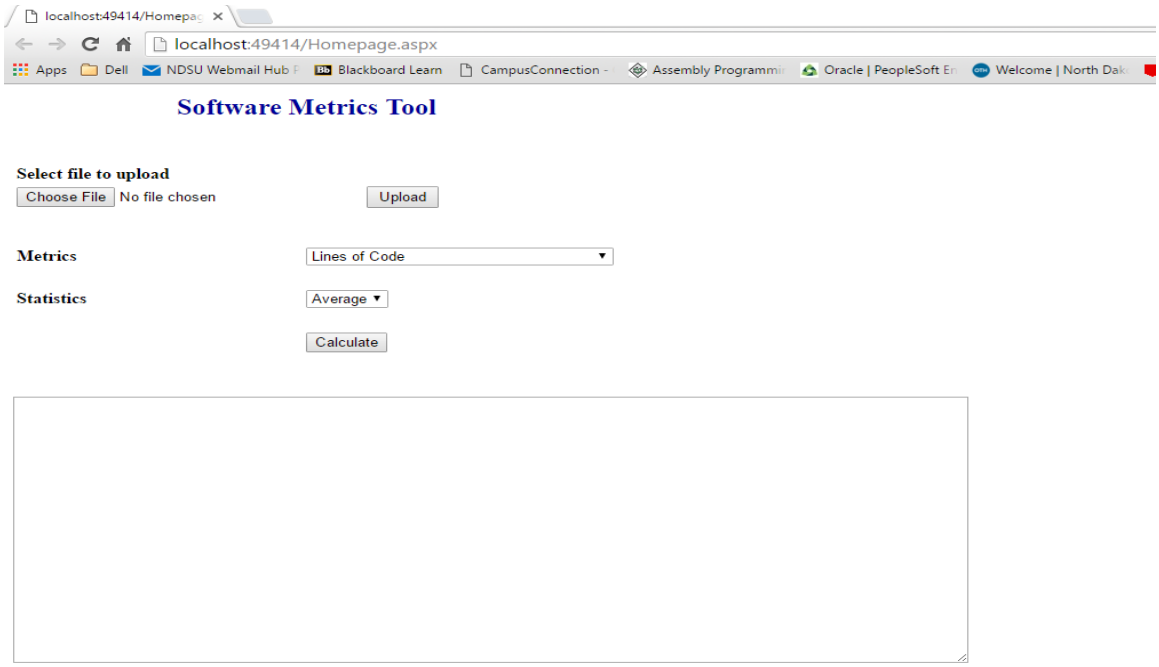

Figure 7: Homepage of the software metrics tool

The following screenshots are the elaborations of the drop-down lists for metrics and statistics in the tool.

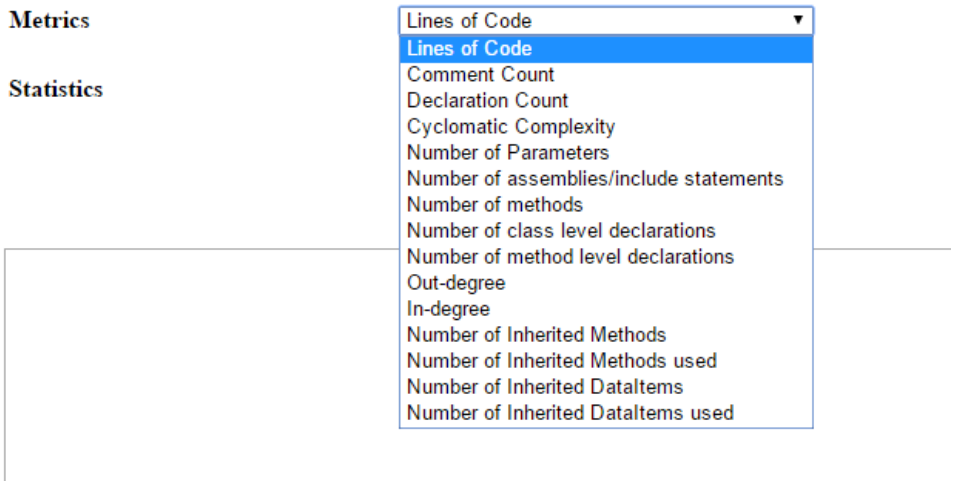

Figure 8: Drop-down list for Metrics listing all the metrics calculated by the tool

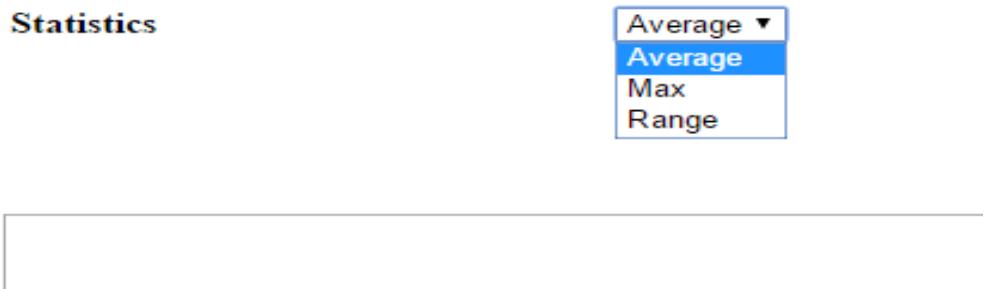

Figure 9: Drop-down list for Statistics listing all the statistics calculated by the tool

The following screenshots demonstrate the steps involved in uploading a source file or a new module for a metric or statistic through the user interface of the software metrics tool.

To upload a file, the 'Choose File' button on the homepage needs to be clicked, then it opens a search window to select the file to be uploaded.

