

PROGRAM ANALYSIS TOOL

Component – Analyze the program using memory consumption and exception handling also ranking system and feedback system

Project Id: 18-055

Gamaarachchi G.A.C.Y
IT15111548

System Requirements Specification (SRS)

Department of Information Technology
B.Sc. Special (Honors) Degree in Information Technology
Sri Lanka Institute of Information Technology

Submitted on 15-05-2018

PROGRAM ANALYSIS TOOL

Project Id: 18-055

Supervisor

.....

Dr. Dilshan de Silva

Declaration

I hereby declare that the submitted project Software Requirements Specification document for Program Analysis Tool is an original work done by Gamaarachchi G.A.C.Y. This document is proprietary and an exclusive property of the SLIIT project group 18-013. List of references I referred for the preparation of this document are given as references at the end of the document.

Member: IT15111548 – Gamaarachchi G.A.C.Y

Signature:

Abstraction

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. And this function focus on about analyze the program according to memory consumption and exception handling give the feedback and develop the ranking system

Table of Contents

Declaration.....	2
List of Tables	6
List of Figures	6
1. Introduction	7
1.1 Purpose.....	7
1.2 Scope.....	7
1.3 Definitions, Acronyms and Abbreviation	8
1.4 Overview.....	9
2. Overall Description.....	10
2.1 Product Perspective.....	12
2.1.1 System interfaces.....	14
2.1.2 User Interfaces.....	15
2.1.3 Hardware interfaces.....	20
2.1.4 Software interfaces	20
2.1.5 Communication Interfaces	20
2.1.6 Memory Constraints.....	20
2.1.7 Operations	21
2.1.8 Site Adaptation Requirements.....	21
2.2 Product Functions	22
2.3 User Characteristic.....	24
2.4 Constraints	25
2.5 Assumptions and Dependencies	25
2.6 Apportioning of Requirements	25
3. Specific Requirements (For Object Oriented Products)	26
3.1 External Interface Requirements.....	26
3.1.1 User Interfaces.....	26
3.1.2 Hardware Interfaces	27
3.1.3 Software Interfaces.....	28
3.1.4 Communication Interfaces	28
3.2 Classes/Objects	28
3.3 Performance Requirements.....	29

3.4	Design Constraint.....	29
3.5	Software System Attributes	30
3.5.1	Reliability	30
3.5.2	Availability.....	30
3.5.3	Security.....	30
3.5.4	Maintainability	30
3.6	Other Requirements	31
4.	Supporting Requirements	31
4.1	Appendices.....	31
	References.....	34

List of Tables

<i>Table 1: Definitions, Acronyms and Abbreviation</i>	<i>8</i>
<i>Table 2: Use Case Scenario 1 - User Registration.....</i>	<i>22</i>
<i>Table 3: Use Case Scenario 2 - Upload source code</i>	<i>23</i>
<i>Table 4: Use Case Scenario 3 - Input expected quality standards</i>	<i>23</i>
<i>Table 5: Use Case Scenario 4 - Provide user feedback to determine developer's performance rate</i>	<i>24</i>

List of Figures

<i>Figure 1: High Level System Architecture.....</i>	<i>10</i>
<i>2. Figure 2: Logo of SonarQube</i>	<i>12</i>
<i>Figure 3: Logo of JArchitect.....</i>	<i>13</i>
<i>Figure 4: Logo of FindBugs</i>	<i>14</i>
<i>Figure 5: ProAnalýza Home Page.....</i>	<i>15</i>
<i>Figure 6: ProAnalýza Login Page</i>	<i>15</i>
<i>Figure 7: Upload Source Code</i>	<i>16</i>
<i>Figure 8: Input expected quality standards</i>	<i>17</i>
<i>Figure 9: Input expected quality standards</i>	<i>17</i>
<i>Figure 10: Tabular representation of complexity metrics evaluation of source code.....</i>	<i>18</i>
<i>Figure 11: Provide user feedback to determine developer's performance rate</i>	<i>18</i>
<i>Figure 12: Performance evaluation of the developer.....</i>	<i>19</i>
<i>Figure 13: View developer's performance analytics</i>	<i>19</i>
<i>Figure 14: Class diagram of ProAnalýza</i>	<i>28</i>
<i>Figure 15: Comparison of existing static code analyzing tool with reference to the features that included in ProAnalýza.....</i>	<i>31</i>
<i>Figure 16: Tabular representation of Gantt Chart.....</i>	<i>32</i>
<i>Figure 17: Gantt chart.....</i>	<i>33</i>
<i>Figure 18: Work Breakdown Structure.....</i>	<i>34</i>

1. Introduction

1.1 Purpose

The purpose of this Software Requirement Specification Document (SRS) is to provide a detailed description of the functionalities of ProAnalýza, a tool to analyze the program and rank the programmer according to their code complexity, measure and effectively display complexity of code using complexity metrics, clearly identifying complex code segments and possible solutions to complexity. This document will primarily cover the system's purpose, intended features while mainly focusing on how the tool will calculate complexity based on Exception. It will also explain the constraints, interfaces, hardware, software and other dependencies of the system. The document is primarily intended for the Supervisor and Subject Coordinator to serve as a reference document while developing this system, but also be of interest to any parties invested in developing the code analysis tool or testers and evaluators. The document is written in a form that any person can read and understand the content while it will also be useful for former researchers who are interested in not only implementing similar code analysis tools but also interested in the study of code complexity and document will cover each of the systems purpose and features and will prioritize on Exception handling and Literature Review on impact of exception handling on complexity. This will also cover the constraints, interfaces, hardware, software and other dependencies of the system. The document is written in a form that any person can read and understand the content.

