Software Requirements Specification (SRS)

Based on IEEE 830 -Adapted for Student Projects

Project Title: Python code analyzer

Team Name & Members:

Zahra Bader Hayyan 2240006156

Hadeel abduallah alqhtani 2240003327

Noor Almuhsen 2240003505

Yomna Al moslem 2240003445

Alzahraa alabbad 2240006089

Eman Alnajem 2240002468

Wedad Mohammed AL-hussaini 2240002853

Advisor: Rahmah alzhrani

Version: 1.0

Date: 8-10-2025

Table of Contents

1. Introduction

1.1 Purpose

This document explains the **Python Code Analyzer**, its main features, and how it works in a simple way. It is created to help students and beginners understand the system, for developers to see how each part is built, and for testers to follow the project and check the tests. The project provides a web-based tool that analyzes Python code and generates a structured report. We developed it to make it easier to check code quality and help developers improve their work efficiently.

1.2 Document Conventions

Style Name: Document Naming
All project documents use the same format for names:
CourseCode_Section_Group#_DocumentName.

Style Name: File Naming
Python files are written in lowercase letters with underscores between words.

Diagram Style:
We used UML diagrams to show how the system works and how the parts are connected.

All the documents were made using Microsoft Word and shared with the team through GitHub.

1.3 Intended Audience and Reading Suggestions

• This document is mainly for **students and beginners** and who want to understand how our Python Code Analyzer works and learn from it. It is also a key reference for developers and testers involved in the project.

1. .**Students and Beginners:** Their main goal is to understand how the Python Code Analyzer works and use it for learning. They can read the introduction and scope sections to understand the main idea of the tool.
2. .**Developers:** Use this document to understand how each part of the system is built and connected. They should read all sections to know every detail of the project. can focus on the planning and design parts.
3. .**Testers:** This document also helps testers who verify the accuracy and reliability of the Python Code Analyzer. They use it to understand expected system behavior, prepare test cases, and confirm that the analysis results and reports are generated correctly. Testers should focus on Sections 3 and 5 to review functional and non-functional requirements for validation purposes

1.4 Scope of the System

The system will provide the following capabilities:
a. Reading and checking Python code files.
b. Showing the number of lines, functions, and classes.
c. Handling simple code errors without crashing.
d. Giving a clear and easy-to-read report.

The following items are excluded from the project scope:
a. Detection of logical errors
b. Code execution
c. Practical performance optimization
d. Integration and external services
e. Code generation
f. Running on mobile devices

1.5 References

1. IEEE830 - SRS.pdf
2. SRS_Template_IEEE830-Students.docx
3. Python Documentation – Python Software Foundation

2. Overall Description

2.1 Product Perspective

The Python Source Code Analyzer is a web-based tool designed to analyze Python files and generate a structured report. It functions as an independent web application that can be expanded in the future to interface with IDEs or code editors, but it is not a part of a larger system.

The main goal is to make it easier for students and developers to understand their code by showing useful information such as the number of functions, classes, and imports. It also points out possible issues and gives suggestions to make the code cleaner and more readable.

In future updates, the tool could be extended to analyze multiple files at once or integrate directly with GitHub or text editors for live code analysis.

2.2 Product Functions

The main function of this tool is to read and analyze Python source code files and generate a clear report that summarizes the main details about the code. It gives an overview of the total lines, number of classes, functions, and imported libraries.

The tool also provides more detailed information, such as each function's name, parameters, and length, along with the classes and their methods. It can detect issues like unused imports, missing docstrings, or code that might be hard to read or maintain.

It also includes a nesting analysis to check how many times loops, conditionals, or functions are nested inside each other, which helps identify complex parts of the code.

At the end of the analysis, the system generates an HTML report that includes a summary, details, detected issues, and suggestions to improve the quality of the code.

2.3 User Classes and Characteristics

This tool is mainly designed for two types of users:

1. Students / Beginner Programmers:

   Students or beginner programmers might have limited experience with python and coding in general. They can use the tool to understand their code better and learn how to write cleaner and more organized programs. The feedback and suggestions from the tool can help them improve their coding skills and follow good programming practices.