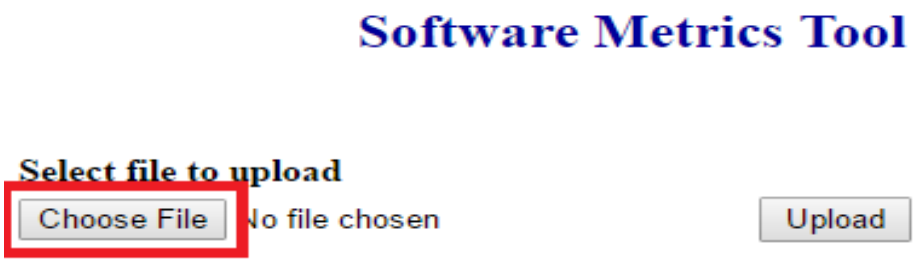

Figure 10: 'Choose File' button on the homepage of the tool

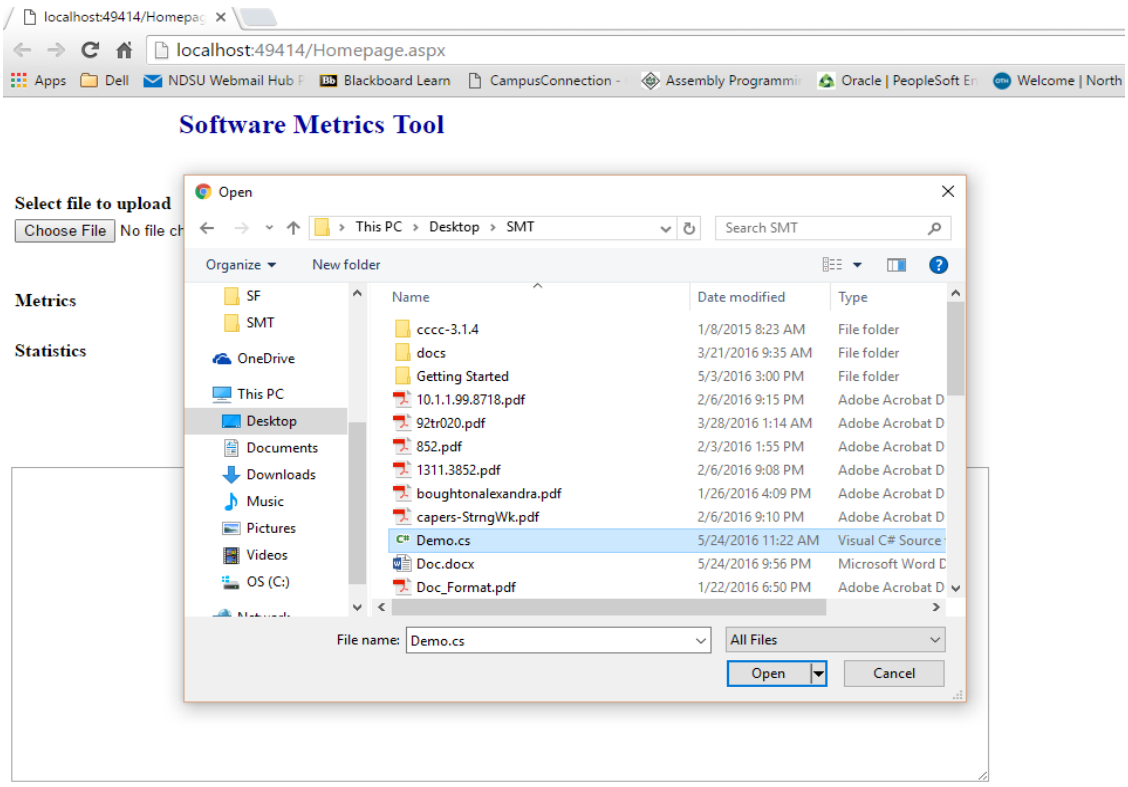

Figure 11: Window to search and select the file to upload

After selecting the file and clicking the ‘Upload’ button, the tool validates the file to verify its extension and displays an error message on the page if the uploaded file has any other unsupported extension. This tool accepts only the files with .cs extension (C# source files).

