

PROGRAM ANALYSIS TOOL

Project Id: 18-055

System Requirements Specification

Component – Analyze code by measuring composite complexity including inheritance using complexity metrics and sentiment analysis of user feedback to determine performance of software developers

P.K.H Palihakkara – IT15113900

Bachelor of Science (Honors) Degree in Information Technology

Department of Software Engineering
Sri Lanka Institute of Information Technology

May 2018

DECLARATION

I hereby declare that the submitted project Software Requirements Specification document for ProAnalýza is an original work done by P.K.H Palihakkara. This document is proprietary and an exclusive property of the SLIIT project group 18-055. List of references I referred for the preparation of this document are given as references at the end of the document.

Member: IT15113900 – P.K.H Palihakkara

Signature:

ABSTRACT

The main objective of this research is to develop a program analyzing tool in a manner that it allows its users to easily understand a given program. The tool is initially built to analyze Object Oriented programs. In addition, it provides ratings regarding the performance of each programmer/developer and suggestions to reduce the complexity of programs which are identified as complex.

With the existing tools, we can measure the quality of the source code and warnings/errors produced. Even, most of the warnings are spurious and the developers are not paying attention to the output. There are many teams involving in software code quality maintenance. Through proposed application, an efficient and convenient manner which measure the complexity of both small and large Java programs will be provided, allowing them to quickly and easily remove code bottlenecks thus reducing maintenance and testing phase costs and efforts significantly.

It's a major requirement to determine the capacity and performance of Software developers assigning to a project in terms of gaining the maximum productivity of projects in commercial scale. We are focusing to develop a more interactive Program analysis tool to analyze the source code quality and to determine the performance of Software developers based on their code quality.

LIST OF TABLES

	Page
Table 1.1 Definitions, Acronyms and Abbreviations	2
Table 2.1 Use Case Scenario 1	19
Table 2.2 Use Case Scenario 2	19
Table 2.3 Use Case Scenario 3	20
Table 2.4 Use Case Scenario 4	20
Table 4.1 Tabular representation of Gantt Chart	39

LIST OF FIGURES

	Page
Figure 2.1 High Level System Architecture	4
Figure 2.2 User Interface 1	9
Figure 2.3 User Interface 2	10
Figure 2.4 User Interface 3	11
Figure 2.5 User Interface 4	11
Figure 2.6 User Interface 5	12
Figure 2.7 User Interface 6	12
Figure 2.8 User Interface 7	13
Figure 2.9 User Interface 8	13
Figure 2.10 User Interface 9	14
Figure 2.11 Software Interface 1	15
Figure 2.12 Software Interface 2	16
Figure 2.13 Use case diagram of ProAnalýza	18
Figure 3.1 User Interface 1	24
Figure 3.2 User Interface 2	25
Figure 3.3 User Interface 3	26
Figure 3.4 User Interface 4	27
Figure 3.5 User Interface 5	28
Figure 3.6 User Interface 6	29
Figure 3.7 User Interface 7	30
Figure 3.8 User Interface 8	31
Figure 3.9 User Interface 9	32
Figure 3.10 User Interface 10	32
Figure 3.11 Class diagram of ProAnalýza	34
Figure 4.1 Comparision of existing static code analyzing tool with reference to the features that included in ProAnalýza	38
Figure 4.2 Gantt Chart	40
Figure 4.3: Work Breakdown Structure	41

Table of Contents

DECLARATION	i
ABSTRACT	ii
LIST OF TABLES	iii
LIST OF FIGURES	iv
1 INTRODUCTION	1
1.1 Purpose.....	1
1.2 Scope.....	1
1.3 Definitions, Acronyms, And Abbreviations	2
1.4 Overview	3
2 OVERALL DESCRIPTIONS.....	4
2.1 Product Perspective.....	5
2.1.1 System interfaces	9
2.1.2 User interfaces	9
2.1.3 Hardware interfaces	14
2.1.4 Software interfaces.....	15
2.1.5 Communication interfaces	16
2.1.6 Memory constraints	16
2.1.7 Operations	17
2.1.8 Site adaptation requirements.....	17
2.2 Product Functions	18
2.3 User characteristics	21
2.4 Constraints	21
2.5 Assumptions and dependencies	21
2.6 Apportioning of requirements.....	22
3 SPECIFIC REQUIREMENTS.....	23
3.1 External Interface Requirements.....	23
3.1.1 User interfaces	23
3.1.2 Hardware interfaces	33
3.1.3 Software interfaces.....	33
3.1.4 Communication interfaces	33
3.2 Classes/Objects	34
3.3 Performance Requirements	35
3.4 Design Constraints	35
3.5 Software System Attributes	36
3.5.1 Reliability.....	36
3.5.2 Availability	36
3.5.3 Security	36
3.5.4 Maintainability	36
3.6 Other Requirements	37
4 SUPPORTING INFORMATION.....	38
4.1 Appendices.....	38
5 REFERENCES	42

1 INTRODUCTION

This section gives a scope description and overview of everything included in this SRS document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided.

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the “ProAnalýza”, program analysis tool which would analyze and measure and display complexity of code effectively using traditional and novel complexity metrics by clearly identifying complex code segments. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications. This document is primarily intended to be proposed to a customer for its approval and a reference for developing the first version of the system for the development team.

1.2 Scope

The “ProAnalýza” is a web application which helps specially software developers to analyze, measure and effectively display complexity of code written in Java language using complexity metrics by clearly identifying complex code segments which would increase the maintainability of the software product. In addition, to check the quality of a program based on the industry or company specific quality standards and rank the software developers according their code quality. The application should be free to download from either a website or similar services.

Users can provide their expected quality standards and their source code to the system. This information will act as the bases for the analysis of a program and its results displayed to the user.

Furthermore, the software needs an Internet connection to fetch and display results. All system information is maintained in a database, which is located on a web-server. This application also has the capability of representing both summary and detailed information after analyzing the source code.

1.3 Definitions, Acronyms, And Abbreviations

Table 1.1: Definitions, Acronyms and Abbreviations

Term	Definition
SRS	Software Requirements Specification
Source code	A series of programming statements written in Java language
User	Someone who interacts with the proposed tool
Stakeholder	Any person who is interact with the system
ANLTR	Another Tool for Language Recognition
OOP	Object-Oriented Programming
GUI	Graphical User Interface
JVM	Java Virtual Machine
MySQL	My Structured Query Language.
HTML5	The fifth and current major version of Hyper Text Markup Language
PHP5	The fifth and current major version of Hypertext Preprocessor which is a server-side scripting language designed for web development
CSS3	The latest evolution of the Cascading Style Sheets language
AJAX	Asynchronous JavaScript and XML
Bootstrap	A free and open-source front-end library for designing websites and web applications
jQuery	A cross-platform JavaScript library designed to simplify the client-side scripting of HTML
TCP/IP	Transmission Control Protocol/Internet Protocol
RAM	Random Access Memory
developer	Software developer/Software engineer who develops a software
NLP	Natural Language Processing
MTBF	Mean Time between Failures
MTTF	Mean Time To Failures

1.4 Overview

The remainder of this document includes two chapters, appendixes and references. The second one provides an overview of the system functionality and system interaction with other systems. This chapter also introduces different types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.

The third chapter provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are to specify the requirements more precisely for different audiences.

The appendixes in the end of the document include comparison of existing static source code analyzing tools with reference to the features which will be consisted in ProAnalýza and a work breakdown structure of the project.

2 OVERALL DESCRIPTIONS

This section will give an overview of the whole system. The system will be explained in its context to show how the system interacts with other systems and introduce the basic functionality of it. It will also describe what type of stakeholders that will use the system and what functionality is available for each type. At last, the constraints and assumptions for the system will be presented.

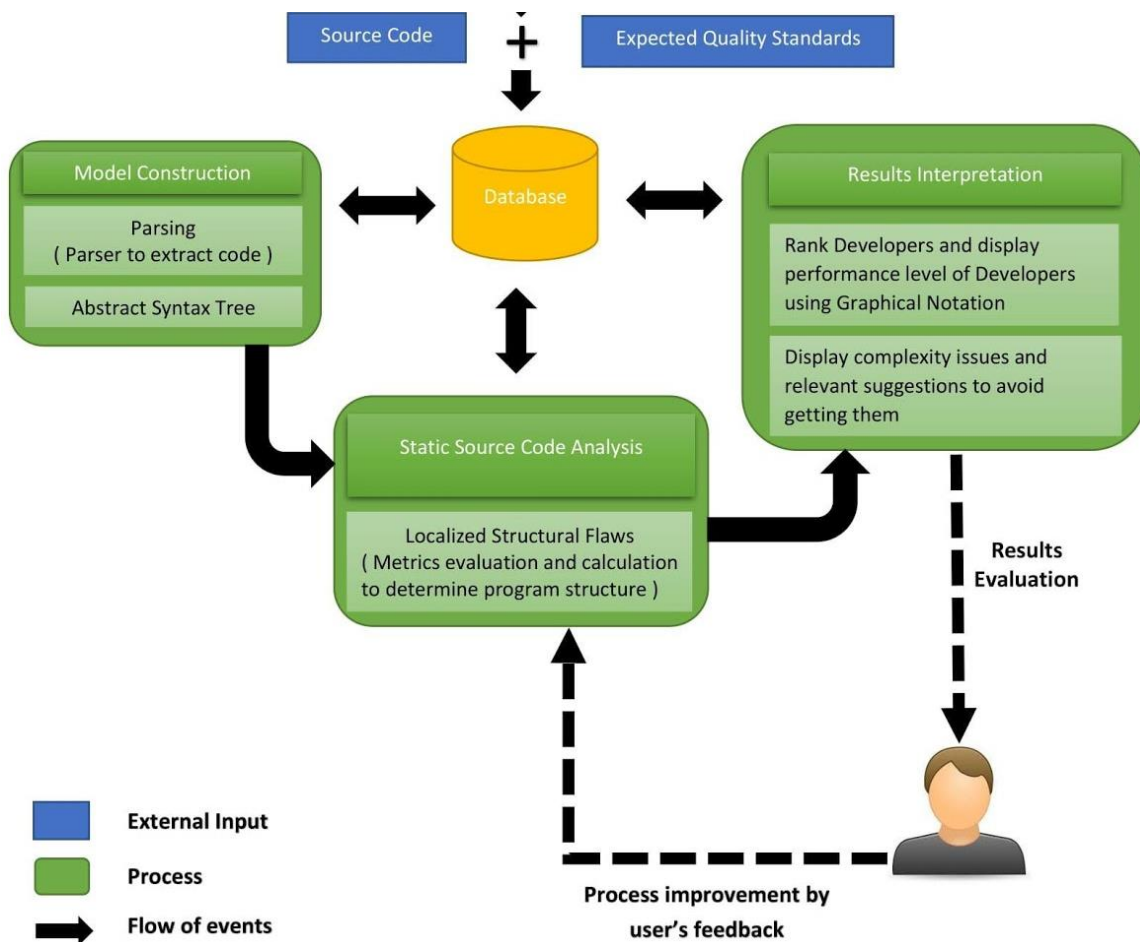


Figure 2.1: High Level System Architecture

2.1 Product Perspective

This system will consist of three main components: model construction, static source code analysis and results interpretation.