1.2 Scope

The code complexity analysis tool, ProAnalýza, will be developed with two primary objectives in mind. They are as follows,

Objective 1: To create a comprehensive tool to analyze and increase the maintainability of the software, measure and effectively display complexity of code using complexity metrics, clearly identifying complex code segments and plan the productivity and maintainability of the software product.

Objective 2: To create a comprehensive tool that will check the quality of a program to industry or company specific quality standards. The standards will be selectable by the user through an interactive interface and developers rank according there productivity.

Based on the above objectives, the tool will consist of 3 dominant modules. The program complexity measuring and displaying module and the quality standard adherence checking module and ranking system and effective feedback given system

The code complexity measuring module is divided into core components based on what metrics will be used to measure the complexity of code. They are as follows,

1. Coupling
2. Exception handling
3. Inheritance
4. Control Structures
5. CLOC
6. LOC
7. Operand operators
8. Memory consumption

This document will touch briefly on the quality standard adherence checking module, while from the complexity measuring module, the document will describe in detail the “Code parsing with ANTLR & analyze code in terms of exception handling & memory consumption and ranking system” core component. The main objective of this component will be to identify the most important complexity metrics related to exception handling and memory consumption and also develop the ranking system those metrics to accurately and reliably calculate the complexity of a program based on exception handling and memory consumption.

1.3 Definitions, Acronyms and Abbreviation

Table 1: Definitions, Acronyms and Abbreviation

Term	Definition
SRS	Software Requirements Specification
GUI	Graphical User Interface

JVM	Java Virtual Machine
MySQL	My Structured Query Language.
IDE	Integrated Development Environment
HTML	Hyper Text Markup Language
AJAX	Asynchronous JavaScript And XML
IIS	Internet Information Services
TCP/IP	Transmission Control Protocol/Internet Protocol
RAM	Random Access Memory

1.4 Overview

This SRS document intends to cover all the functional and non-functional requirements of the ProAnalýza, “Analyze the program using memory consumption and exception handling also develop ranking system for developers” core component.

The remainder of this document includes two more chapters, appendixes and references. The second chapter provides an overall description of the system functionality and system interaction with the complete system and product functions through use case stories and user characteristics. This chapter also introduces different types of stakeholders/users and their interaction with the system. Furthermore, the chapter includes the system constraints, assumptions and dependencies.

The third chapter provides the specific requirements of the system and a description of the different system interfaces. Additionally, performance requirements, design constraints and software system attributes will also be discussed here. Appendix will include a detailed class diagram of the system and other supporting information.

The tool will mainly consist of a desktop application that will initially present the user with an interface that will allow him/her to “parse” his program code into the tool. An ANTLR plugin will be used to handle the parsing of the program code. After which the user will be

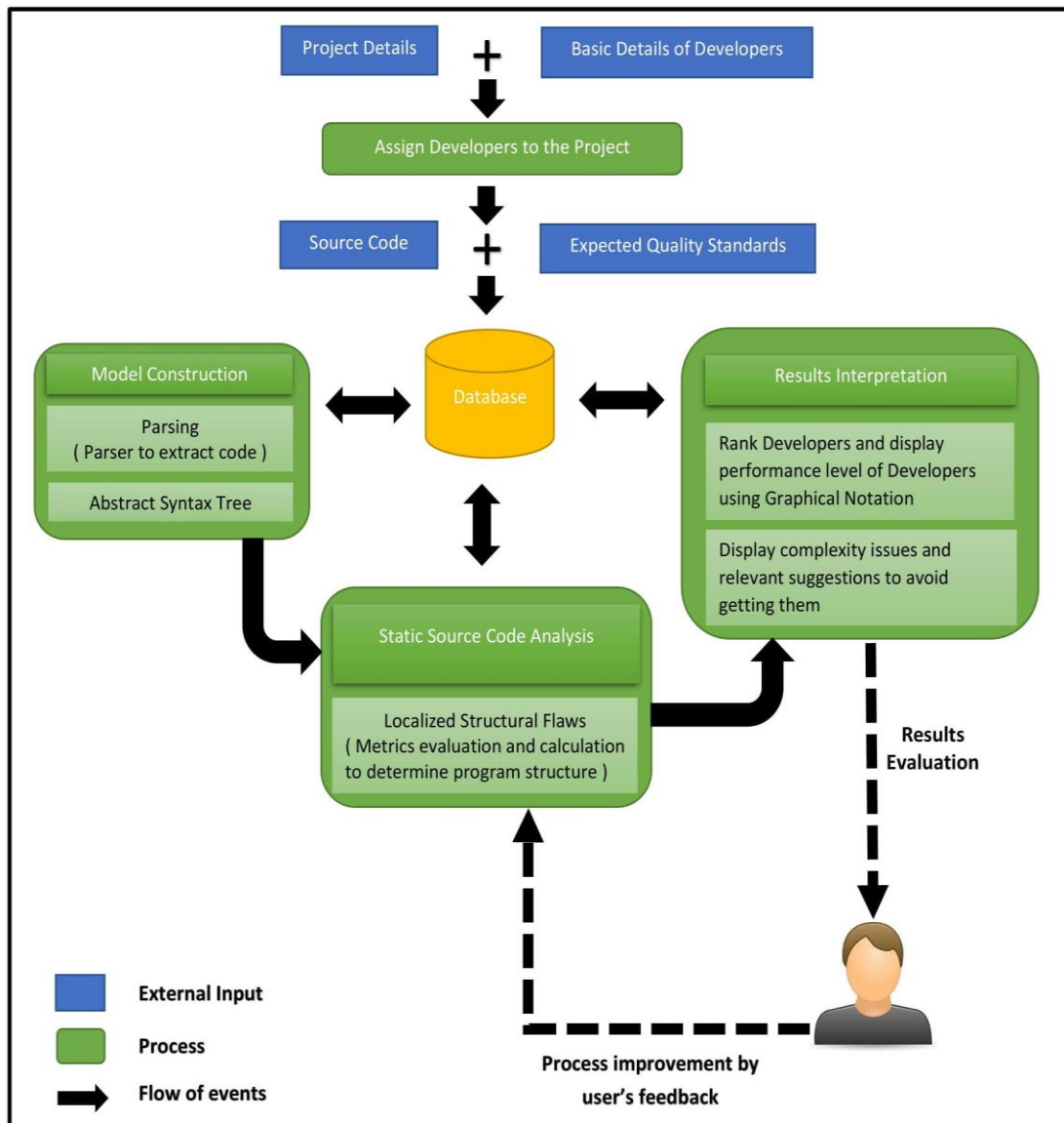
presented with the option of selecting whether he/she wishes to check their code for code quality adherence, and if so, the user will be presented with an interface that will allow him/her to select what aspects of code quality he/she wishes to check their code for. Finally, the user will be presented with the option of selecting which core component of complexity he/she wishes to measure his code for (that is, in relation to Coupling, Inheritance, Control Structures or Exception Handling).