2. Developers / Instructors:

   These users have more experience with programming. They can use the tool to quickly check the quality of the python code, analyze student submissions, or review their own work. It helps them save time when evaluating or maintaining multiple python files.

3. Testers / Quality Assurance Members:

   Testers ensure that the Python Code Analyzer performs as expected. They validate that each function, report, and interface works correctly under different conditions. They also confirm that the tool handles syntax errors gracefully and generates accurate and complete HTML reports. Basic knowledge of software testing and Python is required

All users are expected to have basic knowledge of python syntax and be able to open and read code files.

2.4 Operating Environment

The Python Source Code Analyzer will run on any standard PC or laptop capable of running Python. It is designed to work as a simple web-based tool that doesn’t require complex installation.

Hardware Requirements:

- A standard computer or laptop

Software Requirements:

- Python 3.x
- Libraries: ast, inspect, os, pathlib, radon, jinja2

Operating Systems Supported:

- Windows
- macOS
- Linux

The output of the tool will be generated in HTML format, which can be viewed using any web browser.

2.5 Design and Implementation Constraints

- The tool must be developed using Python 3.x.
- The system shall be implemented as a web-based application, where analysis and report generation are performed on the server and accessed through a browser interface.
- It should only use built-in or open-source libraries.
- The output report must be generated in HTML format for easy viewing in any browser.
- The system should follow Python PEP8 standards to keep the code clean and readable.
- The project must be completed within the course timeline and submitted before the final deadline.
- The design should be simple and user-friendly, so users can analyze their code easily without advanced setup

2.6 Assumptions and Dependencies

- Users are expected to have Python 3.x installed on their computers before running the tool.
- The input files are assumed to be valid Python (.py) files with correct syntax.

- The tool depends on several Python libraries such as ast, inspect, os, pathlib, radon, and jinja2 for analysis and report generation.
- The system requires an internet connection only for the first installation of these libraries.
- It is assumed that users have basic knowledge of Python and can open and run the tool from their computer.
- The output (HTML report) depends on the browser compatibility, so it should be viewed in an updated web browser.

3. Specific Requirements

3.1 Functional Requirements

This section describes the functionality and system services of the Python Code Analyzer.

The functional requirements depend on the needs of developers, students, and beginners who use the tool to analyze Python code and generate reports.

The following subsections describe both the high-level user requirements and the detailed system requirements of the tool.

3.1.1 File Analysis

- The system shall allow users to upload a Python file for analysis.
- The system shall extract the number of lines, functions, and classes.
- The system shall identify all imported libraries.

3.1.2 Report Generation

- The system shall generate a structured HTML report.
- The system shall summarize key metrics such as complexity and maintainability.
- The system shall display notes and recommendations for code improvement.

3.1.3 Error Handling

- The system shall detect syntax errors and continue analysis without crashing.
- The system shall notify the user of any invalid file format.

3.1.4 User Interaction

- The system shall allow users to view and download the report.
- The system shall allow re-analyzing files after modification.

3.2 Use Case Specifications

In this section, we will describe use case details: actors, preconditions, main flow, alternative flows, and postconditions.

Use Case	Actors	Preconditions	Main Flow	Postconditions
Upload file	User	User has a (.py) file ready	User opent the analyzer page →user clicks "choose file"→system veridies file type →system display "file uploaded successfully"	File is stored temporarily on the server during analysis and automatically deleted after processing
Analyze code	User	File that has been uploaded successfully	User click "analyzer" → system parses the file → system start count lines ,functions,classes,and libraries → system detects syntax errors if any	Analysis results are ready in the memory
Generate report	System	Analysis results exist.	System creates an HTML report using jinja2. → System includes metrics and error details→ System displays the report in the browser.	User can view the structured report.

Handle Syntax Error	System	Uploaded code has syntax issues.	System detects the error using try–except→ System adds error info to the report→ System displays message: "Syntax error detected."	Report includes syntax error details.
Download the report	User	The HTML report has been generated and displayed.	User clicks the "Download Report" button→ System prepares the report for download→ System saves it as a local .html file on the user's device→ System confirms download completion	User receives a saved copy of the report locally.