When user input the source code and expected quality standards, model construction will be initiated. There parsing the source code in which character stream of the source code breaks up into constituent pieces and imposes a grammatical structure on them will be take place. In this approach, NLP is use for code parsing. Rather than developing a parser from scratch with the use of NLP, one of the existing parsing tool, ANLTR would be used in integration.

After parsing the code and generating an abstract syntax tree, static source code analysis is carry out by evaluation of complexity metrics based on the program structure. We are focusing on following major factors of an OOP concepts in complexity metrics evaluation.

1. Coupling
2. Exception handling
3. Inheritance
4. Control Structures

In this specification, more explanation is given about analyzing source code based on inheritance which is a component of complexity measuring module and development of an algorithm based on complexity metrics to accurately measure coupling of a program.

In object-oriented programming, objects will be characterized by classes. It is possible to learn a lot about an object based on the class it belongs to. Objected oriented programming takes this concept to a Whole new level. It permits classes to be defined in relation to other classes. Every subclass will inherit a state from the super class. Despite this, subclasses are not restricted to the behaviors and states that they have taken from their super class. A subclass can combine methods and variables with the traits they have inherited from their super class. It is also possible for subclasses to override any methods that they have inherited, and they can create unique implementations for these methods. It is also possible to use more than just one level of inheritance. An inheritance structure can be generated which can be as deep as you want it to be. This inheritance structure is called a class hierarchy. The variables and methods can extend through the levels of the class hierarchy.

In most cases, a hierarchy that is deep tends to have behaviors which are distinct. It should always define what the classes are instead of how they are used. The object class should be at the zenith of the class hierarchy. Every class should descend from it in a direct or indirect manner. The variable of an object type can retain a reference for any object, and an example of this would be a class. For example, the object could define behaviors that may be attributed to the objects that are processed by the Java Virtual Machine. There are a number of power advantages to the concept of inheritance. Subclasses can generate distinct behaviors which are based on the common attributes that are present in their super class. Because of inheritance, it is possible for programmers to use the same code many times over. Programmers can generate super classes which are named abstract classes. Abstract classes will characterize behaviors which are common. While some aspects of this behavior may be defined, a large portion of it will not be defined at all. It shows how subclasses are connecting to their super classes, and it can also allow you to understand which traits have been passed from the super class to its subclasses. It is one of the most powerful features of object-oriented programming.

ProAnálýza would implement following criteria which establish relevant functionalities in static source code analysis based on inheritance.

- Single inheritance level of each class
- A count and names of immediate parent class(es)
- A count and names of immediate child class(es)
- A count and names of ancestor class(es)
- A count and names of descendant class(es)
- Multilevel inheritance
- Hierarchical inheritance

In addition, the conflicts which can be there in source code with respect to inheritance is as follows and those conflicts will be indicated as warnings/errors when analyzing source code.

- Member Conflicts
 - Name conflicts can occur - same member name from more than one base class
 - Derived class can overshadow base class members name

- Use scope resolution operator to resolve conflicts
- Multiple Inheritance Conflicts
 - Derived class may combine more than one copy of a member
 - Base class may combine more than one copy of a member

Other than inheritance, following OOP concepts are being analyzed.

- Coupling
 - Call to a regular method in the same class
 - Call to a regular method in a different class
 - Call to a recursive method in a same class
 - Call to a recursive method in a different class
 - Referencing a data member of the same class
 - Referencing a data member in a different class
 - Calling from another method in the same class
 - Calling from another method in a different class
 - Call to a recursive method in a same class
 - Referencing of a data member in the current class by a method of a different class
 - Coupling types – Data, stamp, control, global and context
- LOC
 - LOC: count of all the lines in a program except for blank and commented lines count lines of code
 - ILOC: count of all the lines in a program that are ending with semicolons
 - eLOC: Count of all the lines in a program except for blank lines, commented lines and lines with only brackets
 - Commented lines
 - Number of executable and non-executables statements
 - CLOC: count comment lines of code
- Polymorphism
 - Method overriding
 - Method overloading

- Encapsulation
- Tokens
 - Number and type of the operators used – Arithmetic, relational, bitwise, logical, assignment, decisional, and dot operators
 - Number and type of the operands used – Identifier, return type of a method, string, numerical values
- Control structures
 - Number of control structures
 - Number of conditions checked by a single control structure
 - iterations in iterative control structure (Nesting level)
 - The types of basic control structures
 - The nesting levels of control structures
- Abstraction
- Variable types
- Exception handling

2.1.1 System interfaces

Windows 10 operating system will be used as the development platform. One of the major technology that we use in software invocation is based on Java language. The linking between the system and the Operating system is handled by the Microsoft Windows environment and JVM.

2.1.2 User interfaces

The following is the main GUI of ProAnalýza where user can gain access to all functionalities available in it.

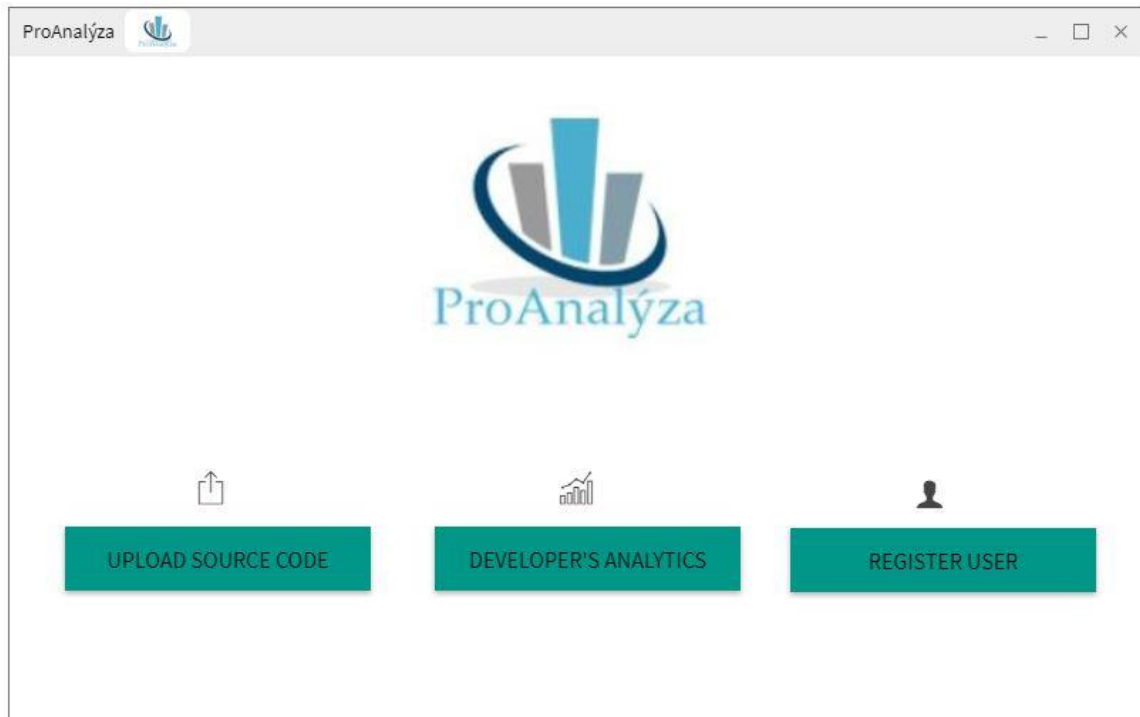
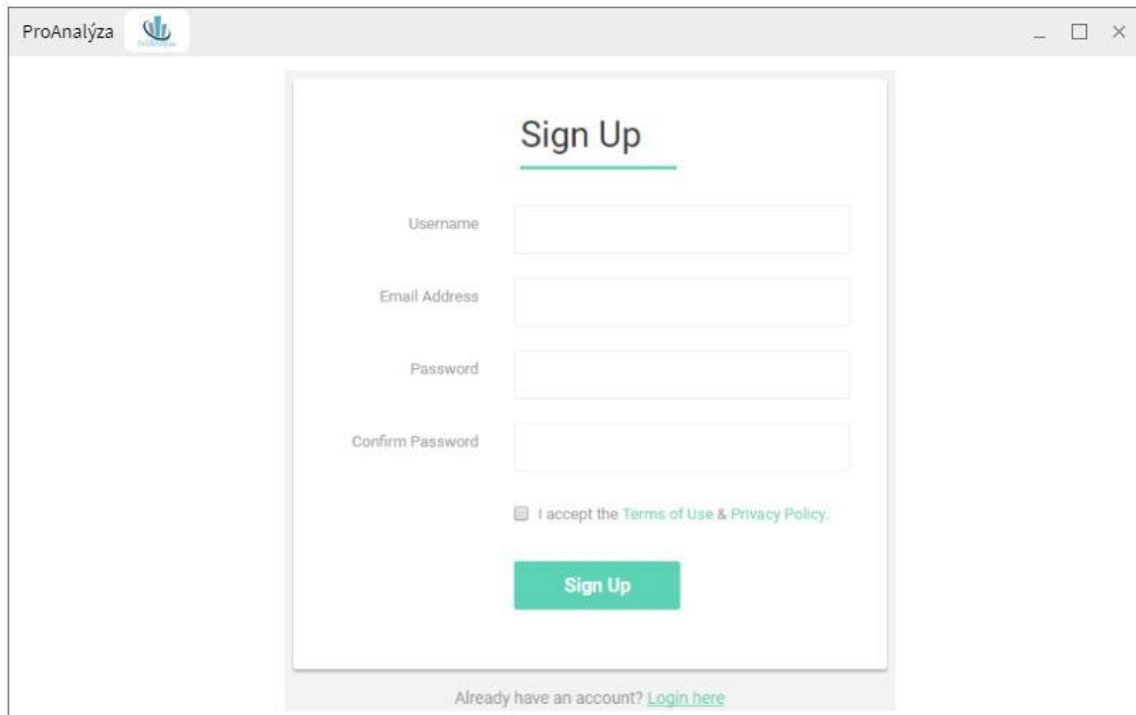


Figure 2.2: User Interface 1

The screenshots of the GUI for the core functionalities of ProAnalýza are as follows.