2. Overall Description

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 of the system will be presented.

The system will be made up of four primary features as detailed in the high level system architecture below,

Figure 1: High Level System Architecture



The Data Ingestion level of the system is where the program code is fed to the tool. This is also one of two layers where the user has direct interaction with the tool. This will also be where the user is presented with the option of checking his/her program code for quality standards adherence. Once the program code has been fed to the tool, through the use of ANTLR, the tool will capture key points in the code (such as object names, class names, declarations, control structures, etc.) and create a parser tree. These captured key metrics will then be used to calculate complexity. The user will also state in relation to which core metric he/she wishes to calculate complexity.

Depending on the user's core metric preference, the metrics captured during Data Ingestion will be passed into the relevant core metric algorithm. The tool will be calculating complexity based on Program Structure and metrics derived from it. After much literature review, the

following 4 metrics have been considered as the best metrics upon to base the complexity on in relation to program structure,

1. Coupling
2. Exception handling
3. Inheritance
4. Control Structure Types and Nesting level

The Data Storage layer speaks for itself and will be the point in which all values, both results and raw data, will be stored. The Reporting and Output level is the second layer that presents the user with direct interaction with the tool. The user will be able to see a visual representation of the output in the following forms,

1. Annotated program structure tree
2. Graphical and Tabular representations of Complexity
3. Degree of program adherence to quality standards
4. Suggested Complexity Solutions annotated in the original code

2.1 Product Perspective

1. SonarQube

2. Figure 2: Logo of SonarQube



- Upgrading the version of the server is a bit cumbersome and could be made slightly easier. Allowing admin users to upgrade the software through the front-end would make upgrading easier.
- Another improvement is with false positives. Sometimes the tool can say there is an issue in your code but, really, you have to do things in a certain way due to external dependencies, and it's very hard to indicate this is the case. There is a way to mark the code/method with the issue number, but having to add comments/annotations in your code for your static analysis tool gives a bad user experience in UI/UX perspective.
- Unable to have different groups or projects within a single SonarQube server for different environments in a project development life cycle.
- A better design of the interface and add some new rules.
- When we have a thousand products published over it, we expect it to be more efficient in terms of serving requests from the browser.
- Ease of use/interface.
- It requires advanced heuristics to recognize more complex constructs that could be disregarded as issues.
- There is need for support for the additional languages and ease of use in adding new rules for detecting issues.

3. JArchitect

Figure 3: Logo of JArchitect



Limitations:

- Documentation issues.
- Cost of small developers

4. FindBugs

Figure 4: Logo of FindBugs



Limitations:

- FindBugs can report false warnings, which are warnings that do not indicate real errors.
- Hard to write code and maintain.

2.1.1 System interfaces

Windows 10 operating system will be used as the development platform. One of the major technologies 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

