**Abbottabad University of Science & Technology**

# SOFTWARE REQUIREMENTS SPECIFICATION

**(SRS DOCUMENT)**

# For

## < Key Word Search System>

***By***

**Urooj Mustafa** **14670**

Table of Contents

# 1. Introduction

## 1.1 Purpose

The purpose of the Keyword Search System is to help users quickly locate important terms or phrases within a file. It aims to automate the search process, eliminating the need for manual scanning of files. This system can be particularly useful for searching large text files, logs, reports, or even datasets to find relevant entries. By inputting a keyword, users can instantly find the context in which that word appears and retrieve the specific lines of interest. The system can be further extended to support more advanced search features, such as case-sensitive searches, regular expressions, and more.

## 1.2 Document Conventions

Headings: Main sections use numbers like 1, 2, 3, and subsections are labeled with decimals (e.g., 1.1, 1.2).

Formatting: Important terms, keywords, and file names are shown in bold or italics for easy recognition.

File Names and Commands: These are presented in italics or quotes to set them apart.

Error Messages: Messages from the system or errors are shown in a simple box format.

These conventions are used to make the report easier to read and follow

## 1.3 Project Scope

The **Keyword Search System** project focuses on providing a simple, efficient solution for searching specific keywords within text files. The system is designed to help users quickly locate occurrences of a keyword, along with their positions, in large text documents.

**In Scope**:

- **Basic Search Functionality**: Search for keywords within a single text file.
- **Case Sensitivity**: Option to perform case-sensitive or case-insensitive searches.
- **Error Handling**: Graceful handling of errors such as file not found or invalid input.
- **Simple Command-Line Interface (CLI)**: A text-based interface for easy user interaction.
- **Extensibility**: Potential future expansion to support additional features like batch file processing or advanced search options.

**Out of Scope**:

- **Graphical User Interface (GUI)**: No graphical interface will be provided in the initial version.
- **Multi-file Search**: Searching across multiple files or directories is not supported in the basic version.
- **Advanced File Formats**: Support for non-text files (e.g., PDF, DOCX) is not included in the current scope.

The project will focus on creating a reliable, fast, and easy-to-use system within the defined scope. Future releases may expand the functionality based on user needs and feedback.

### 1.3.1 Scope Definition

The scope of the Keyword Search System project is to develop a simple Python application that enables users to search for a specific keyword within a text file. The system will read the content of a given text file, search for the keyword, and display the lines where the keyword appears.

**Key features of the project include:**

Text File Processing: The system will handle reading and processing text files provided by the user.

Case-Insensitive Keyword Search: The user can input a keyword, and the system will search for it without being affected by letter case (e.g., searching for "python" will also match "Python").

Display of Search Results: The system will display the lines from the file where the keyword is found, along with the line numbers.

The project will not include more complex features like searching across multiple files, handling large datasets, or advanced search algorithms such as regular expressions. The focus is on creating a simple, easy-to-use tool for basic keyword searches in text files.

### 1.3.2 Core Features

### 1.3.3 Subsequent Releases

Future releases of the Keyword Search System can include several enhancements and additional features to improve functionality and user experience. Some of the potential upgrades include:

1. **Support for Multiple File Types**: Currently, the system works with basic text files. Future versions could support additional file formats such as PDFs, Word documents, or even HTML files.
2. **Advanced Search Options**: Future releases can include options for case-sensitive searches, regular expression searches, or the ability to search for multiple keywords simultaneously.
3. **Search Results Filtering**: Users could filter search results by specifying conditions such as the number of occurrences or proximity of the keyword to other words in the text.

4. **Graphical User Interface (GUI)**: A GUI could be added to make the system more user-friendly, allowing users to select files, input keywords, and view results in a more interactive way.
5. **Search History**: The system could keep track of previously searched keywords and files, allowing users to quickly access past searches.
6. **Performance Enhancements**: Optimizing the system for handling larger files or processing more complex searches efficiently.

These enhancements would make the Keyword Search System more versatile, powerful, and suitable for a wider range of use cases.