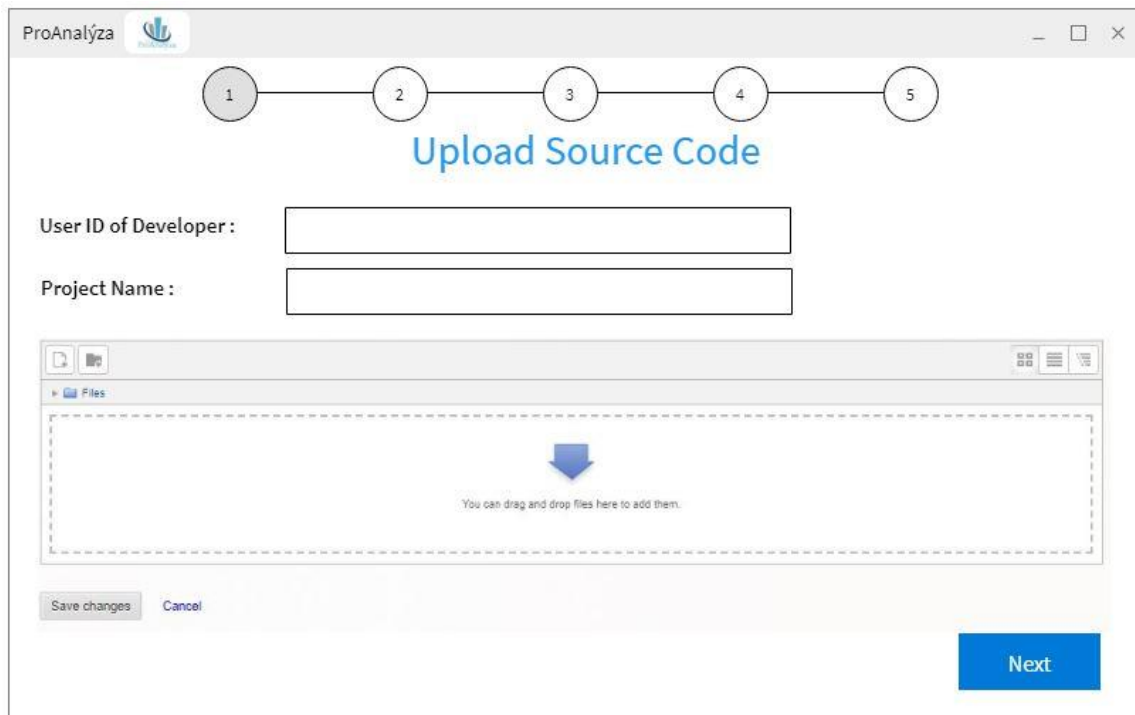
1. User Registration



The screenshot shows a web browser window titled "ProAnalýza" with a logo. The main content is a "Sign Up" form. The form has a title "Sign Up" with a green underline. Below the title are four input fields: "Username", "Email Address", "Password", and "Confirm Password". Below these fields is a checkbox labeled "I accept the Terms of Use & Privacy Policy." with a green link "Terms of Use & Privacy Policy." to its right. Below the checkbox is a green "Sign Up" button. At the bottom of the form, there is a link "Already have an account? Login here" in green text.

Figure 2.3: User Interface 2

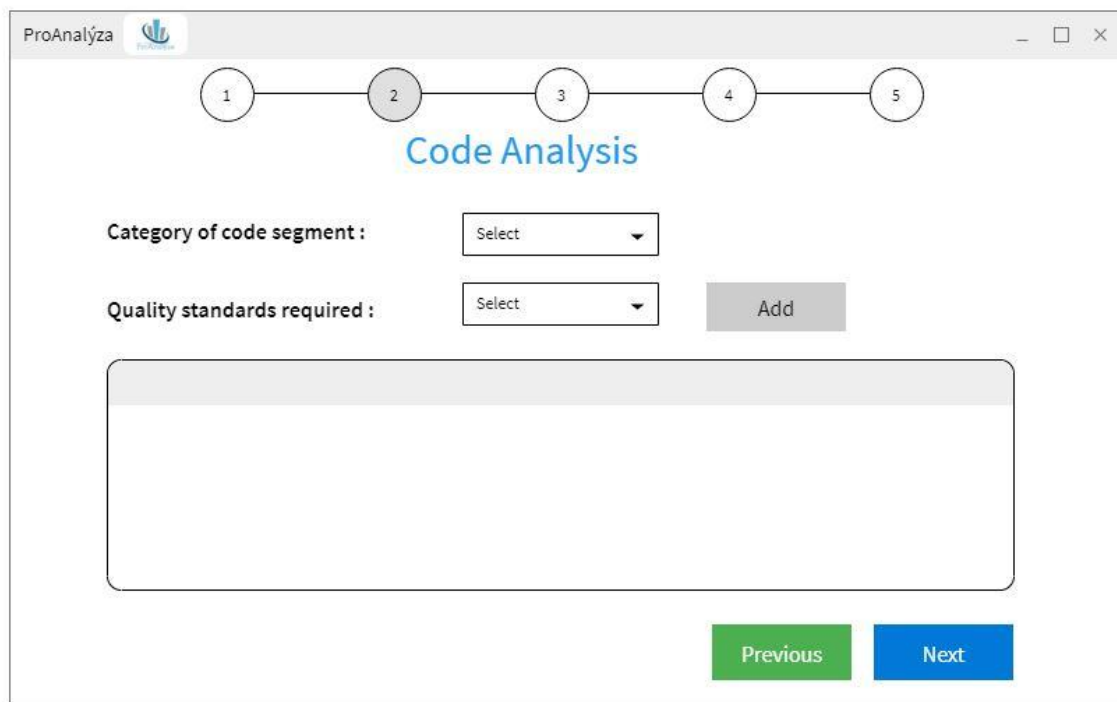
2. Upload source code



The screenshot shows a web application window titled "ProAnalýza". At the top, there is a progress bar with five numbered circles (1 to 5). Circle 2 is highlighted, indicating the current step. Below the progress bar, the title "Upload Source Code" is displayed in blue. The form contains two input fields: "User ID of Developer:" and "Project Name:". Below these fields is a file upload area with a dashed border and a blue arrow pointing down, with the text "You can drag and drop files here to add them." Below the file upload area, there are two buttons: "Save changes" and "Cancel". At the bottom right, there is a blue "Next" button.

Figure 2.4: User Interface 3

3. Input expected quality standards



The screenshot shows a web application window titled "ProAnalýza". At the top, there is a progress bar with five numbered circles (1 to 5). Circle 2 is highlighted, indicating the current step. Below the progress bar, the title "Code Analysis" is displayed in blue. The form contains two dropdown menus: "Category of code segment:" and "Quality standards required:". Below the "Quality standards required:" dropdown, there is an "Add" button. Below the "Add" button, there is a large empty rectangular box. At the bottom right, there are two buttons: "Previous" (green) and "Next" (blue).

Figure 2.5: User Interface 4

ProAnalýza

1 — 2 — 3 — 4 — 5

Code Analysis

Category of code segment : Class

Quality standards required : Select Add

Composite Complexity ×
Inheritance ×
List of Operators/Operands ×

Previous
Next

Figure 2.6: User Interface 5

4. Tabular representation of complexity metrics evaluation of source code

ProAnalýza

1 — 2 — 3 — 4 — 5

Composite Complexity

Package Name													
Class Name													
Method Name	Statement Number	Executable statement	List of operators	List of operands	Size (S)	Weight due to nesting level of control structures (Wn)	Weight due to inheritance level of statements (Wi)	Weight due type of control structures (Wc)	Total weight (Wt)	S x Wt	Complexity of a method		
										5			
										6			
										7			
Complexity of Method A											18		
										4			
										5			
										9			
										2			
Complexity of Method B											20		
Complexity of class											38		

Previous
Next

Figure 2.7: User Interface 6

5. Provide user feedback to determine developer's performance rate

ProAnalýza

1 2 3 4 5

User Feedback

User ID of Software Developer : SE129

1. Issue/requirement/customer understanding :

2. Transparency, updating, updating status reporting :

3. Project performance improvements & upgrades :

Previous Next

Figure 2.8: User Interface 7

6. Performance evaluation of the developer

ProAnalýza

1 2 3 4 5

Performance Evaluation

User ID of Software Developer : SE129

Questions	Weight	Score (out of 5)	Weighted Average
RESULTS	60		
Issue/requirement/customer understanding	20	4	9.6
Transparency, updating, status reporting	5	3	1.8
Product performance improvements & upgrades	15	5	9
Documentation & quality control	15	5	9
Process adherence	10	5	6
New projects/initiatives	15	2	3.6
New technologies	10	3	3.6
Code contribution	10	4	4.8
Sectional Score >>			3.95
BEHAVIOR	40		
Culture fitment	15	2	2.4
Maturity	25	3	6
Team spirit, Group work	25	3	6
Leading from front, in words and in action	15	2	2.4
Personal and Work attitude	20	3	4.8
Sectional Score >>			2.7

Save

Figure 2.9: User Interface 8

7. View developer's performance analytics

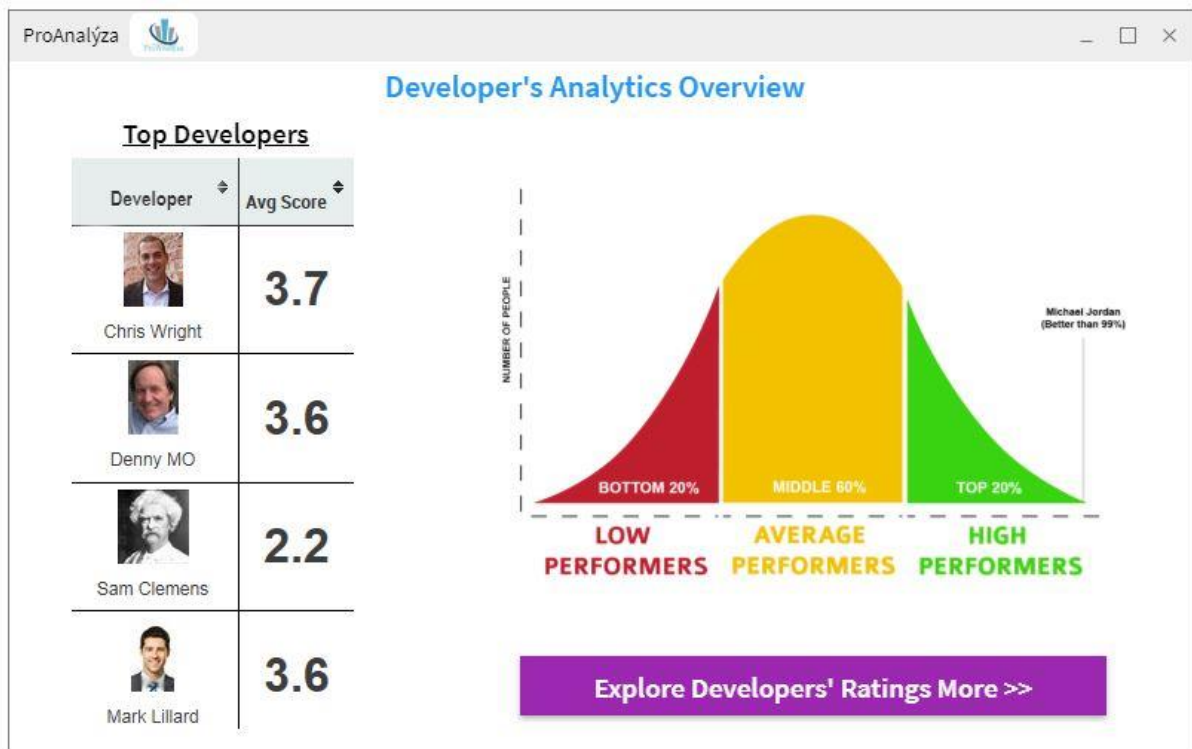


Figure 2.10: User Interface 9

2.1.3 Hardware interfaces

No specific hardware interface is required apart from a standard PC/Laptop with minimum requirement of Intel Pentium IV with at least 1.6GHz processor or above and hardware devices such as USB dongle/ADSL router to connect to the internet.