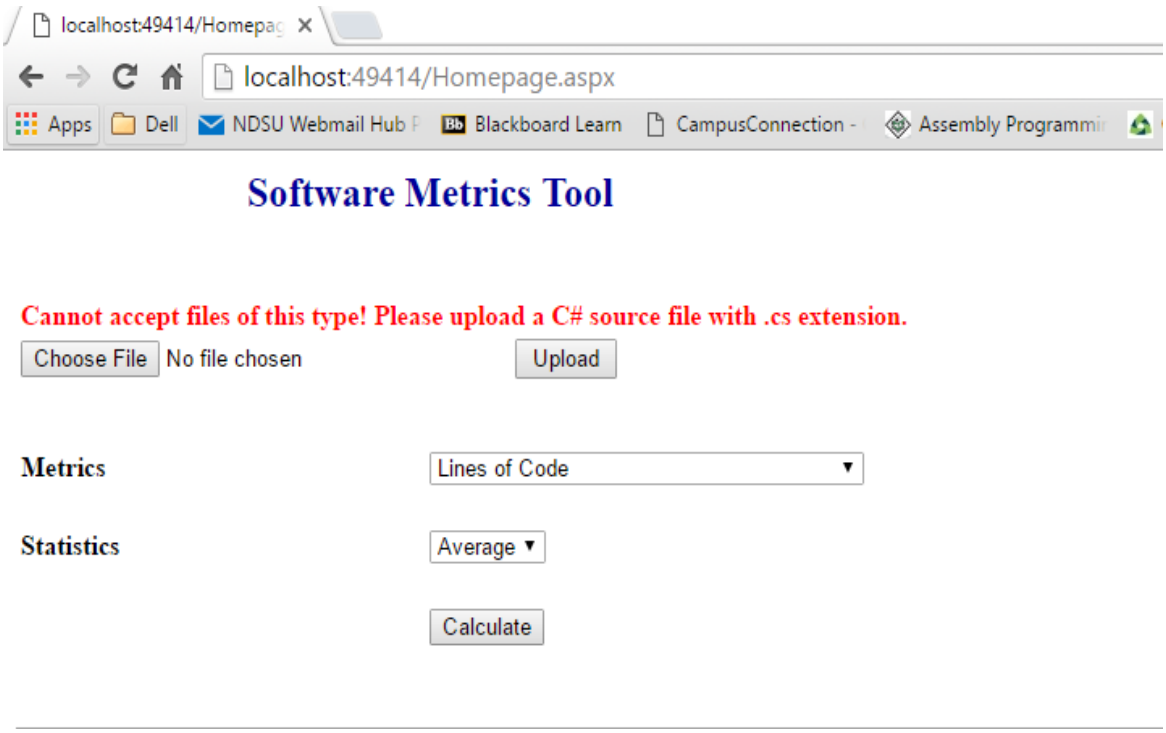

Figure 12: Error message when an unsupported file is uploaded

After verifying the extension of the file, the tool prompts the user to select the file type to determine if it is a file for which the metrics are to be calculated or if it is a new module for calculating a metric or a statistic. If the uploaded file is a module for a new metric or statistic, it prompts the user to enter the name of the metric or statistic so that the name can be added to the appropriate drop-down list and also to the database. The name of the metric or statistic is a required field and the tool throws an error message if it is not entered by the user. If the uploaded file is a file for which the metrics and statistics are to be calculated, the user needs to select the 'Others' option in the file type.