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

Figure 5: ProAnalýza Home Page

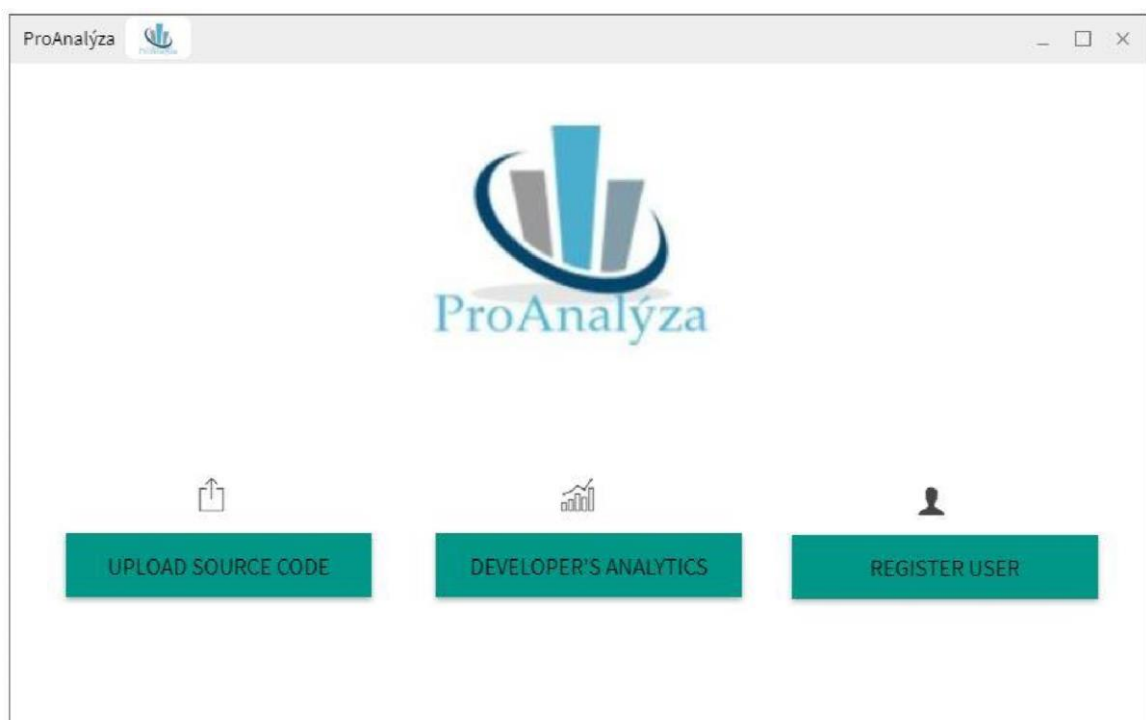
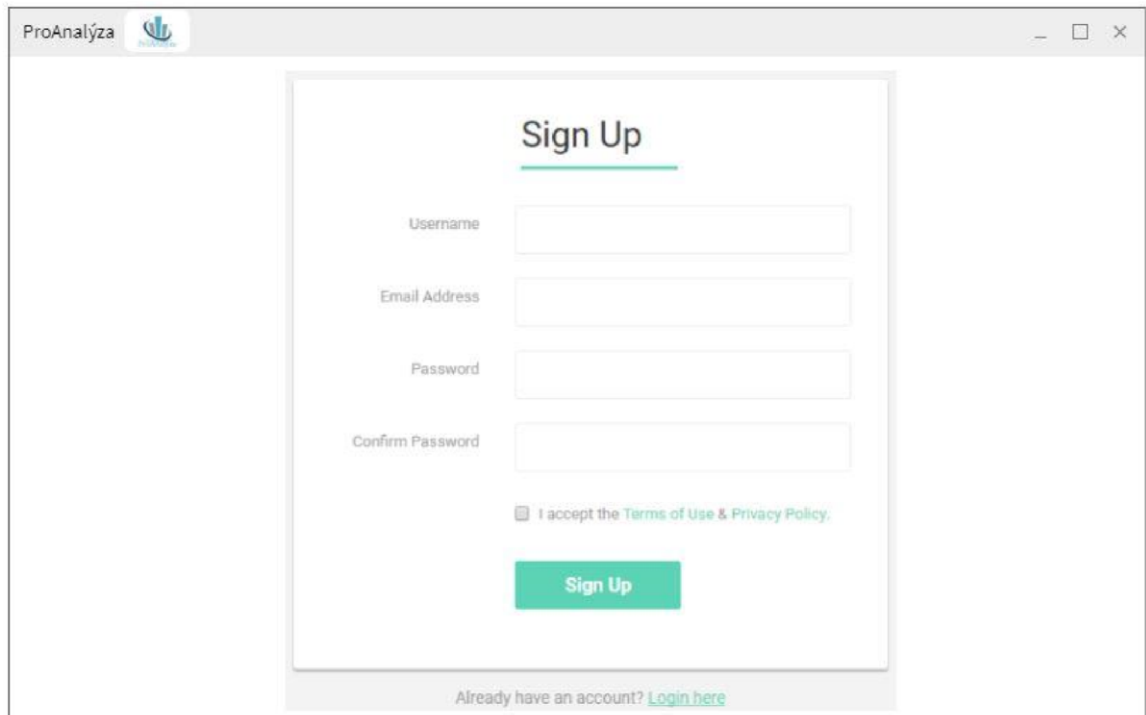


Figure 6: ProAnalýza Login Page



The image shows a web browser window titled "ProAnalyza" with a "Sign Up" form. The form is centered and contains the following elements: a title "Sign Up" with a green underline, four input fields for "Username", "Email Address", "Password", and "Confirm Password", a checkbox labeled "I accept the Terms of Use & Privacy Policy.", a green "Sign Up" button, and a link "Already have an account? Login here" at the bottom.

ProAnalyza

Sign Up

Username

Email Address

Password

Confirm Password

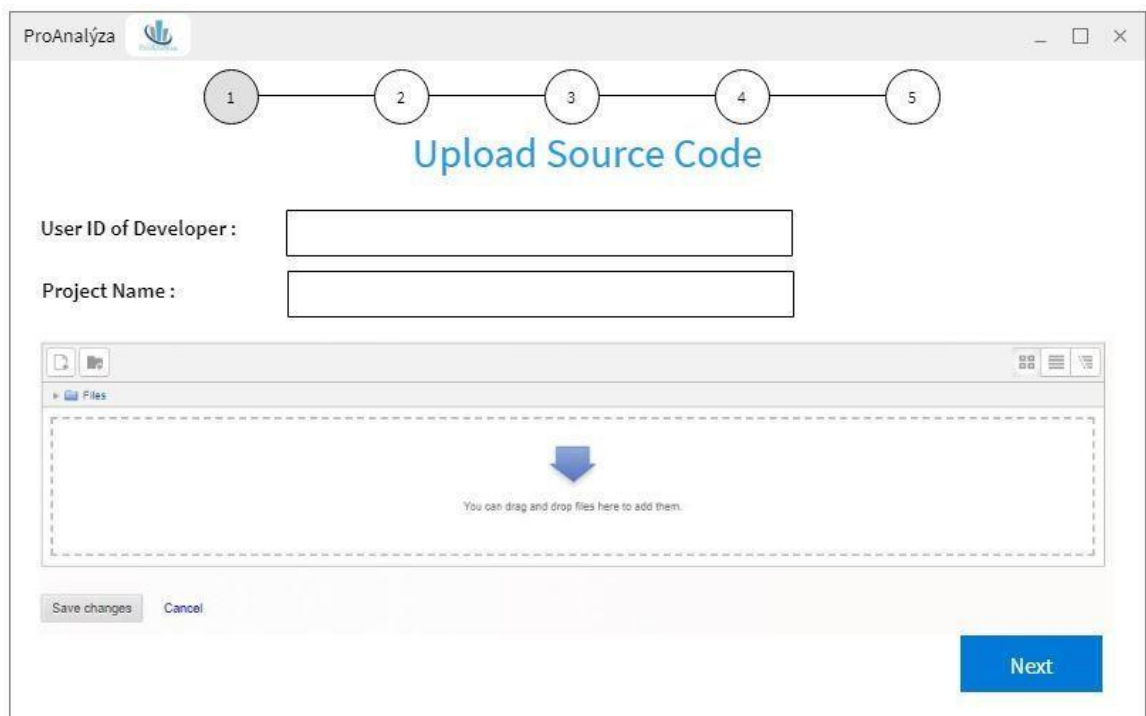
☐ I accept the [Terms of Use & Privacy Policy.](#)