2.1.4 Software interfaces

Following software interfaces are required to use ProAnalýza.

- In our approach, XAMPP is used to connect to MySQL database and to interpret scripts written in PHP5 programming.

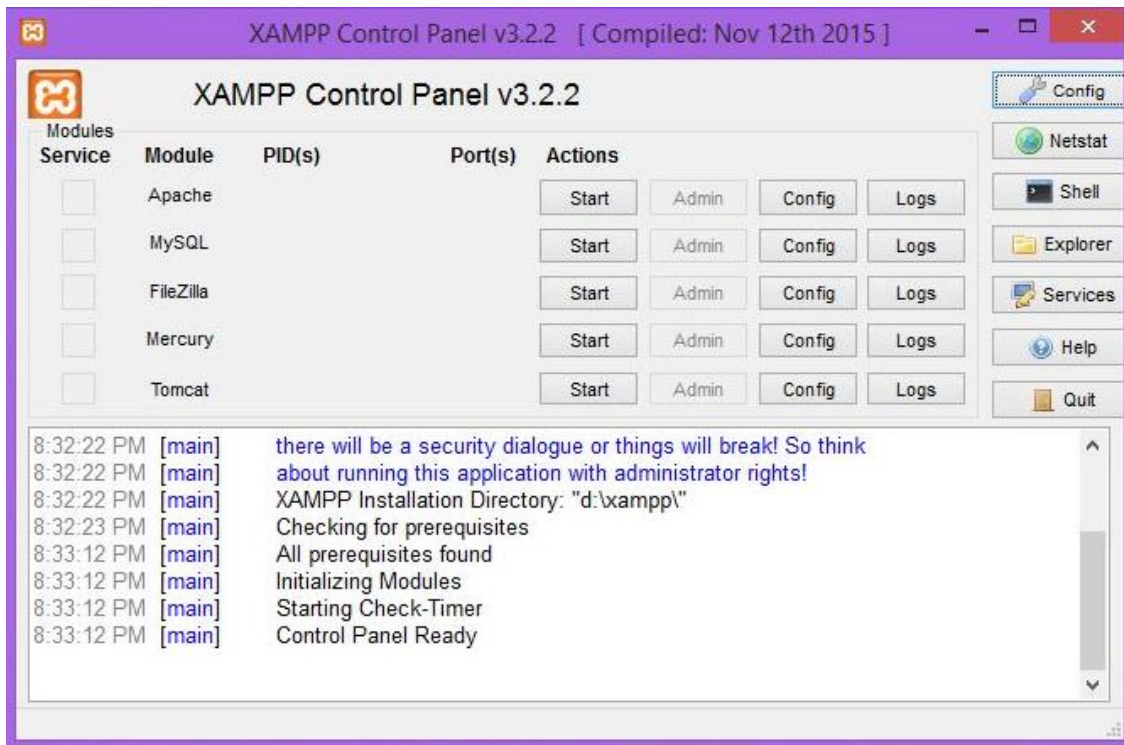


Figure 2.11: Software Interface 1

- Latest updated web browser that supports JQuery, AJAX, HTML5, PHP5, CSS3 and Bootstrap technologies.

Eg: Internet Explorer version 5.0 or above, Mozilla Firefox version 3.5 or above, Google Chrome etc.

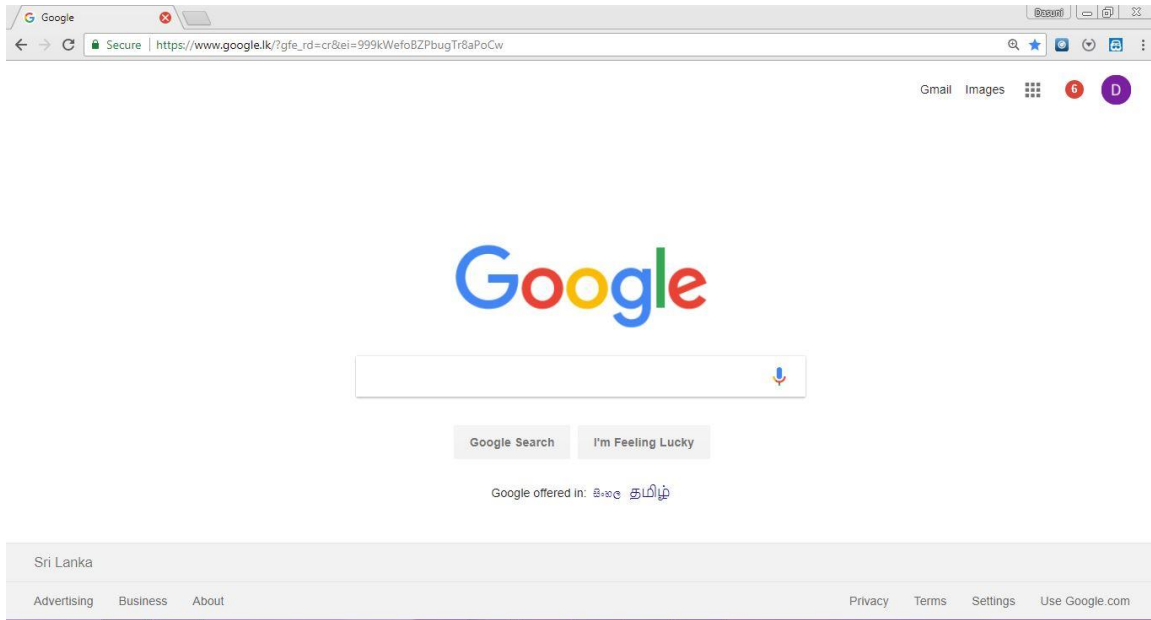


Figure 2.12: Software Interface 2

2.1.5 Communication interfaces

Following communication interfaces are required to use ProAnalyza.

- Internet connection with fairly high bandwidth will be required to run the system efficient since it deals with large about of data. It will use the HTTP protocol for communication over the internet and for the intranet communication will be through TCP/IP protocol suite.
- Database connection interface will be required to connect with the database.

2.1.6 Memory constraints

The device with ProAnalyza installed should have at least RAM of 2GB or higher to run the application.

2.1.7 Operations

The following are the main operations that will be available to a user of the ProAnalýza.

1. User registration

Prior to user login, the user must provide relevant information and get register to the system.

2. Upload source code

When the user successfully login to the system, the user will be able to upload source code through an interface that will allow him/her to ingest their code to the tool.

3. Input expected quality standards

After uploading the source code, the user will first be prompted with an interface where he/she can input the quality standard expected to measure and evaluate source code with respect to complexity metrics.

4. Tabular representation of complexity metrics evaluation of source code

Results after analyzing the source code complexity with respect to the user expected quality standards will be represented in the form of tables.

5. Provide user feedback to determine developer's performance rate

To determine performance of a developer, user feedback must be provided in addition to the complexity metrics evaluation.

6. Performance evaluation of the developer

With user feedback and complexity metrics evaluation of source code, each developer will be rank and through an interface developer's performance analysis will be displayed.

2.1.8 Site adaptation requirements

Following site adaptation requirements are identified regarding the implementation of ProAnalýza.

- The server must have MySQL installed on it.
- Server machine must be running Apache server to deploy the web application.
- The user machine should have Java Virtual Machine installed.

2.2 Product Functions

This section includes the requirements that specify all the fundamental functions of the ProAnalýza.