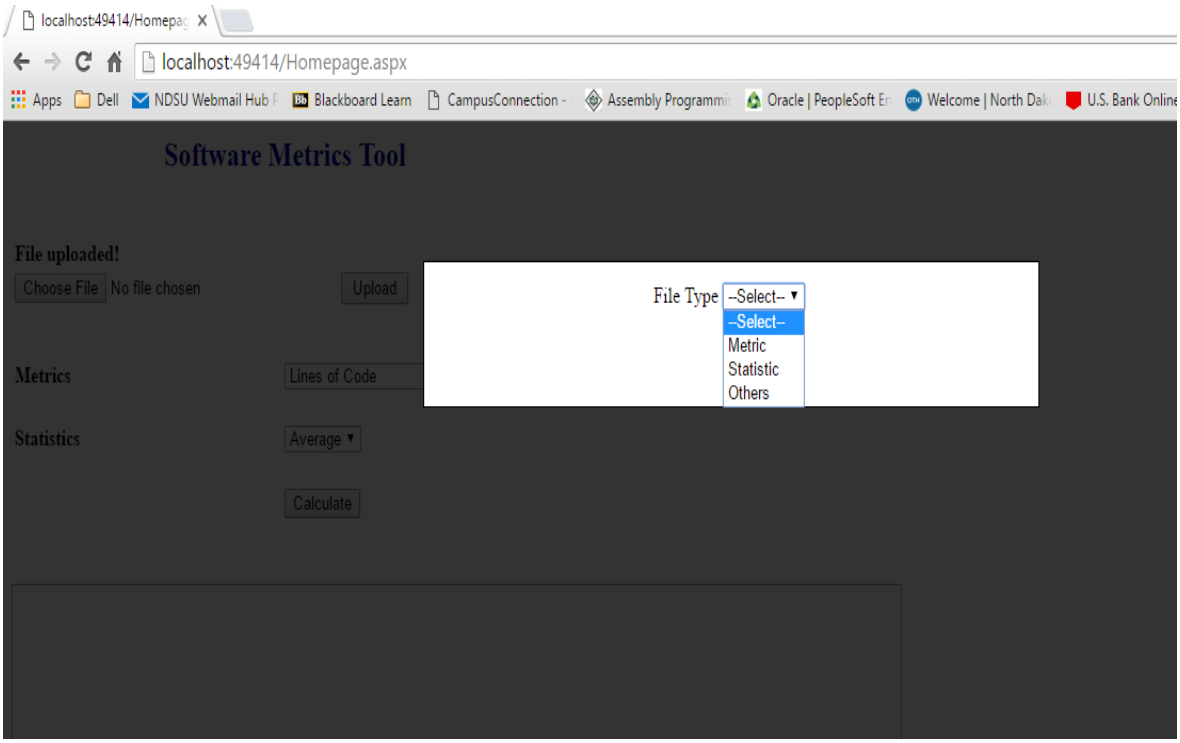

Figure 13: Overlay screen to select the file type

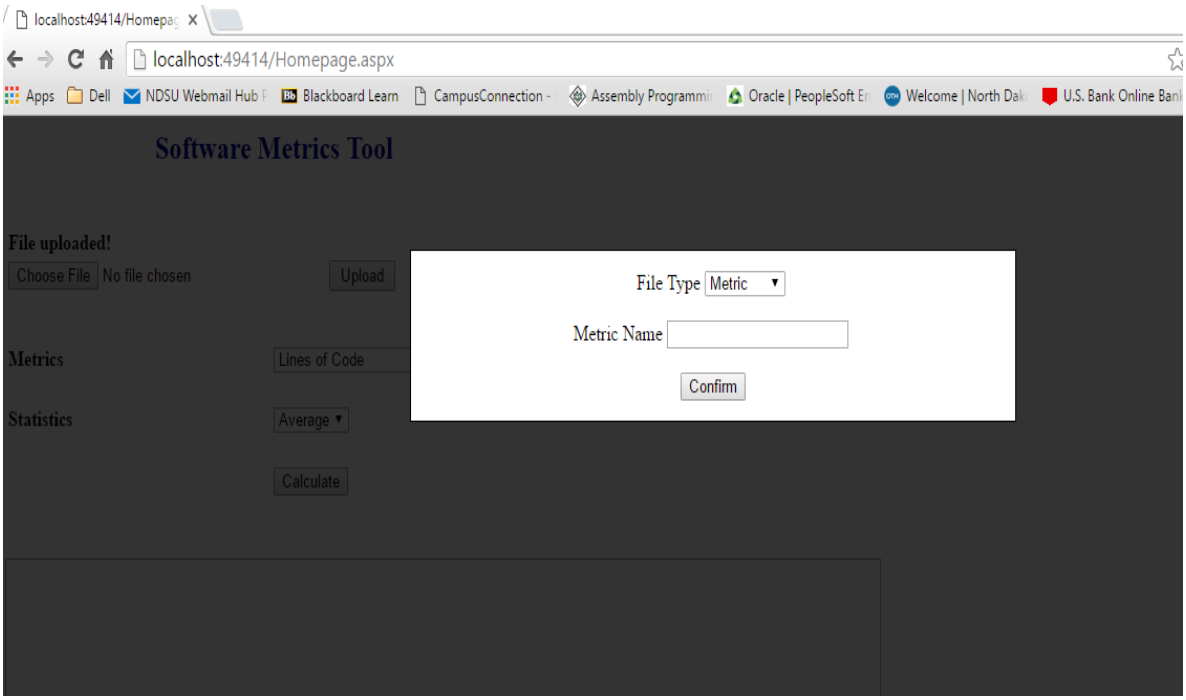

Figure 14: Screen prompting the user to enter the name of the metric when the file type is selected as 'Metric'

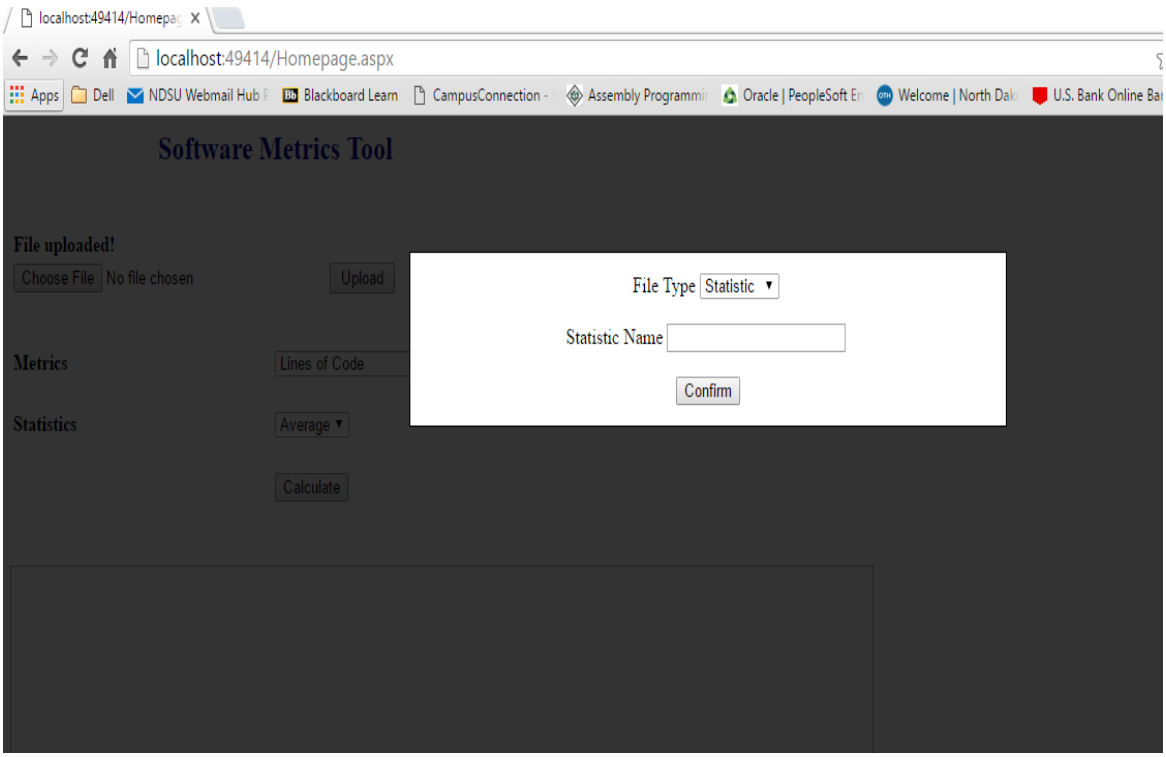

Figure 15: Screen prompting the user to enter the name of the statistic when the file type is selected as 'Statistic'