Table 1 use case

4. External Interface Requirements

4.1 User Interfaces

4.1.1 Upload Page

- A button that allows the user to upload a Python (.py) file.
- A field displaying the file name after selection.
- An "Analyze Code" button to start the analysis process.
- An error message if the uploaded file is not a valid Python file.

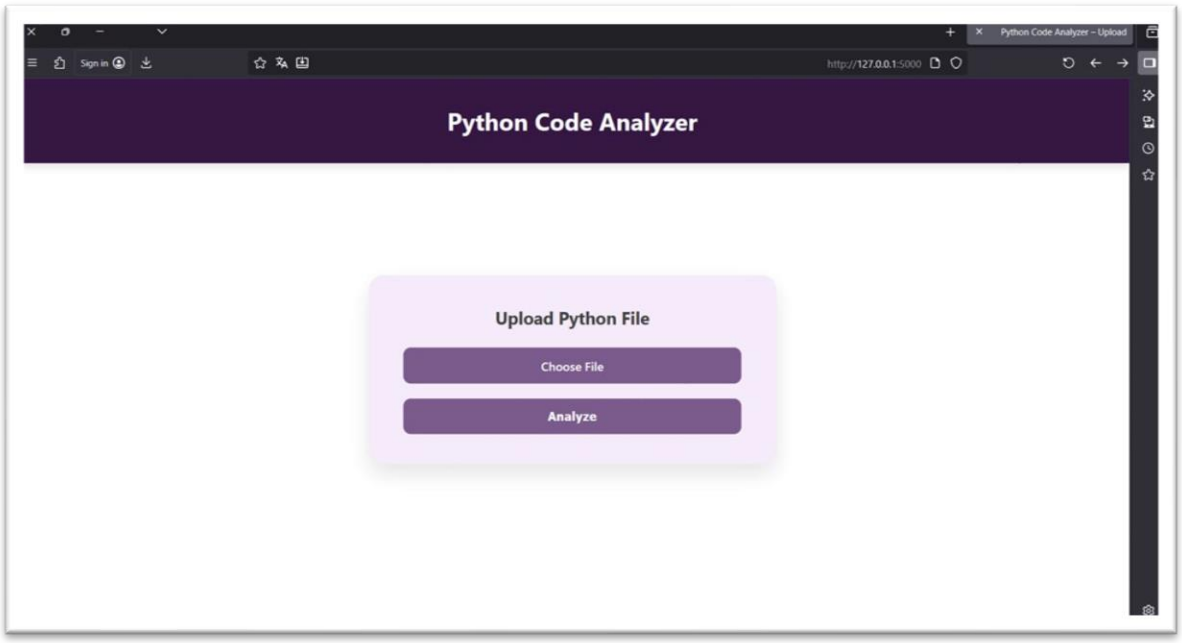

Figure 1 – User Interface of the Upload Page

4.1.2 Results Page

a) Summary metrics (total lines, number of classes, functions, imports)
b) Detailed structure for each class and function
c) Nesting analysis (how many nested blocks exist)
d) Syntax errors or warnings (if any)
e) Suggestions and recommendations (if any)
f) Buttons: Download Report / Return to Upload Page

4.2 Hardware Interfaces

The system does not require any special hardware devices.
• It runs on standard user devices such as laptops, PCs, or tablets.

4.3 Software Interfaces

The system interacts with the following software components:

4.3.1 Python Environment

a) Python 3.x interpreter used for parsing and analyzing the uploaded file.

4.3.2 Python Libraries

a) ast used for parsing and analyzing code structure.
b) inspect used to extract additional metadata.
c) jinja2 used to generate HTML reports.
d) pylint used for detecting simple style issues or unused imports.

e) radon used for complexity and maintainability metrics.

4.3.3 Web Server

a) Handles file upload requests from the user.
b) Sends the uploaded Python file to the analyzer module.
c) Returns the generated report to the browser.