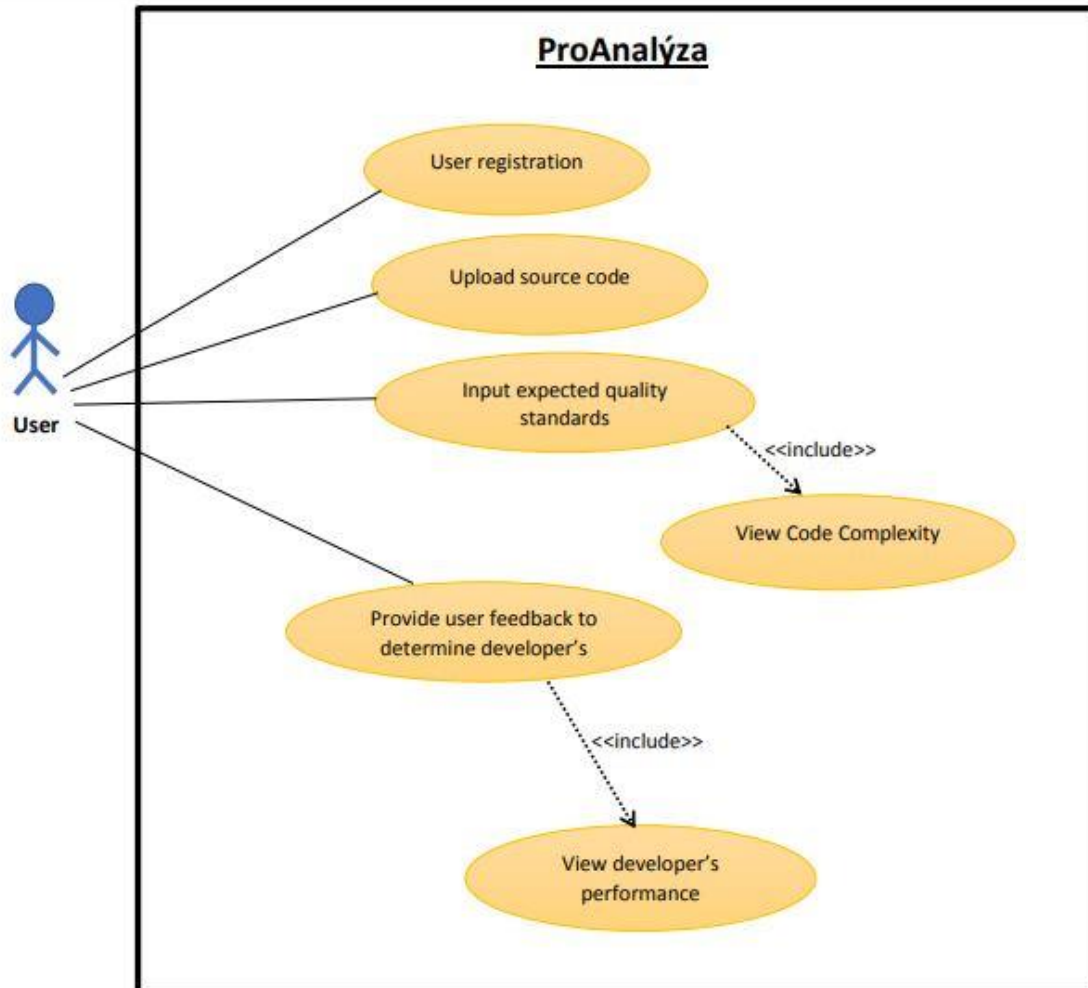


Figure 2.13: Use case diagram of ProAnalýza

Table 2.1: Use Case Scenario 1

Use Case ID:	1
Use Case Name:	User registration
Actors:	User
Pre-conditions:	Program successfully launched Database connection is live User can view the main interface of ProAnalýza
Post-conditions:	User registration successful
Flow of events:	1. Navigate to the user registration menu 2. Enter relevant information in the registration form 3. Then press “Sign Up” button
Alternative flows:	2a. Upon validation of user information, if there’s an error or wrong format of details being entered by the user, he/she will be prompt a notification by the system.
Exceptions:	None

Table 2.2: Use Case Scenario 2

Use Case ID:	2
Use Case Name:	Upload source code
Actors:	User
Preconditions:	Program successfully launched Database connection is live User can see the upload interface
Post-conditions:	Program code successfully uploaded
Normal Flow:	1. Click Browse button 2. Select the source code folder 3. Click upload button
Alternative Flows:	1a. Drag and drop program code folder into upload area
Exceptions:	None

Table 2.3: Use Case Scenario 3

Use Case ID:	3
Use Case Name:	Input expected quality standards
Actors:	User
Preconditions:	Program code successfully uploaded Database connection is live User can see the quality standards interface
Post-conditions:	User successfully submitted quality standards preference and user can see the tabular representation of complexity metrics evaluation of source code in an interface
Normal Flow:	1. Select desired metric from the combo box 2. Click "Add button" 3. Click "Next" button
Alternative Flows:	2a. Click "Previous" button to upload the source code
Exceptions:	None

Table 2.4: Use Case Scenario 4

Use Case ID:	4
Use Case Name:	Provide user feedback to determine developer's performance rate
Actors:	User
Preconditions:	Database connection is live User can see the user feedback interface
Post-conditions:	User successfully submitted user feedback about the developer's performance and user able to see the ratings of developer's performance in an interface
Normal Flow:	1. Fill the relevant information in the form 2. Click "Next" button
Alternative Flows:	2a. Click "Previous" button to see complexity evaluation again to provide user feedback
Exceptions:	None

2.3 User characteristics

Basic knowledge of using computers and Java programming is adequate to use this application. Knowledge of how to use a mouse or keyboard and internet browser is necessary. The user interface will be friendly enough to guide the user.

There will only be one user (this can range from an industry expert, to a student, to a lecturer/teacher or industry employee) that will have the same level of privileges and will have access to the complete functionality of the tool.

2.4 Constraints

- Access to the web is required (Hence an internet connection with fairly high bandwidth is necessary).
- All the tools and technologies used for the development should be open source. Since we could accommodate a low budget for the implementation.
- ProAnalýza shall operate on PCs running Windows 95 or later at a minimum speed of 100 MHZ.
- Java, HTML5, PHP5, CSS3, Bootstrap, NLP shall be the implementation technologies.

2.5 Assumptions and dependencies

It is assumed that the user is familiar with an internet browser and also familiar with handling the keyboard and mouse (Users should have basic knowledge using a computer).

Since the application is a web application there is a need for the internet browser. It will be assumed that the users will possess decent internet connectivity.

Server should be up and running 24x7 hours.

There should be sufficient memory and processing power to run the system (as specified in 2.1.6)

2.6 Apportioning of requirements

The requirements described in sections 1 and 2 of this document are referred to as primary specifications; those in section 3 are referred to as requirements (or functional) specifications. The two levels of requirements are intended to be consistent. Inconsistencies are to be logged as defects. In the event that a requirement is stated within both primary and functional specifications, the component will be built from functional specification since it is more detailed.

3 SPECIFIC REQUIREMENTS

This section includes specific requirements related to analyzing source code based on coupling which is a component of complexity measuring module.

3.1 External Interface Requirements

3.1.1 User interfaces

The basic GUIs of ProAnalýza are shown in the section 2.1.2.

All the GUIs that will required to analyze a Java program in terms of inheritance are as follows.

To determine the complexity of a program with respect to inheritance as mentioned in the section 2.1.2 user must create an account to login to system and then must upload the source code. After that user will be prompt with an interface where he/she has to input the expected quality standards. Therefore, to evaluate complexity of a program with respect to inheritance user must select the category of code segment as package, class or method and quality standards required as inheritance. The GUI indicated below provides a clear picture of this scenario. Each of these GUI represent the steps that needs to be follow.

- In the Main window of ProAnalýza, the user must click on “REGISTER USER” to obtain access to the tool.

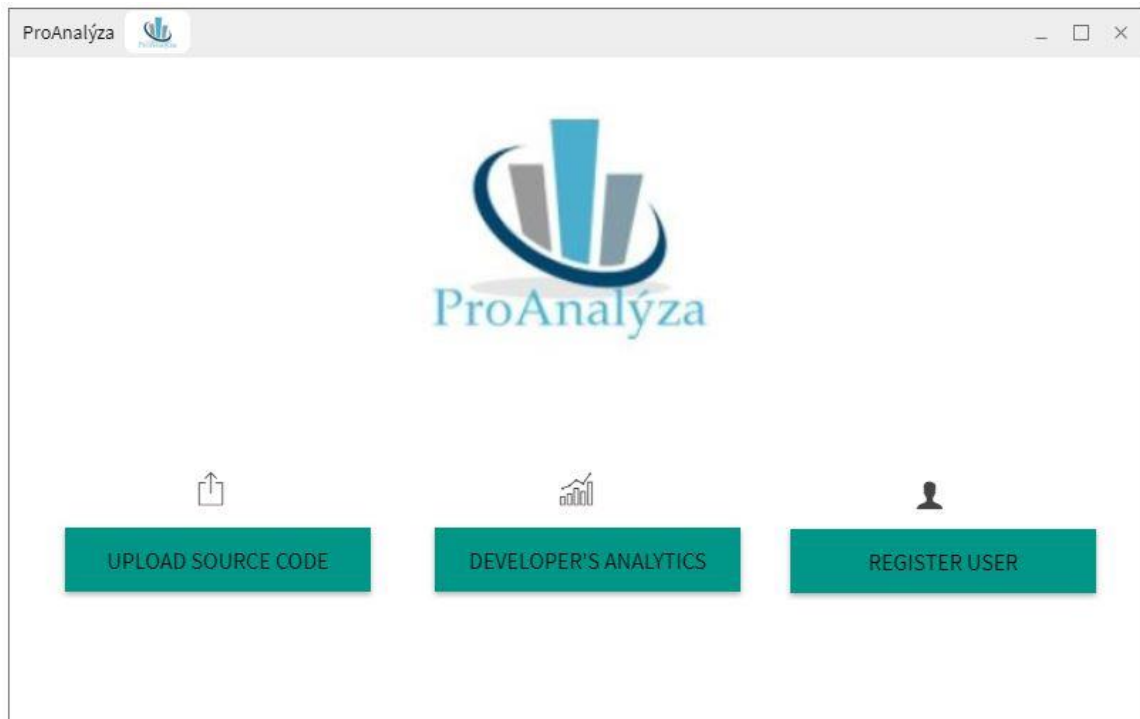
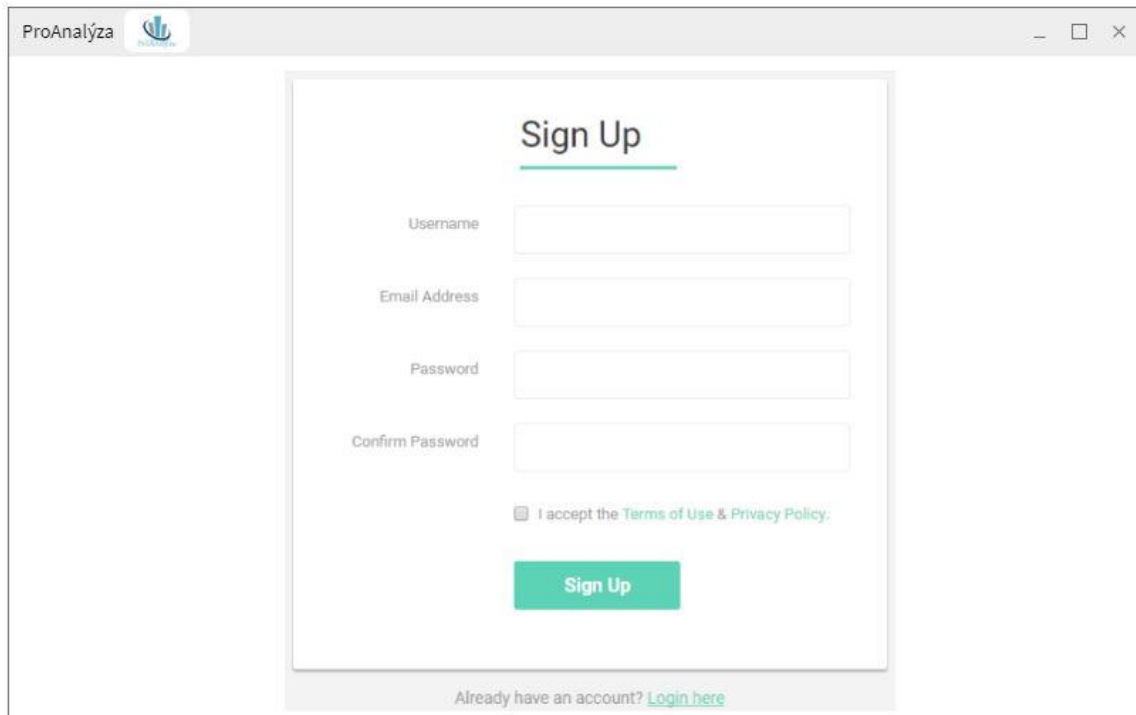


Figure 3.1: User Interface 1

- Once the user provided successfully registered to the system by providing relevant information the user can get registered the system. The following interface provides user registration GUI where user needs to fill his/her details.