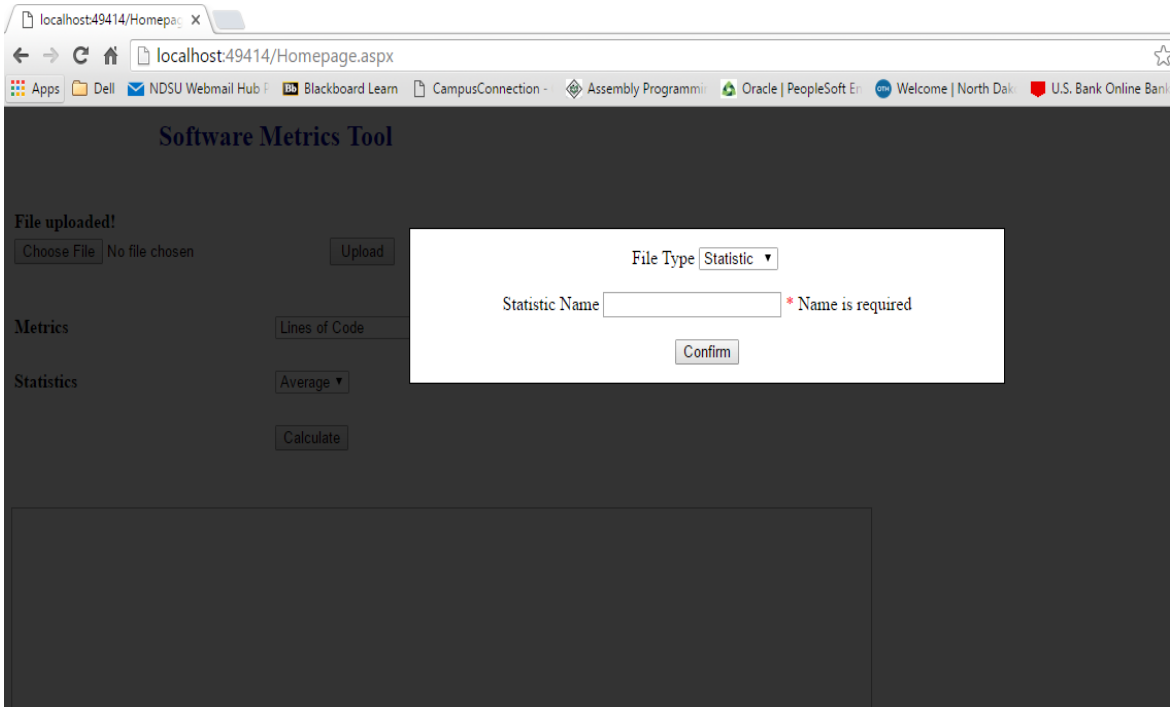

Figure 16: Screen prompting that the name is a required field

On clicking the ‘Confirm’ button after entering a name for the metric or the statistic, the file is uploaded and the name of the metric or statistic is added to the respective drop-down list in the home page of the tool and also to the database table in the backend. For the demo purpose, a metric and a statistic are added and the metric name is given as ‘Demo Metric’ and the statistic name is given as ‘Demo Statistic’.

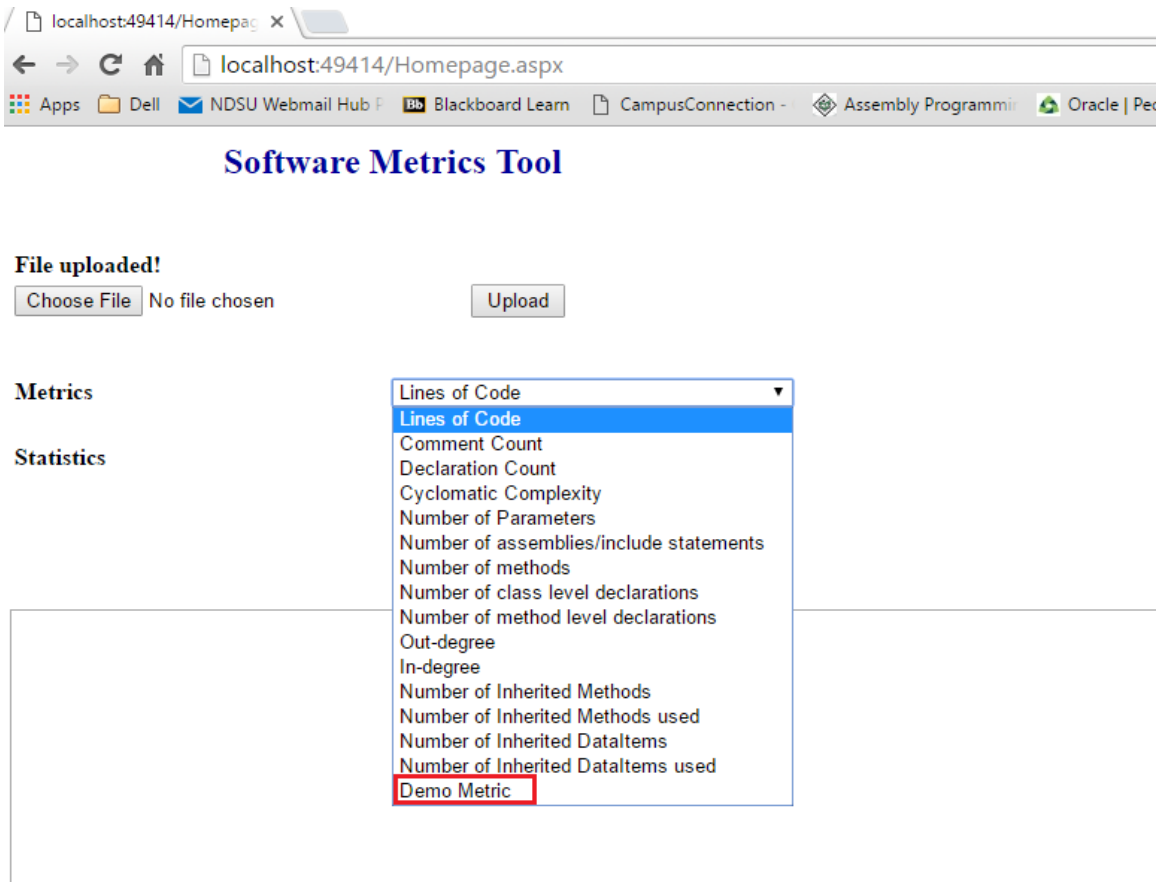

Figure 17: The new metric uploaded with the name ‘Demo Metric’ is added to the ‘Metrics’ drop-down list in the homepage of the tool