### 1.3.4 Alignment with User and Business Goals

The Keyword Search System is designed to align closely with both user and business objectives by addressing the need for efficient and reliable search capabilities in text files. Here's how the project aligns with these goals:

**User Goals:**

Ease of Use: The system provides a simple and intuitive interface where users can quickly search for keywords in text files, making it highly user-friendly and accessible even for those with minimal technical expertise.

Efficiency: The tool saves time by automating the keyword search process, allowing users to quickly locate specific information without manually scanning large documents.

Customization: Users can specify any keyword to search for and instantly view results, enhancing flexibility and ensuring the system meets a variety of personal or professional needs.

**Business Goals**:

Cost-Effective Solution: The Keyword Search System is an affordable solution that can be integrated into various business processes such as document management, content review, and data analysis without requiring significant resources or software investment.

Productivity Improvement: By automating the search process, businesses can streamline workflows, especially in environments where handling large volumes of text data (e.g., reports, logs, or manuals) is common. This leads to improved productivity and faster decision-making.

Scalability: The system can be easily adapted for more complex search scenarios, such as handling larger datasets, supporting additional file formats, or integrating with other business tools, making it a scalable solution as business needs evolve.

Overall, the Keyword Search System is well-aligned with both user expectations for ease of use and efficiency, as well as business objectives for improving productivity and offering a cost-effective, scalable solution

### 1.4 References

https://docs.python.org/3/

https://stackoverflow.com/questions/

https://stackoverflow.com/questions/

## 2. over all Description

The Keyword Search System is a simple Python-based application designed to allow users to search for a specified keyword within a text file. It offers a straightforward and efficient way for users to locate relevant information by searching through large text files, whether for research, analysis, or other purposes. The system reads the file, searches for the user-defined keyword, and displays the specific lines where the keyword appears.

**The project is divided into the following key components:**

File Handling: The system is built to read text files, allowing users to input the name of the file they want to search within. It processes the content of the file line by line to identify occurrences of the keyword.

Keyword Search: Once the file is loaded, the system searches for the given keyword. The search is case-insensitive, meaning it will match the keyword regardless of whether it is written in uppercase or lowercase letters.

User Interaction: The user interacts with the system by providing two key inputs: the name of the file and the keyword they wish to search for. The system then outputs all lines in the file where the keyword is found, along with the line numbers.

Error Handling: The system includes basic error handling to deal with scenarios such as the file not being found or the user providing an invalid file name or keyword.

The overall goal of this project is to provide a user-friendly, efficient, and easily extendable tool for text-based searches. It is intended for personal use or small-scale applications where users need to extract specific information from text files without the complexity of more advanced search tools.

Future enhancements may include adding support for multiple file formats, improving performance for larger files, and adding more advanced search features like regular expressions.

### 2.1 Product Perspective

The Keyword Search System is a standalone Python-based application that functions as a simple, effective tool for searching specific keywords within a text file. It can be integrated into any environment that requires quick keyword searches within documents or logs, making it suitable for both individual users and businesses dealing with large amounts of text data.

In terms of product perspective, the Keyword Search System is:

1. **Standalone Tool**:
    - The system is designed as a self-contained application that requires no additional software or complex configurations. Users only need Python installed on their system to run the application.
    - The system operates in a command-line interface (CLI) where users provide inputs (file name and keyword) and receive the results directly in the console.
2. **Simplicity and Ease of Use**:
    - The product is simple, with minimal setup or learning required. Users can quickly input a file name and keyword to perform a search.
    - It's designed to cater to users with limited technical knowledge, making it a practical tool for non-programmers or anyone needing quick access to search functionality.
3. **Text-Based Application**:
    - Currently, the system supports plain text files, where it scans the text and returns the lines containing the keyword. This approach makes it ideal for small to medium-sized files, such as logs, reports, and general text-based documents.
    - The system is not intended to replace more sophisticated search tools but serves as a lightweight and efficient solution for simple use cases.
4. **Extensibility**:
    - The system's design allows for future improvements. For instance, adding support for multiple file types (e.g., PDFs, Word documents) or integrating more complex search features (e.g., regex-based search, case-sensitive search) is possible as the product evolves.
    - It can also be expanded to support graphical user interfaces (GUIs), making it more accessible to a broader audience.