The screenshot displays a web browser window titled "ProAnalýza" with a logo. The main content is a "Sign Up" form. The form includes four input fields: "Username", "Email Address", "Password", and "Confirm Password". Below these fields is a checkbox labeled "I accept the Terms of Use & Privacy Policy." with a link to the policy. A green "Sign Up" button is positioned below the checkbox. At the bottom of the form, there is a link: "Already have an account? [Login here](#)".

Figure 3.2: User Interface 2

- Then the user can provide user credentials to the system by the GUI as follows and upon authentication, if those credentials are accurate the user will be able to gain access to the system.

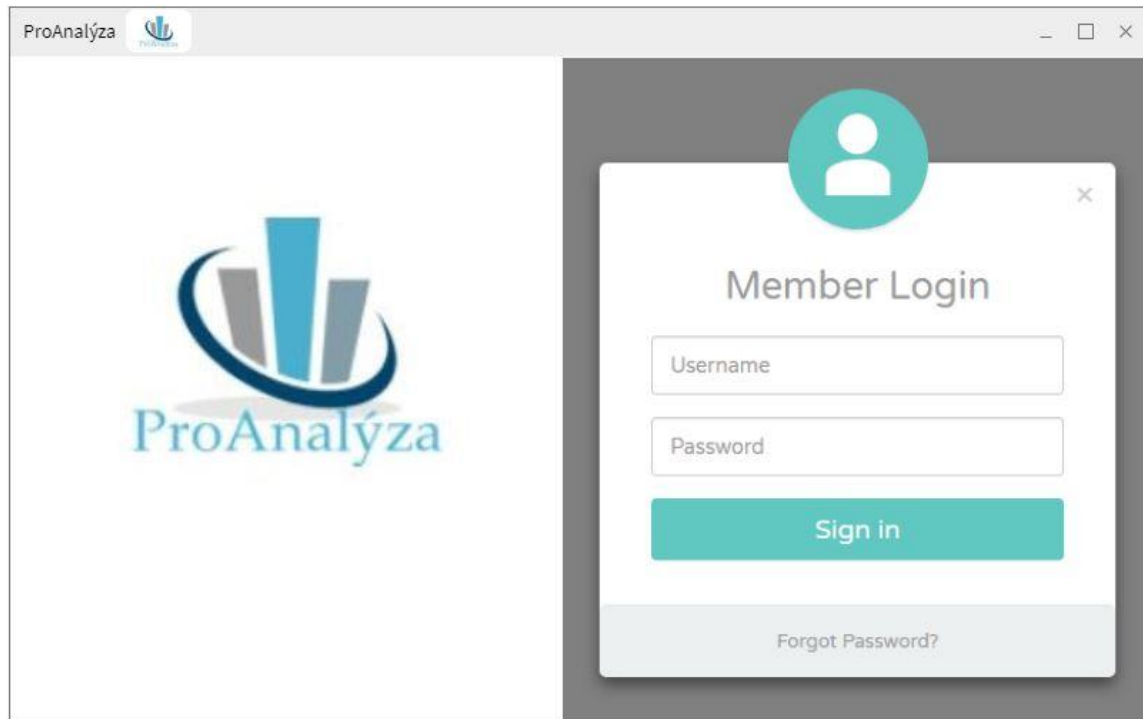


Figure 3.3: User Interface 3

- When the user successfully login to the system he/she will be able to upload source code from the GUI as follows.

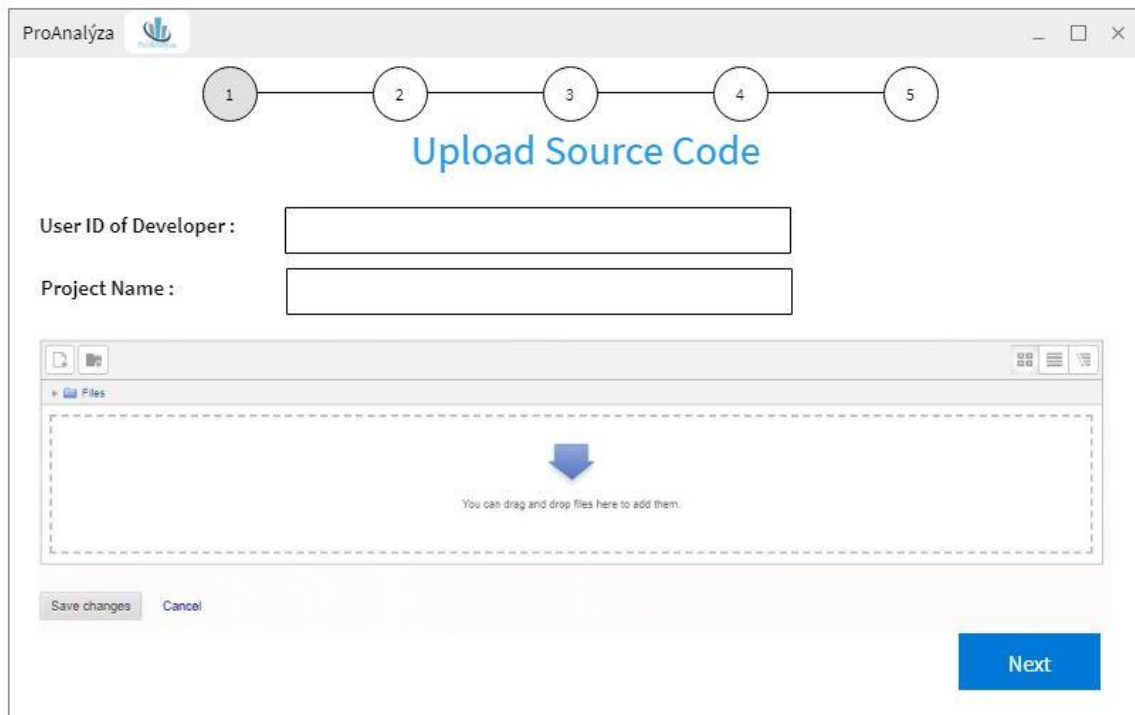


Figure 3.4: User Interface 4

- Then the user can click “Next” button to input the expected quality standards from the GUI given below.

The screenshot shows a web application window titled "ProAnalyza" with a logo. At the top, a progress bar consists of five numbered circles (1-5), with circle 2 highlighted in grey. Below the progress bar, the text "Code Analysis" is displayed in blue. The main form area contains two labels: "Category of code segment :" and "Quality standards required :". Each label is followed by a dropdown menu with "Select" as the placeholder text. To the right of the "Quality standards required :" dropdown is a grey "Add" button. Below these elements is a large, empty rectangular box with a light grey header. At the bottom right of the form are two buttons: a green "Previous" button and a blue "Next" button.

Figure 3.5: User Interface 5

- If the user required to analyze the program in terms of inheritance, the user must select criteria as aforementioned and which is given in the GUI as follows.

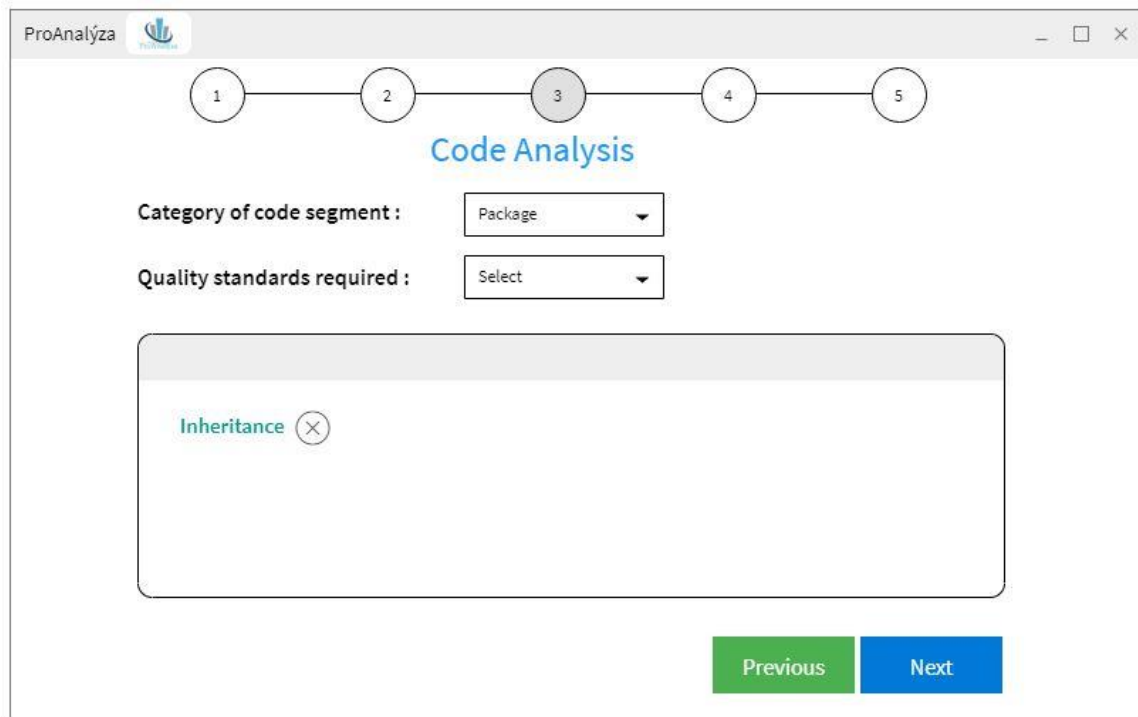


Figure 3.6: User Interface 6

- By clicking “Next”, the user will be able to observe the results of complexity evaluation of source code in terms of inheritance as follows.