[Sign Up](#)

Already have an account? [Login here](#)

2. Upload source code

Figure 7: Upload Source Code



The image shows a web browser window titled "ProAnalyza" with an "Upload Source Code" form. The form is centered and contains the following elements: a progress bar with five steps (1, 2, 3, 4, 5), a title "Upload Source Code" in blue, two input fields for "User ID of Developer" and "Project Name", a file upload area with a dashed border and a blue arrow pointing down, and a "Next" button at the bottom right. The file upload area also contains the text "You can drag and drop files here to add them." and buttons for "Save changes" and "Cancel" at the bottom left.

ProAnalyza

1 2 3 4 5

Upload Source Code

User ID of Developer :

Project Name :

You can drag and drop files here to add them.

[Save changes](#) [Cancel](#)

[Next](#)

3. Input expected quality standards

Figure 8: Input expected quality standards

The screenshot shows the 'ProAnalyza' application window with a progress bar at the top containing five circles, where the second circle is highlighted. Below the progress bar, the title '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 currently showing 'Select'. To the right of the second dropdown is a grey 'Add' button. Below these elements is a large, empty rectangular box for listing standards. At the bottom right of the form are two buttons: a green 'Previous' button and a blue 'Next' button.

Figure 9: Input expected quality standards

This screenshot shows the same 'ProAnalyza' application window, but now the quality standards have been added. The 'Category of code segment :' dropdown now shows 'Class'. The 'Quality standards required :' dropdown still shows 'Select', and the 'Add' button remains. The large rectangular box below now contains three items, each with a green text label and a grey circle with an 'X' to its right: 'Composite Complexity', 'List of Operators/Operands', and 'Inheritance'. The 'Previous' and 'Next' buttons are still present at the bottom right.

4. Tabular representation of complexity metrics evaluation of source code

Figure 10: 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

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

Figure 11: 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 :

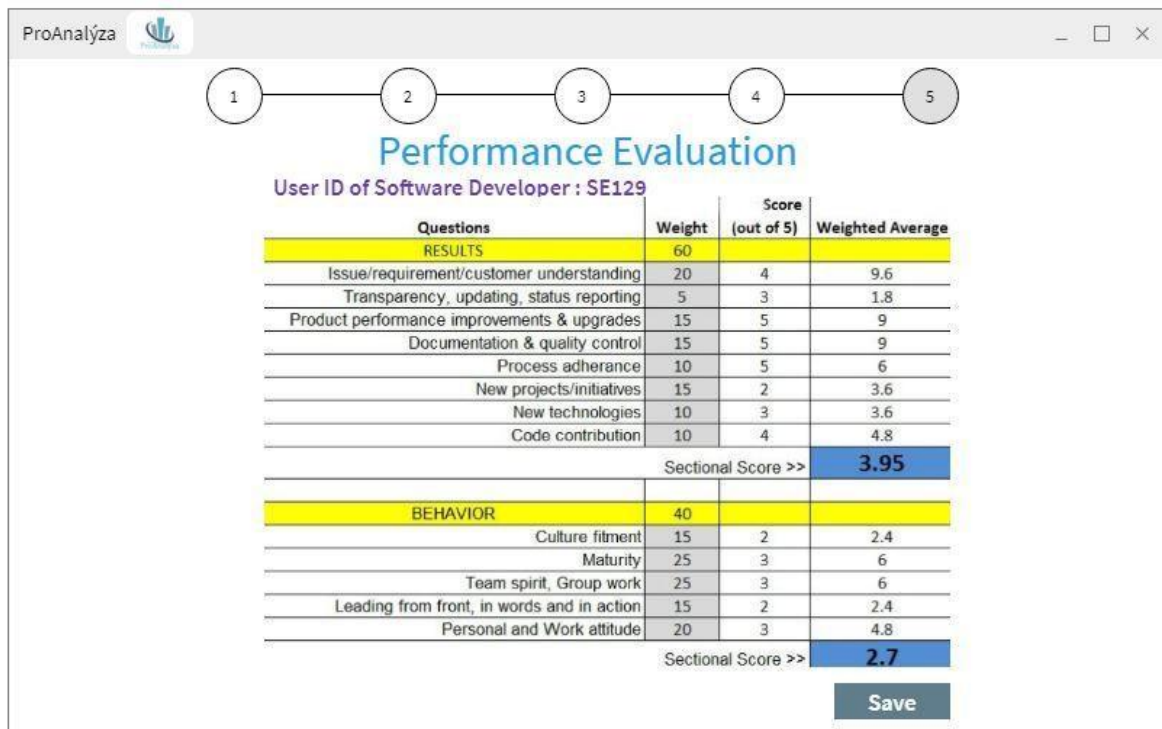
2. Transparency,updating,updating status reporting :