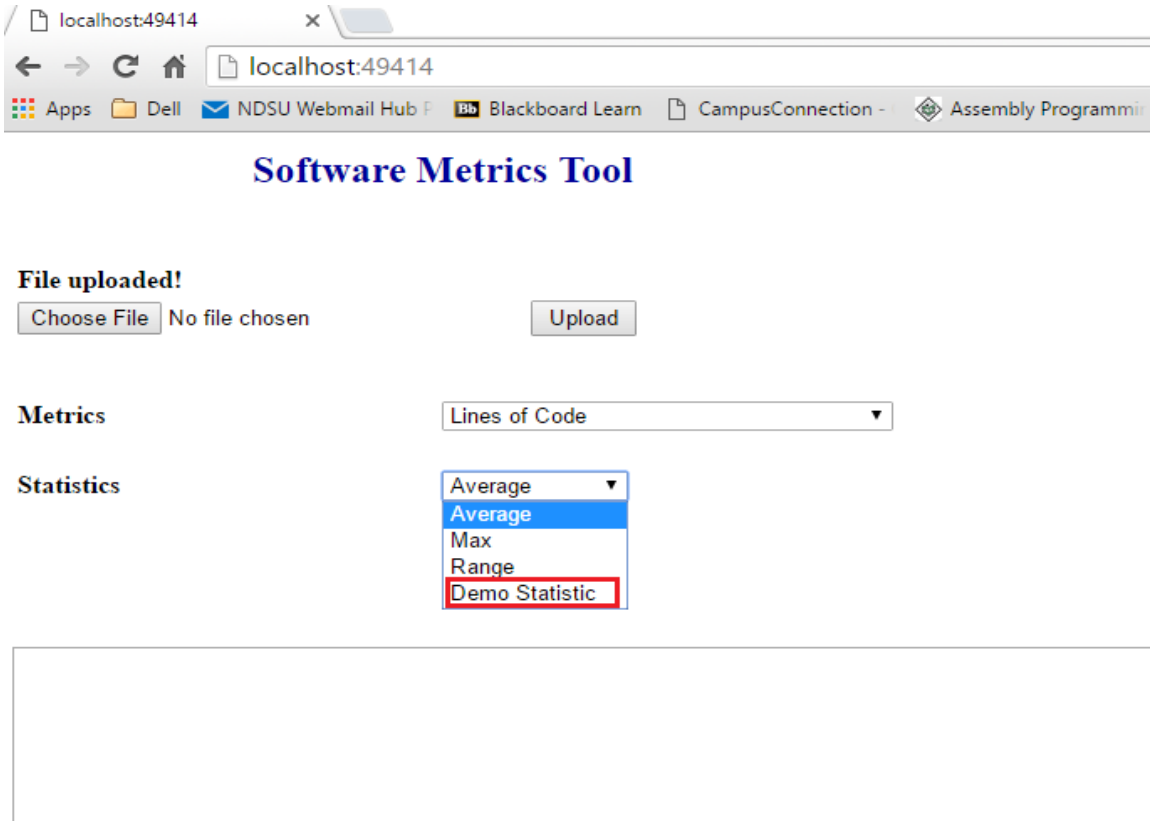

Figure 18: The new statistic uploaded with the name 'Demo Statistic' is added to the 'Statistics' drop-down list in the homepage of the tool

Finally, to calculate different metrics and statistics for the files, select the appropriate metrics and statistics from the drop-down lists and click on the 'Calculate' button, after which the results are displayed in the text area of the home page. The results area is scrollable which enables the user to scroll through all the results generated by the tool.

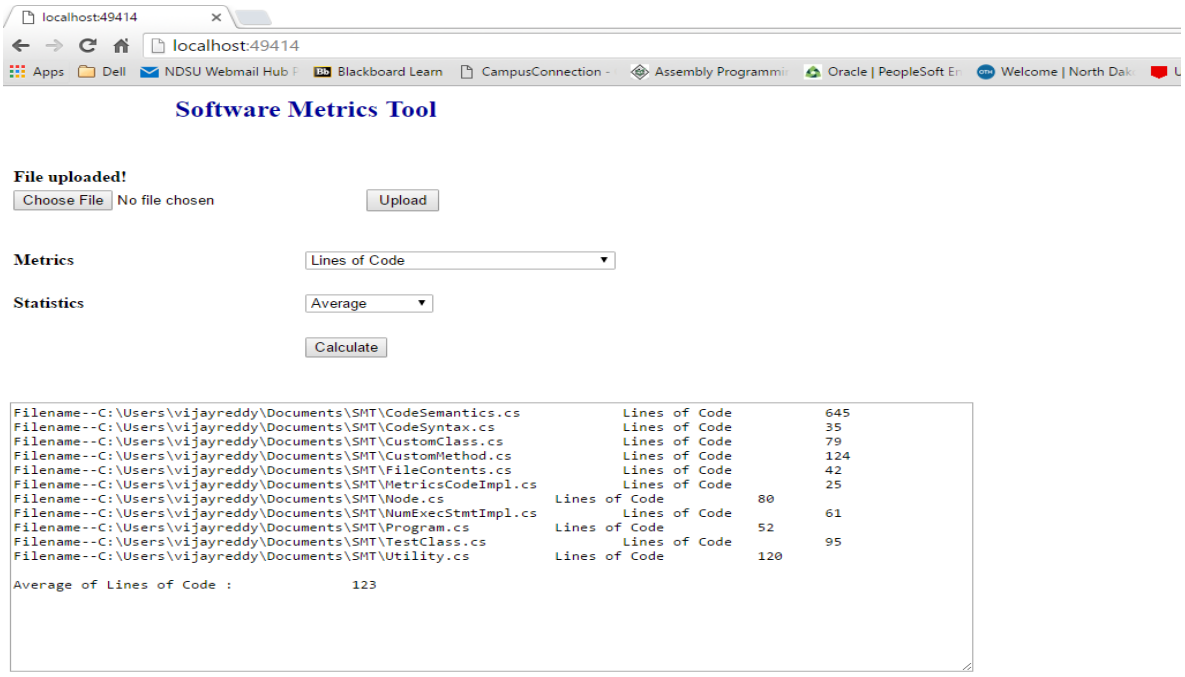

Figure 19: Results displayed in the text area with the file names, metric ('Lines of Code' in this case) and statistic ('Average' in this case) calculated for each of those files