4.3.4 Browser Interface

Must be compatible with modern browsers such as:

a) Google Chrome
b) Mozilla Firefox
c) Microsoft Edge

4.4 Communication Interfaces

a) The system communicates with the user through HTTP/HTTPS requests.
b) File upload is handled through standard multipart/form-data.
c) The server returns an HTML page containing the analysis report.
d) No external APIs or third-party communication services are used.

5. Non-Functional Requirements

5.1 Performance Requirements

- **Analysis Response Time:** Analyze Python code files up to 500 lines within **5 seconds** of clicking the "Analyze Code" button.
- **Report Generation:** Generate the structured HTML report using **Jinja2** within **2 seconds** after analysis completion.
- **File Size:** Successfully process Python files up to **1 MB** in size

5.2 Security Requirements

- **Data Handling:** Uploaded Python files are temporarily stored on the server during analysis and deleted immediately after the report is generated. No files are permanently saved

- **Access Control:** Restrict access to the GitHub source code repository to **team members only**.
- **No Sensitive Data:** Not require user authentication (login/password) as it does not handle sensitive data.

5.3 Reliability and Availability

- **Error Tolerance:** Handle **minor syntax errors** in uploaded Python files gracefully, ensuring the system continues to function without crashing.
- **Uptime:** the system should run smoothly without unexpected crashes or interruptions.
- **Data Integrity:** Ensure accurate and reliable **data transfer** between the backend and the web interface during the analysis and reporting process.

5.4 Maintainability

- **Code standards:** All code must follow (PEP8) to keep it clean and easy to manage and maintain
- **Modularity:** Be implemented using a **modular structure** to facilitate updating or extending analysis features in future versions.
- **Testing:** Include a comprehensive suite of **unit tests** (using `unittest` and `pytest`) to verify that all components work correctly after any modifications or feature extensions.

5.5 Portability

- **Platform:** Be accessible via a standard web interface on all common operating systems (Windows, macOS, Linux).
- **Browser Support:** Fully support current versions of **Google Chrome**, **Mozilla Firefox**, and **Microsoft Edge**.
- **Backend Environment:** Run on a web server supporting the **Python 3.x interpreter**.
-

6. Appendices

6.1 Glossary of Terms

Term	Definition
Python code analyzer	A web-based tool that reads and analyze Python code and generate reports
AST (Abstract Syntax Tree)	A tree representation of the structure of Python code, used by the analyzer to extract elements such as functions, classes, and imports.

Jinja2	A Python library used to generate HTML reports from templates.
Pylint	A Python tool that checks code for style issues and unused imports.
Radon	A python library that calculates code complexity
HTML Report	The structured output file that displays the results of the analysis
Syntax Error	An error in python that occurs when the structure or grammar is incorrect
Static analysis	The process of analyzing code without executing it
User interface (UI)	The part of the system that the user interacts with , such as buttons and pages
Web server	The backend component that processes uploaded files , runs the analyzer, and sends results back to the user
HTTP/HTTPS	Communication protocols used to send and receive data between the user's browser and the web server