3. Project performance improvements & upgrades :

Previous Next

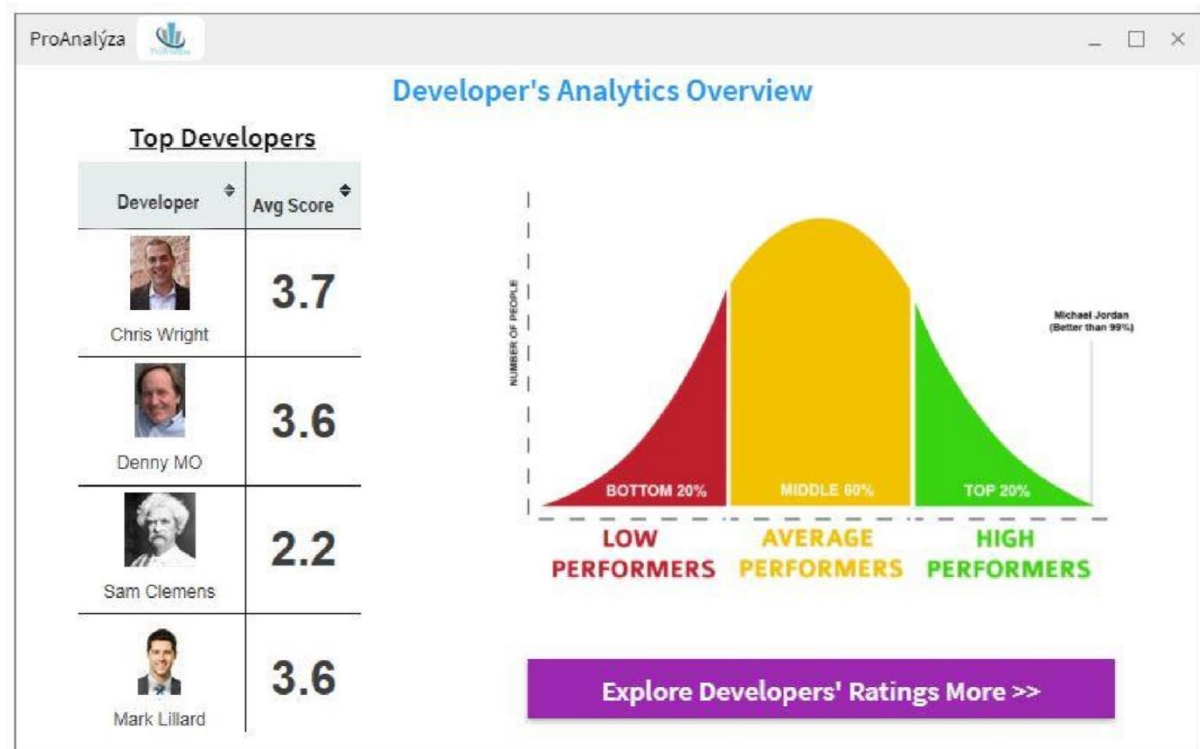
6. Performance evaluation of the developer

Figure 12: Performance evaluation of the developer



7. View developer's performance analytics

Figure 13: View developer's performance analytics



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 web.

2.1.4 Software interfaces

Following software interfaces are required in order to use pro Analýza. Calculating Complexity based on coupling feature,

- MySQL
- Eclipse IDE Interface
- Latest updated web browser that supports JQuery and AJAX Eg: Internet Explorer version 5.0 or above, Mozilla Firefox version 3.5 or above, Google Chrome etc.
- ANTLR Interface
- Java Library

2.1.5 Communication Interfaces

Following communication interfaces are required to use ProAnalýza.

- 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 ProAnalýza 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.

Example: User interface must exist in three different languages: Hungarian, Serbian, and Slovak.

2.2 Product Functions

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

Table 2: Use Case Scenario 1 - User Registration

Use Case ID:	1
Use Case Name:	User registration
Actors:	User
Pre-conditions:	Program successfully launched Database connection is live