Overall, the Keyword Search System provides a simple, effective solution for keyword searching in text files, with potential for further enhancements as user needs grow. It fits well into environments where quick and reliable text searches are required without the complexity of larger software tools.

### 2.1.1 Product Context

The Keyword Search System exists within the broader context of document management, data processing, and information retrieval systems. It is designed to serve as a simple, efficient tool for searching keywords within text files, which is a common task in many fields such as research, content management, software development, and data analysis.

The system interacts with the following components and environments:

1. **User Interaction**:
    - The primary users of this system are individuals or businesses dealing with text-based data. These could include researchers, content writers, system

administrators, or analysts who need to quickly find specific information within documents.
    - Users interact with the system through a simple command-line interface (CLI), where they provide inputs (file name and keyword) and receive search results.
2. **Text Files**:
    - The system primarily deals with text files, which are the input data source. It reads these files, processes the content, and searches for a user-defined keyword. The system assumes the files are in a readable text format (e.g., `.txt` files).
    - Future versions may extend this to support other file types such as PDFs, Word documents, or HTML files, expanding the scope of the product.
3. **Python Programming Language**:
    - The system is built using Python, a versatile programming language known for its simplicity and power in file handling, string manipulation, and data processing. Python's built-in libraries (e.g., `open()`, `read()`, and `input()`) facilitate the development of this project.
    - Python provides a robust environment for rapid development, and its portability ensures the system can be run on various platforms such as Windows, macOS, and Linux.
4. **Standalone Application**:
    - The Keyword Search System operates as a standalone application, meaning users can run it directly without needing complex installations or dependencies. All users need is Python installed on their machine.
    - This makes the system accessible to a wide range of users, from individuals working on personal projects to businesses that need quick keyword search functionality within their workflow.

In the future, the Keyword Search System could be integrated into larger software ecosystems, such as content management systems (CMS), document management software, or enterprise resource planning (ERP) systems, to provide seamless keyword searching capabilities across various types of documents and data formats. This would expand the product's context to a more comprehensive information retrieval tool suitable for more complex business needs.

### 2.1.2 Product Origin

The Keyword Search System originated from the need to create a simple, efficient tool for searching specific keywords within text files. The motivation behind developing this system was to address the common challenge of quickly locating relevant information in large documents or sets of data without the need for manual inspection. This problem is especially prevalent in fields such as research, content creation, software development, and system administration, where working with text-based files is routine.

The idea for the project was inspired by:

1. **Basic File Handling**:
    - The system was conceived as a way to demonstrate core file handling techniques in Python, specifically how to read and manipulate text files. As part of learning

and applying Python skills, it made sense to create a practical application that could be useful in real-world scenarios.

2. **Text Search Functionality**:
    - Many existing software tools offer advanced search capabilities, but for individuals working with small to medium-sized text files, a lightweight and simple tool is often sufficient. The system was designed to fulfill this need, allowing users to perform fast keyword searches with minimal setup.
3. **Educational Purpose**:
    - The project was also created as a learning exercise to showcase the power of Python for text processing and to demonstrate practical problem-solving through programming. It serves as an example of how even a simple script can provide valuable functionality in day-to-day tasks.
4. **User Demand for Simplicity**:
    - There is often a demand for user-friendly tools that do not require complicated installations or configurations. The Keyword Search System was designed to be a straightforward application that can be used by anyone with basic computer skills, making it widely accessible.

The Keyword Search System started as a small project and has the potential to grow into a more comprehensive tool with added features, making it both a useful utility and a stepping stone for more advanced software development in the future.