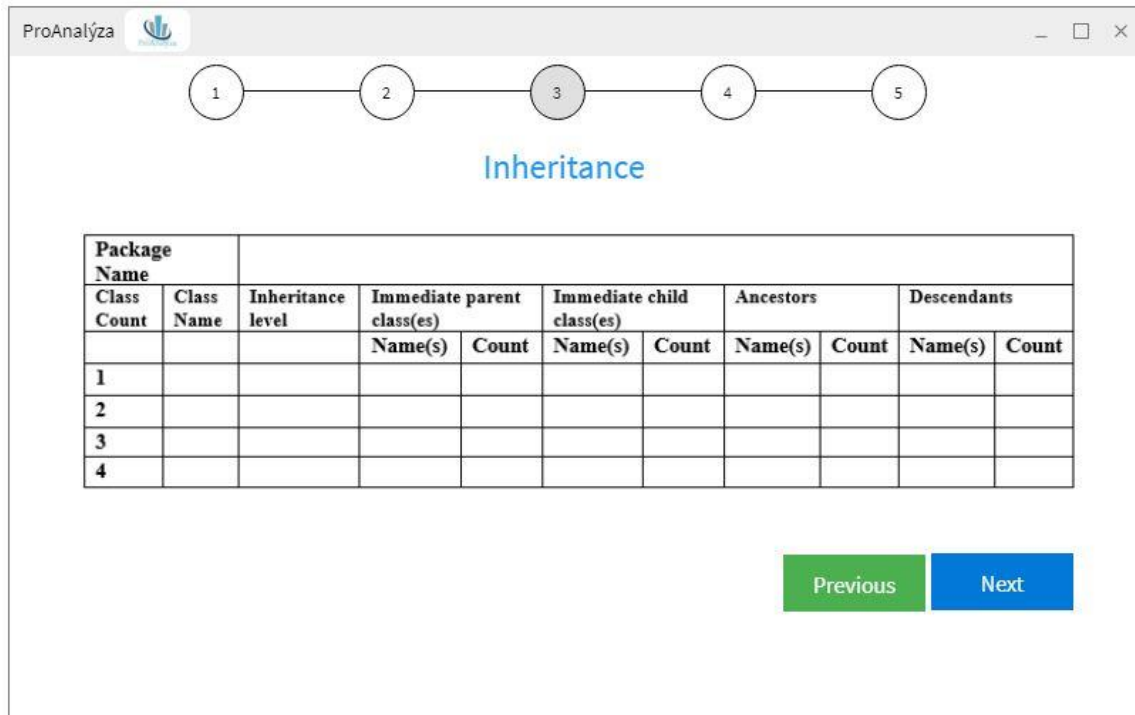


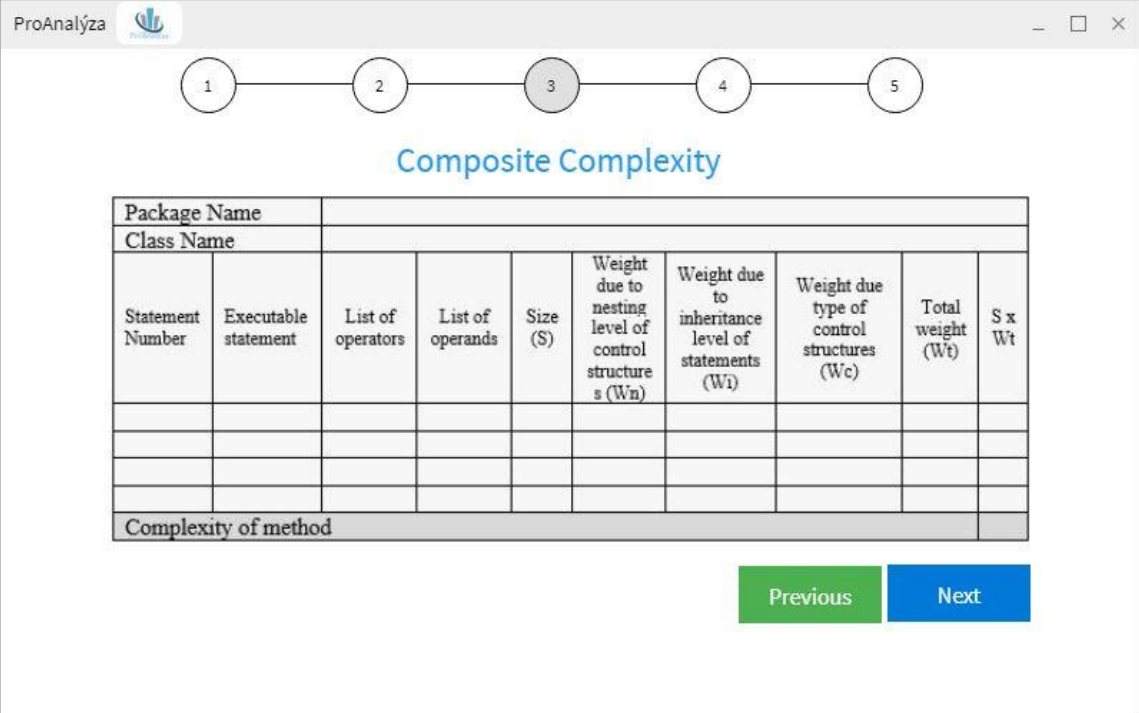
Figure 3.7: User Interface 7

The above GUI indicates only the package wise evaluation related to inheritance. In a similar manner, the user will be able to analyze a program with respect to inheritance, class wise and method wise.

The performance of developer will be determined based on the above results as well as sentiment analysis of user feedback as discussed in section 2.1.2

The composite complexity of a Java program will be analyzed, and results will be displayed as follows.

- Composite complexity of a method



ProAnalýza

1 2 3 4 5


Composite Complexity

Package Name										
Class Name										
Statement Number	Executable statement	List of operators	List of operands	Size (S)	Weight due to nesting level of control structures (Wn)	Weight due to inheritance level of statements (Wi)	Weight due to type of control structures (Wc)	Total weight (Wt)	S x Wt	
Complexity of method										

Previous Next

Figure 3.8: User Interface 8

- Composite complexity of a class

ProAnalýza 

1 — 2 — 3 — 4 — 5


Composite Complexity

Package Name											
Class Name											
Method Name	Statement Number	Executable statement	List of operators	List of operands	Size (S)	Weight due to nesting level of control structures (Wn)	Weight due to inheritance level of statements (Wi)	Weight due type of control structures (Wc)	Total weight (Wt)	S x Wt	Complexity of a method
										5	
										6	
										7	
Complexity of Method A											18
										4	
										5	
										9	
										2	
Complexity of Method B											20
Complexity of class											38

Previous
Next

Figure 3.9: User Interface 9

- Composite complexity of a system

ProAnalýza 

1 — 2 — 3 — 4 — 5

Composite Complexity

Package Name												
Method Name	Statement Number	Executable statement	List of operators	List of operands	Size (S)	Weight due to nesting level of control structures (Wn)	Weight due to inheritance level of statements (Wi)	Weight due type of control structures (Wc)	Total weight (Wt)	S x Wt	Complexity of a method	Complexity of a class
A										5		
A										6		
A										7	18	
B										4		
B										5		
B										9		
B										2	20	
Complexity of class D												38
E										5		
E										6		
E										6	17	
F										4		
F										7		
F										5	16	
Complexity of class D												33
Complexity of the system												71

Previous
Next

Figure 3.10: User Interface 10

3.1.2 Hardware interfaces

As mentioned in section 2.1.3, no specific hardware interface is required for the analysis of a program in terms of inheritance apart from a standard PC/Laptop with minimum requirement of Intel Pentium IV with at least 1.6GHz processor or above and hardware devices such as USB dongle/ADSL router to connect to the internet.

3.1.3 Software interfaces

XAMPP is used to connect to MySQL database and to interpret scripts written in PHP5 programming and a latest updated web browser that supports jQuery, AJAX, HTML5, PHP5, CSS3 and Bootstrap technologies.

A precise explanation is discussed in section 2.1.3.

3.1.4 Communication interfaces

All communication interfaces are required to use ProAnalýza are discussed in section 2.1.4. Accordingly, an internet connection with high bandwidth and a database connection interface will be required to connect with the database.

3.2 Classes/Objects

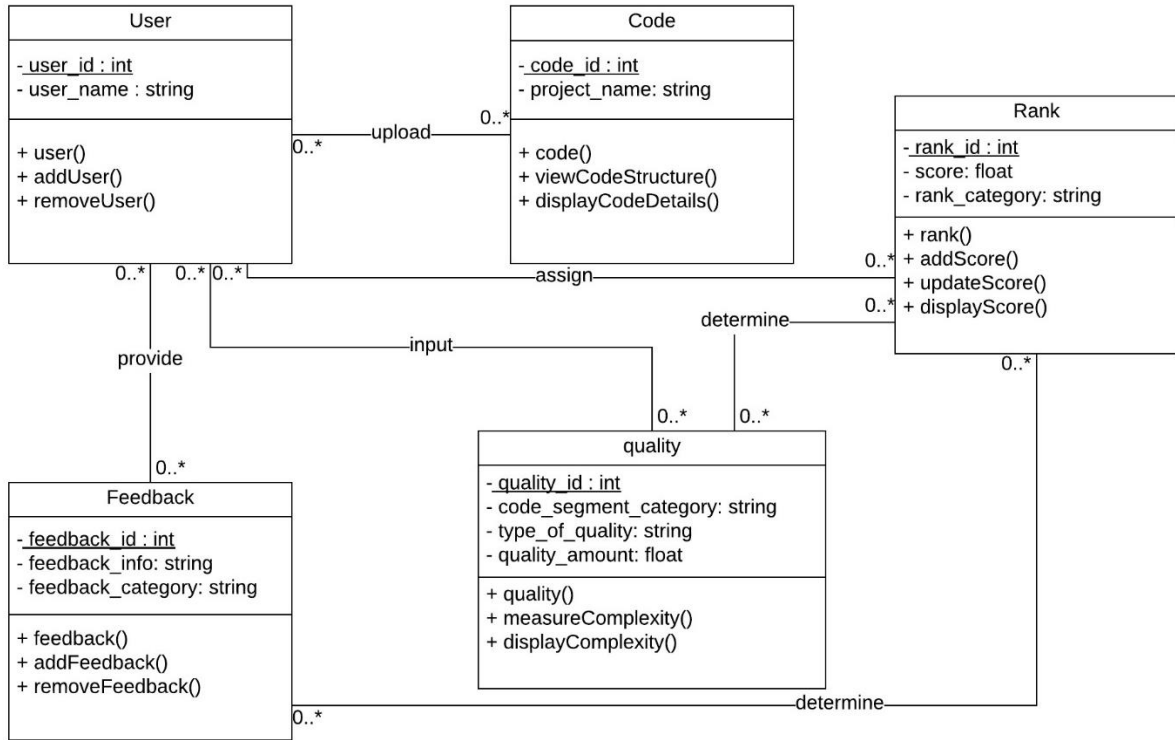


Figure 3.11: Class diagram of ProAnalýza

3.3 Performance Requirements

Since this software is going to web – based, it does require a powerful server machine with high band internet access. Server machine should have a powerful CPU and high-speed internet access so that it can handle multiple users at the same time. Another performance requirement is the storage space. Higher storage space means more user and bigger workspace per user so higher the storage, better the performance. Performance requirement by the user side is, web application should be developed as a lightweight web app so that it can work on almost any platform even with slower internet connections. System should be able to deal with multiple users at the same time.