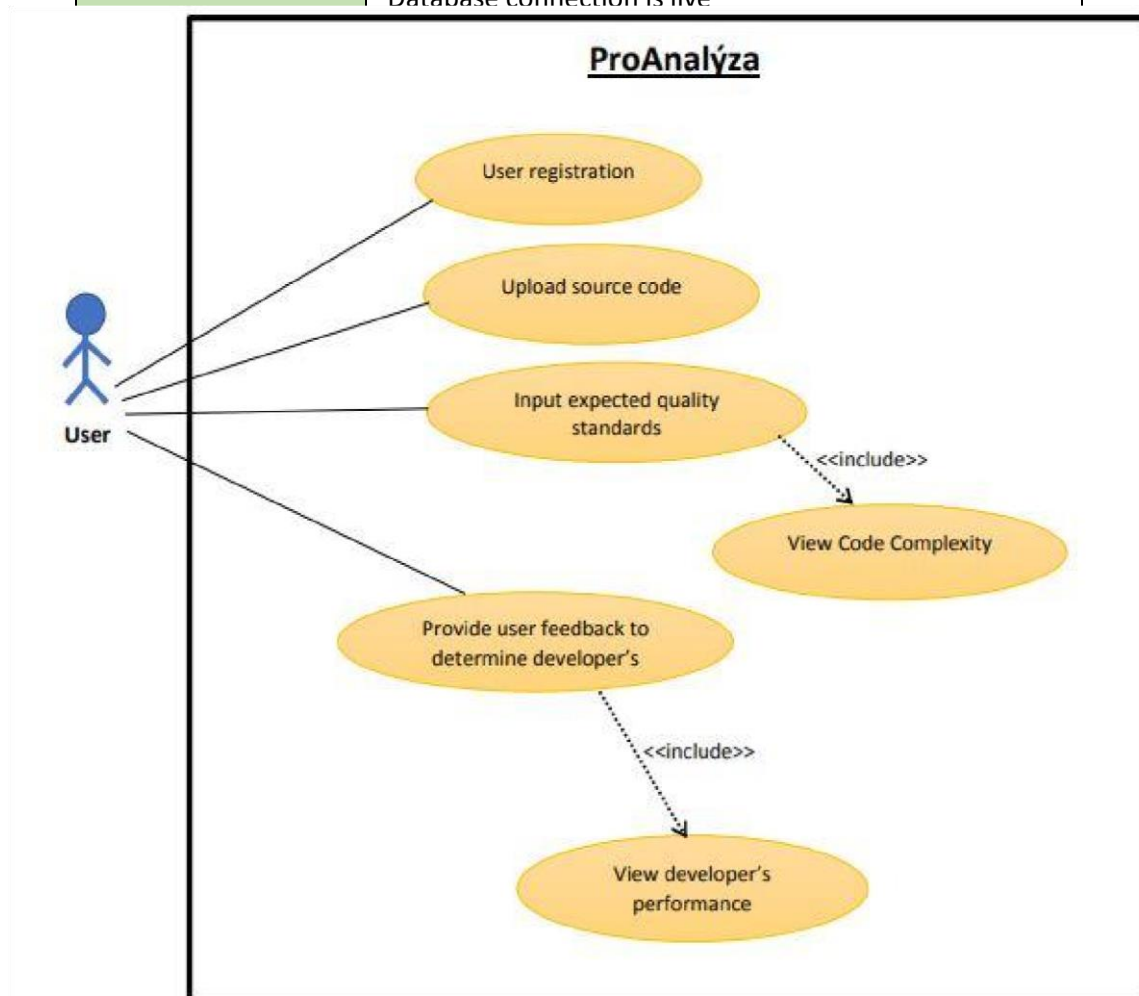


Table 3: Use Case Scenario 2 - Upload source code

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 4: Use Case Scenario 3 - Input expected quality standards

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 5: Use Case Scenario 4 - Provide user feedback to determine developer's performance rate

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 Characteristic

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 (For Object Oriented Products)

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

Below is the descriptive information of the user interfaces described in section 2.1.1.

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

ProAnalýza Login Page - 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.

Upload Source Code - When the user successfully login to the system he/she will be able to upload source code from the GUI as above figure 7.

Input expected quality standards - Then the user can click “Next” button to input the expected quality standards from the GUI as above figure 8. If the user required analyzing the program in terms of inheritance, the user must select criteria as aforementioned and which is given in the GUI as above figure 9.

Tabular representation of complexity metrics evaluation of source code - The composite complexity of a Java program will be analyzed, and results will be displayed as above figure 10.

Provide user feedback to determine developer’s performance rate – Developers can give feedbacks to their colleagues.

Performance evaluation if the developer – According to the feedbacks finally gives the each developer’s performance evaluation as above figure 12.

View developer’s performance analytics – Developers can view their own performance analytics.

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

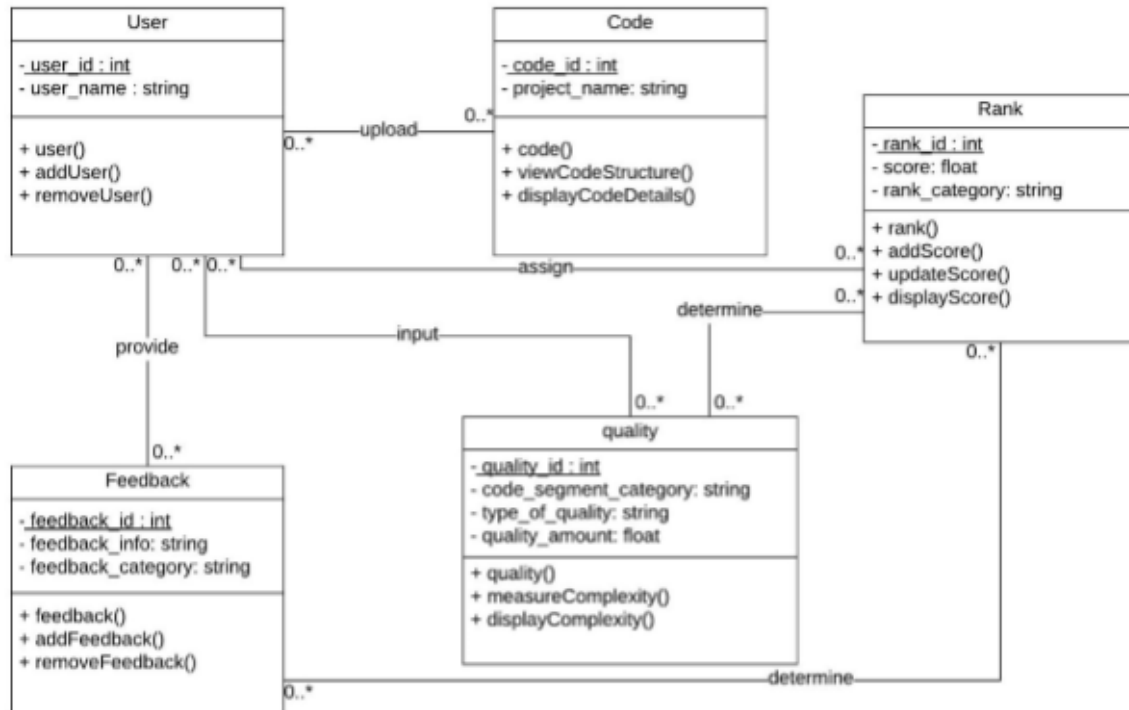
XAMPP is used to connect to MySQL database and to interpret scripts written in PHP5 programming and a latest updated web browser that supports jQuery, AJAX, HTML5, PHP5, CSS3 and Bootstrap technologies. A precise explanation is discussed in section 2.1.3.