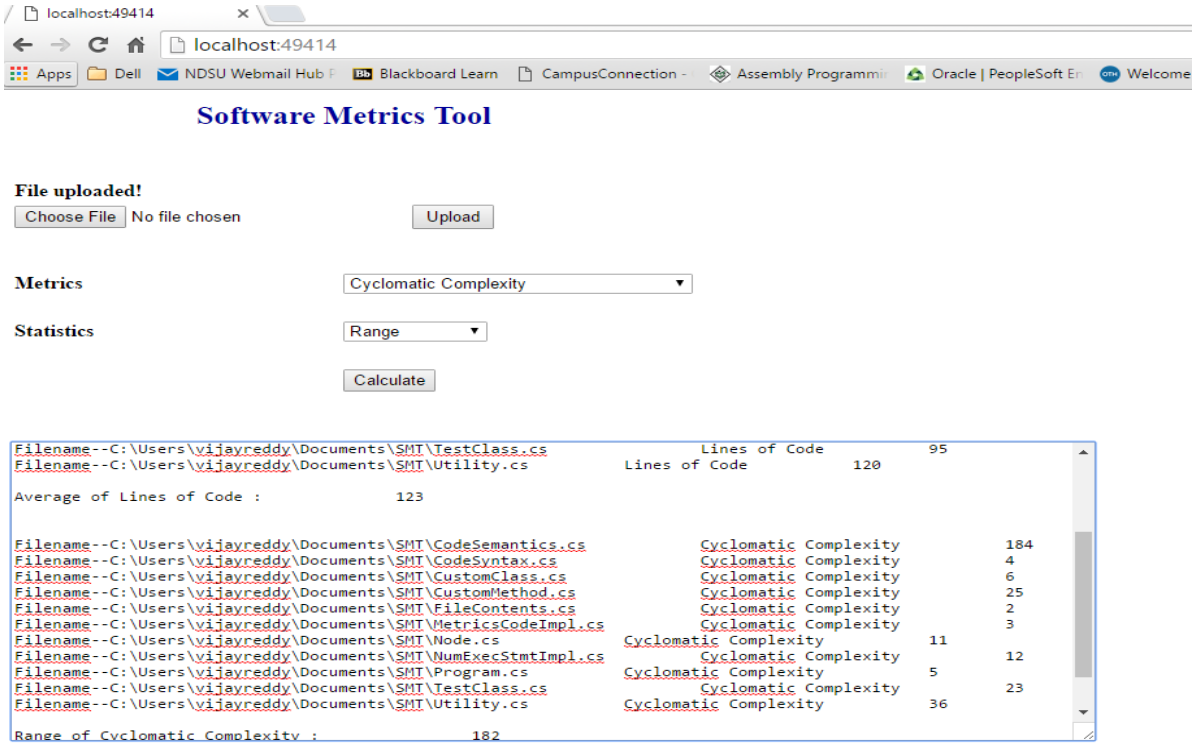

Figure 20: Results after multiple metrics and statistics are selected

Testing the Application

The web-based software metrics tool developed through this research is tested using different open-source C# projects available in the internet [27] [28]. All the projects used for evaluation are free to download from the internet. The tool is tested by randomly selecting different metrics and statistics for each of the project. Generally, each project is of a different size i.e. the number of C# source files in each of them varies and in this case, the highest number of C# files tested in a project are 245 and the total number of C# files combined in all the tested projects are 1777. The project names and the number of C# source files in each of them are shown in the table below.

Table 1: Name of the Project with the number of files in each of them

Name of the Project	***Number of C# files***
Customer Relationship Management System	245
School Management System	183
Pharmacy Information Management System	168
Gym Management System	123
Point of Sale	117
Employees Management System	91
Kiddie Care	88
Recruitment Process System	88
RTA Information System	82
Network Load Balancing System	66
Peer to Peer File Sharing System	56
Dental Management System	49

Table 1: Name of the Project with the number of files in each of them (continued)

Name of the Project	***Number of C# files***
Banking System	46
Blood Inventory Management System	43
Online Discuss Forum	42
Grade Book Application	38
Riverside Parts Master Project	38
Project Management System	36
Database Enterprise Manager	31
Sales and Inventory System	27
Knowledge Processing Application	26
Data Hiding Application	22
Network Monitoring System	19
Password Generator	16
Count-down Timer	12
Battery Information Application	10
OOP Login	9
Calculator	6
Total	***1777***

The tool worked successfully for the above projects by calculating different metrics and statistics.

CONCLUSION

A software metric, as discussed in the earlier chapters, is the measure of a particular property or characteristic of a program's performance or efficiency and the main objective for calculating a software metric is to obtain reproducible and quantifiable measurements which can be used to predict defects in the code, optimize the performance, schedule planning, quality assurance and cost estimation. There are many tools available in the market or on the internet like the SourceMonitor, NDepend, SLOC Metrics, Code Counter Pro, SD C# Metrics, McCabe IQ, Code Metrics etc., which are capable of calculating software metrics for applications developed using C# programming language, while some of them even support multiple programming languages, but, none of them is a web-based tool which makes it inevitable to download and install the tool first before using it and some of them even require a license to be bought by the user before using the tool to calculate the software metrics. The other issue with these tools is that the software metric values calculated by them are sometimes different from each other as each of them may use a different definition for calculating a metric and they only support a limited number of metrics which makes it hard for the user to use different tools for calculating metrics for different software components.