3.4 Design Constraints

- The system shall be built using open source web development software/s that conforms to Microsoft's GUI standards.
- The computers must be equipped with web browsers such as Internet explorer.
- The product must be stored in such a way that allows the client easy access to it.
- Response time for loading the product should take no longer than five minutes.
- A general knowledge of basic computer skills is required to use the product
- Data should not become corrupted in case of system crash or power failure (fault tolerance).

3.5 Software System Attributes

3.5.1 Reliability

There are no requirements for MTBF or MTTF for this version of the ProAnalýza defined in this SRS. However, in accordance with industry recommended practices, the software system should undergo feature testing, load testing, and regression testing prior to release and/or deployment.

3.5.2 Availability

Reasonable efforts should be made to ensure the ProAnalýza is available with an uptime of 95%. The uptime is calculated as the percentage of time during the year in which the software system was available to the public. A 95% uptime percentage allows for an average of 18.25 days per year, 36 hours per month, or 8.4 hours per week of downtime.

3.5.3 Security

ProAnalýza must follow industry recommended practices for secure software development. At a minimum, the software development must practice the principle of least privilege for defining access-level requirements of the software system and its associated services. The production-release version of the ProAnalýza must pass an automated dynamic application security testing tool.

3.5.4 Maintainability

The architecture, design, implementation, and documentation of the application must minimize the maintenance costs of the software system. The maximum person-time required to fix a security defect (including regression testing and documentation update) must not exceed two person days. Otherwise, the software system must be taken offline or the offending feature disabled. The average person-time required to make a minor enhancement (including testing and documentation update) must not exceed one person week.


3.6 Other Requirements

To improve portability, software should run on variety of platforms and variety of connection speeds. As explained in the performance requirements section, software should be lightweight so that it can run on a machine with slow internet connection. To make the web application lightweight, simple libraries and tools should be used at developing phase. Portability of ProAnalýza can be gain by running on most number of different platform without an additional effort. To achieve this, web application should be developed by using the common technologies and tools which are provided by all common web browsers and operating system such as HTML5, JQuery etc.

ProAnalýza must guide users through an interface based on end user concepts. It must be easy to learn and does not obstruct the thematic understanding of the users and must make it easy to correct mistakes. It must be designed so that it can migrate to upgraded hardware or new versions of the operating systems involved. In addition, it must provide solutions/rules regarding data encoding problems such as supporting different character sets, name truncation rules, name matching in case of misspelling etc.

4 SUPPORTING INFORMATION

4.1 Appendices



Low Memory Usage	★		★	
Static and Dynamic code analysis	★	★		
Affordable			★	
Accurate				★
Comprehensive rules				★
Quality checking	★	★		★
View program structure		★		
Visual Representation	★	★		★
Rate software developers based on their code quality				

Figure 4.1: Comparison of existing static code analyzing tool with reference to the features that included in ProAnalýza

Table 4.1: Tabular representation of Gantt Chart

Task Name	Start	Finish	Duration (days)
Forming Group	12/22/17	12/29/17	7
Identify research topic	12/30/17	01/09/18	10
General research about topic	01/05/18	01/22/18	17
Project topic assessment	01/23/18	02/14/18	22
Project charter preparation and submission	02/15/18	02/23/18	8
Deciding research methodology	02/24/18	03/08/18	12
Project proposal report writing and submission	03/09/18	04/01/18	23
Project proposal presentation	03/19/18	04/04/18	16
(SRS)/Design document preparation and submission	04/09/18	05/07/18	28
Project Status document - SRS preparation	04/10/18	05/08/18	28
Progress presentation – I	05/09/18	06/25/18	47
Research paper	07/11/18	08/06/18	26
Prototype development	04/29/18	07/16/18	78
Progress presentation – II	07/03/18	08/07/18	35
Final report (Draft)	08/20/18	09/01/18	12
Final report (Draft) feedback submission	09/05/18	09/20/18	15
Website Assessment	09/18/18	10/04/18	16
Final Report (Soft Bound)	09/12/18	10/06/18	24
Final Submissions – CD with deliverables	10/03/18	10/28/18	25
Final Presentation	10/23/18	11/12/18	20
Final Report (Group) Hard Bound	10/30/18	11/21/18	22

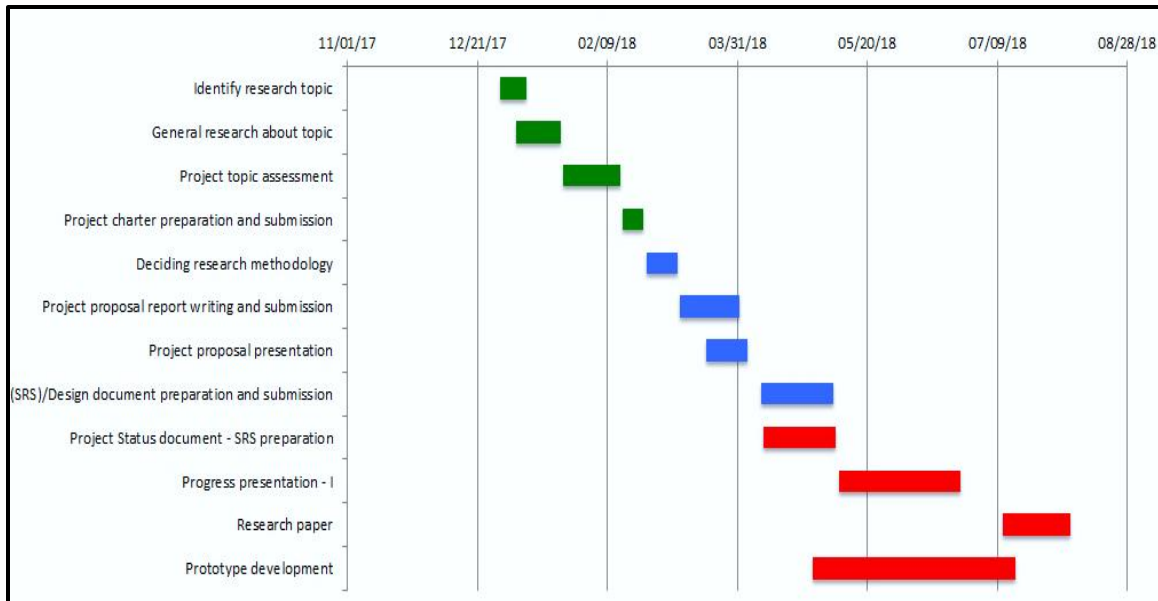


Figure 4.2: Gantt Chart

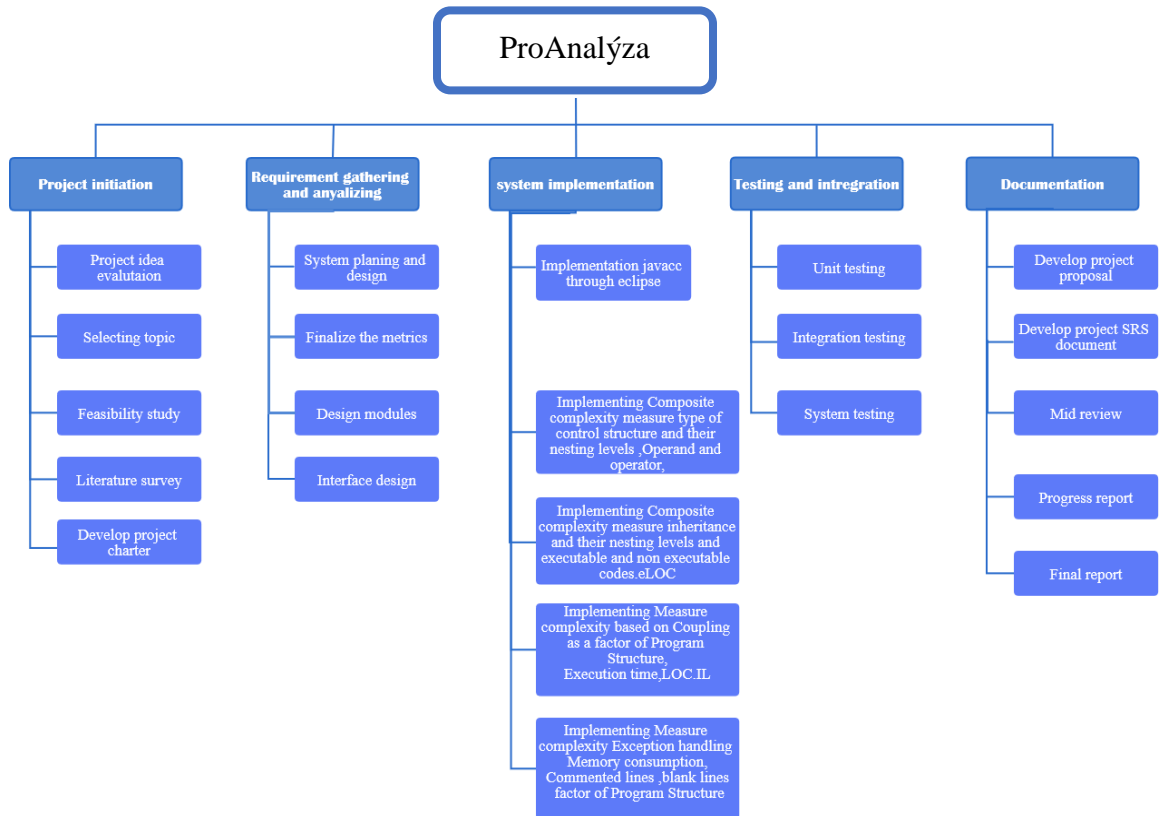


Figure 4.3: Work Breakdown Structure

5 REFERENCES

- [1] Zitser, M., Lippmann, R. and Leek, T., 2004, October. Testing static analysis tools using exploitable buffer overflows from open source code. In ACM SIGSOFT Software Engineering Notes (Vol. 29, No. 6, pp. 97-106). ACM.
- [2] Binkley, D., 2007, May. Source code analysis: A road map. In 2007 Future of Software Engineering (pp. 104-119). IEEE Computer Society.
- [3] Chowdhury, G.G., 2003. Natural language processing. Annual review of information science and technology, 37(1), pp.51-89.
- [4] Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J., 2016. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.
- [5] Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledebor, G., Reynolds, P., Sitaram, P. and Ta, A., 1993, May. Identifying and measuring quality in a software requirements specification. In Software Metrics Symposium, 1993. Proceedings., First International (pp. 141-152). IEEE.
- [6] Mekprasertvit, C., 2003. Software Requirements Specification(Doctoral dissertation, Kansas State University).