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 14: 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 Constraint

- 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 platforms 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 Requirements

4.1 Appendices

Figure 15: Comparison of existing static code analyzing tool with reference to the features that included in ProAnalýza



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 16: 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 17: Gantt chart

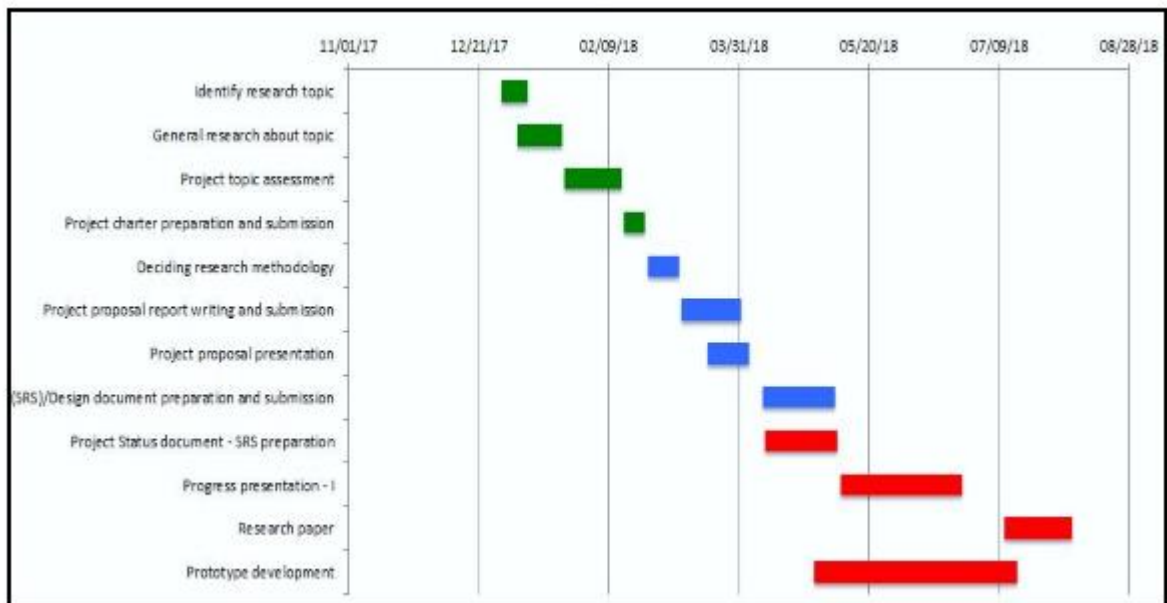
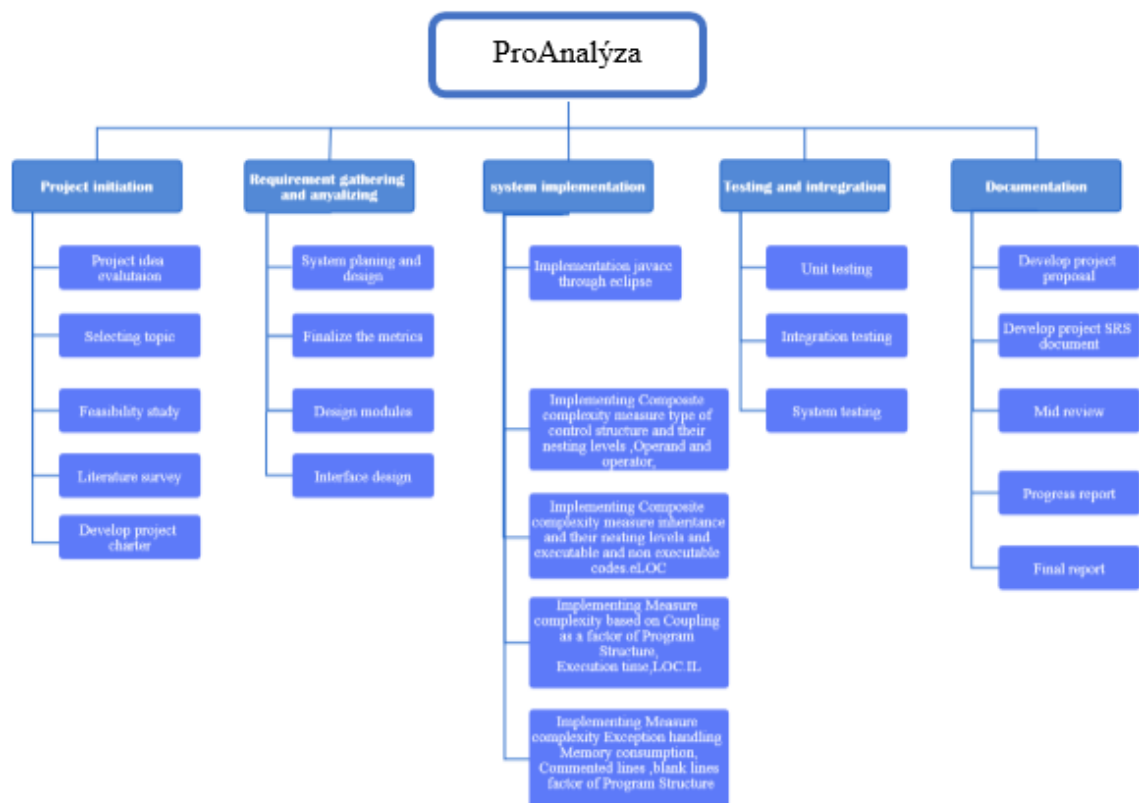


Figure 18: Work Breakdown Structure



References