Table2:Glossary of terms

6.2 Supporting Diagrams

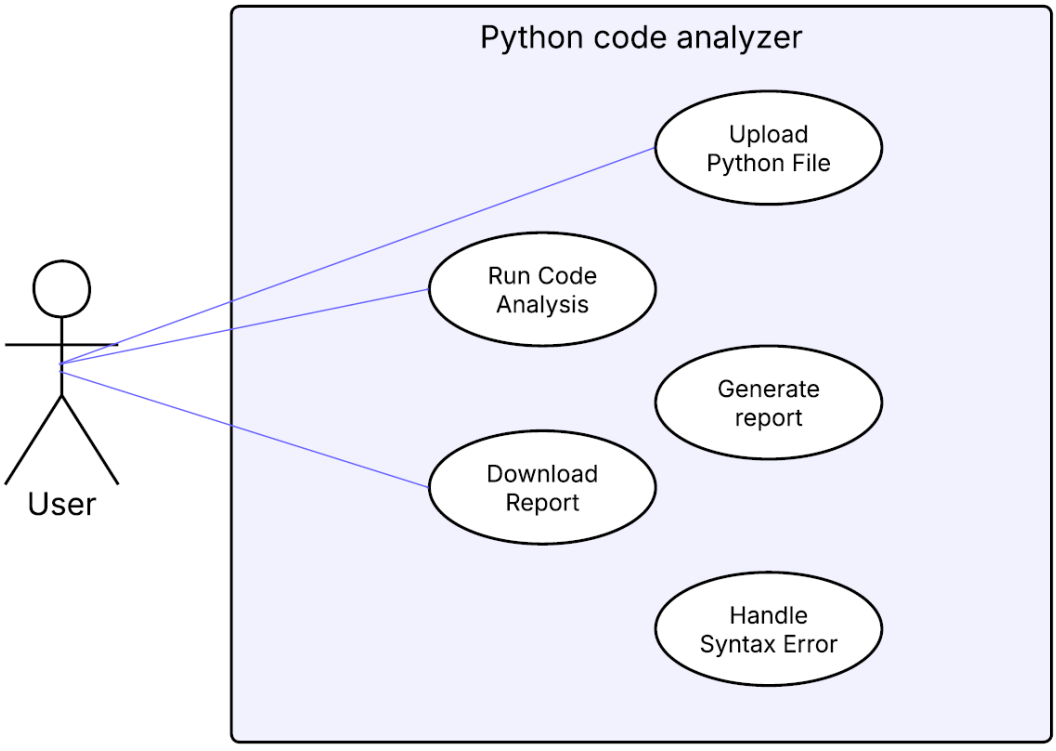

Figure 2 Use-Case Diagram for Python Code Analyzer

The diagram illustrates the main actor (User) and the system's primary functions.
The user uploads Python code, the system performs static analysis, and the user can view or download a report containing results and suggestions.

6.3 References

1. CSC305_5FA1_Group4_ProjectIdea

2. CSC305_5FA1_group4_projectProposal

3. CSC305_5FA1_Group4_SPMP

Evaluation

To be completed by the instructor or supervisor.

5. Project SRS Report (36 points / 6 marks) – Week 11

Section	Excellent (Full Points)	Good (75%)	Fair (50%)	Poor (25% or below)	Points
1. Introduction (5 pts)	Purpose, scope, audience, assumptions, and references clearly stated. Shows strong understanding of project goals.	Covers most parts, some lack detail or clarity.	Minimal explanation of purpose/scope, missing details.	Largely missing or irrelevant.	/5
2. Overall Description (6 pts)	Clear system perspective, major functions identified, user classes defined, operating environment and constraints explained.	Most elements present but some vague.	Only basic features described; missing user/environment/constraints.	Very minimal or missing.	/6

Section	Excellent (Full Points)	Good (75%)	Fair (50%)	Poor (25% or below)	Points
3. Specific Requirements (10 pts)	Well-structured functional requirements in "shall" format; complete use case specs with actors, flows, conditions; diagrams included.	Most requirements/use cases written but some unclear/incomplete.	Few functional requirements, weak use cases, little structure.	Minimal or absent requirements.	/10
4. External Interface Requirements (5 pts)	Detailed and clear: user interfaces, hardware, software, and communication interfaces described.	Covers most but with gaps (e.g., no hardware or comms detail).	Very brief, only one or two interface types.	Largely missing.	/5
5. Non-Functional Requirements (6 pts)	Comprehensive: performance, security, reliability, maintainability, portability all well-covered.	Covers most, but shallow or vague in detail.	Only 2–3 NFRs included, limited explanation.	Very weak or missing.	/6
6. Appendices (2 pts)	Glossary, diagrams, and references included where needed.	Some elements included but not all.	Minimal appendices (e.g., only glossary).	None provided.	/2
7. Overall Quality (2 pts)	Well-structured, consistent format, error-	Clear but with minor	Hard to follow, weak formatting/structure.	Poorly presented, incomplete.	/2

Section	Excellent (Full Points)	Good (75%)	Fair (50%)	Poor (25% or below)	Points
	free writing, professional presentation.	formatting/clarity issues.			

Note: If a student is listed as a project member at the beginning of the SRS but their contributions are **not reflected** (e.g., in requirements, use cases, or assigned responsibilities), that student will **not be awarded the same score** as active contributors.