Therefore, to overcome some of these problems, a web-based software metrics tool is developed through this research which has the ability to calculate software metrics and statistics without the need to download or install any software and also allows addition of new modules for calculating metrics and statistics. It can be used to calculate some of the standard metrics like In-degree, Out-degree, number of declarations, number of executable statements, number of parameters, number of inherited data items, number of inherited methods, number of comments, Cyclomatic complexity etc., and statistics like the average, maximum and range of each metric in

the application. These features make this tool unique and easy to use when compared to many other tools available in the market.

This tool, like any other software application, can be improved in the future to enhance the user experience, security and can also be extended to calculate metrics for applications developed in multiple programming languages. The user interface (UI) of the tool is very simple and can be extended by adding new functionalities like allowing the users to register, login, save files and results in their respective accounts etc., to enhance the user experience. The validation rules for the files uploaded by the users are very basic in the tool. The tool only checks for the extension of the file and accepts the files if they have a C# source file extension (.cs extension). Additional validations could be applied in the future to increase the overall security of the tool. The software metrics tool, at present, supports and calculates metrics for only the applications developed in C# programming language and can be extended in the future to support different programming languages.

BIBLIOGRAPHY

[1] Cem Kaner and Walter P. Bond, Software Engineer Metrics: What do they measure and how do we know? 10th International Software Metrics Symposium, Metrics 2004.

[2] Gordana Rakic and Zoran Budimac, Problems in Systematic Application of Software Metrics and Possible Solution. The 5th International Conference on Information Technology, ICIT 2011.

[3] Tom DeMarco (1984). Controlling Software Projects: Management, Measurement and Estimation.

[4] Software Metric (n.d.). Retrieved from https://en.wikipedia.org/wiki/Software_metric

[5] Margaret Rouse (April, 2005). Metric. Retrieved from http://whatis.techtarget.com/definition/metric

[6] Shane Hastie and Stephane Wojewoda (Oct 04, 2015). Standish Group 2015 Chaos Report. Retrieved from http://www.infoq.com/articles/standish-chaos-2015

[7] Mohammed Javeed Ali (June 15, 2006). Metrics for Requirements Engineering. Retrieved from http://www8.cs.umu.se/education/examina/Rapporter/JaveedAli.pdf

[8] David A. Wheeler (August 1, 2004, Version 2.26). SLOCCount User's Guide. Retrieved from http://www.dwheeler.com/sloccount/sloccount.html

[9] SourceMonitor, Campwood Software (June 20, 2014, Version 3.5). Retrieved from http://www.campwoodsw.com/sourcemonitor.html

[10] NDepend (n.d.), (Version 6). Retrieved from http://www.ndepend.com/docs/getting-started-with-ndepend and https://en.wikipedia.org/wiki/NDepend

[11] Code Counter Pro (n.d.). Retrieved from http://www.geronesoft.com/

[12] SLOC Metrics (n.d.). Retrieved from http://microguru.com/products/sloc/

[13] Robert E. Park (September 1992). Technical Report on Software Size Measurement: A Framework for Counting Source Statements. Retrieved from http://www.sei.cmu.edu/reports/92tr020.pdf

[14] Resource Standard Metrics (n.d.), (Version 7.75). Retrieved from http://msquaredtechnologies.com/m2rsm/index.htm and http://msquaredtechnologies.com/m2rsm/rsm_demo.php

[15] C# Source Code Metrics (n.d.). Retrieved from http://www.semdesigns.com/Products/Metrics/CSharpMetrics.html and http://www.semdesigns.com/Products/Metrics/index.html

[16] James T. Heires (November 15, 2014, Version V4.1.0.3). EZ-Metrix V4.1.0.3 User guide. Retrieved from http://www.jamesheiresconsulting.com/scc/downloads/EZ-Metrix%20User%20Guide.pdf

[17] McCabe IQ (n.d.). Retrieved from http://www.mccabe.com/iq.htm

[18] CodeMetrics (May 21, 2007, Version 10). Retrieved from http://reflectoraddins.codeplex.com/wikipage?title=CodeMetrics&referringTitle=Home

[19] Code Metrics (Visual Studio 2013). Retrieved from https://msdn.microsoft.com/en-us/library/bb385914(v=vs.120).aspx

[20] Norman Fenton, James Bieman (July 28, 2014). Software Metrics: A Rigorous and Practical Approach (Third Edition).

[21] Cyclomatic Complexity (n.d.). Retrieved from http://www.ndepend.com/docs/code-metrics#CC

[22] Karen Ng, Matt Warren, Peter Golde and Anders Hejlsberg (Microsoft Corporation. September, 2012). The Roslyn Project: Exposing the C# and VB compiler's code analysis.

[23] Layered Application Guidelines (n.d.). Retrieved from https://msdn.microsoft.com/en-us/library/ee658109.aspx

[24] UML 2.5 Diagrams Overview (2016). Retrieved from http://www.uml-diagrams.org/uml-25-diagrams.html

[25] E. Allen (2001). Designing extensible applications. IBM developer Works.

[26] Extensibility (n.d.). Retrieved from https://en.wikipedia.org/wiki/Extensibility

[27] C# Source Code (n.d.). Retrieved from http://www.sourcecodester.com/c-sharp

[28] C# Projects and Source Code (n.d.). Retrieved from http://1000projects.org/c-projects-and-source-code-free-download.html