### 2.1.4 Product Ecosystem

The Keyword Search System exists within a broader ecosystem of tools, software, and technologies that support file handling, text processing, and information retrieval. Its interactions with other components form a basic yet effective ecosystem aimed at providing a seamless user experience for searching text-based data. Here's how the product fits within this ecosystem:

1. **Operating Systems**:
    - The Keyword Search System is designed to work on various operating systems (e.g., Windows, macOS, Linux) as long as Python is installed. This makes it a cross-platform tool that can be integrated into diverse environments where users may need to search through text files.
2. **Python Programming Language**:
    - Python is the core technology that powers the Keyword Search System. It handles file reading, keyword search operations, and user interactions. Python's simplicity and ease of use ensure that the system is lightweight and easy to extend in the future.
3. **Text-Based Files**:
    - The system interacts primarily with text-based files (e.g., `.txt`). It is designed to open, read, and search these files line by line. Text files are ubiquitous and easy to work with, making them an ideal format for the Keyword Search System's core functionality.
4. **Command-Line Interface (CLI)**:

- The Keyword Search System operates through a command-line interface, where users input commands such as the file name and keyword for searching. This aligns it with other CLI-based utilities, making it suitable for integration into automated workflows, batch processing, or remote server environments.

5. **Other Text Processing Tools**:
   - The Keyword Search System is part of a larger ecosystem of text processing tools and libraries that include more advanced software like text editors (e.g., Notepad++, Sublime Text), content management systems (CMS), and document search tools (e.g., Elasticsearch, Grep). While these tools offer more advanced search features, the Keyword Search System focuses on simplicity and ease of use for basic searches.
6. **Future Integrations**:
   - In the future, the system could be integrated into larger business ecosystems, such as content management platforms, data analytics tools, or document management systems. This could enable more complex, multi-format search capabilities and allow the system to interact with databases or cloud-based storage solutions.
7. **User Environment**:
   - The user ecosystem includes individuals, researchers, writers, developers, and businesses dealing with text-based data. These users could deploy the Keyword Search System in various workflows, ranging from personal use (e.g., searching documents on a local machine) to business applications (e.g., searching logs, reports, and text files).

The Keyword Search System's ecosystem is designed to be simple yet adaptable, providing users with the core functionality needed to search through text files while allowing for future growth and integration with more complex systems and environments.

## 2.2 User Classes and Characteristics

The Keyword Search System is designed to be simple and accessible to a wide range of users. Below are the primary user classes for this system, each with distinct characteristics and needs:

1. **Individual Users (Basic Users)**
   - **Characteristics**:
     - These are everyday users, such as students, researchers, or anyone working with text-based files (e.g., notes, essays, or reports).
     - They may have limited technical knowledge and are primarily looking for a simple and efficient way to search through their documents.
     - They prefer tools that are easy to use, with minimal setup or configuration.
   - **Needs**:
     - A simple, straightforward tool that allows them to search for a specific keyword in a text file without having to open and read the entire document.
     - User-friendly interface (CLI-based, minimal commands).
2. **Business Users (Professional Users)**

- **Characteristics**:
  - These users may be working in fields such as data analysis, content management, IT support, or administrative roles, where managing and searching through logs, reports, and text files is part of their daily tasks.
  - They typically have a higher level of technical understanding and may need to integrate the system into larger workflows.
- **Needs**:
  - A fast, efficient way to locate keywords across text files quickly.
  - Ability to search large files or multiple documents with ease.
  - They may benefit from additional features in the future, such as advanced search filters or batch processing capabilities.

3. **Software Developers (Technical Users)**
   - **Characteristics**:
     - Developers or programmers who may need to search through code, logs, or other text-based data files.
     - They have a good understanding of programming and may look for ways to automate or integrate the Keyword Search System into their development environment.
   - **Needs**:
     - Fast keyword search across potentially large codebases or log files.
     - Ability to customize the system or integrate it with other tools in their workflow, such as version control systems or automated testing tools.
     - Developers may also be interested in extending the system by adding new features or functionalities.
4. **System Administrators (Technical/Support Users)**
   - **Characteristics**:
     - IT professionals or system administrators who manage large server environments or handle logs, configuration files, and other system-related text files.
     - These users have a high level of technical expertise and may need a more efficient way to monitor, troubleshoot, or search through system logs and configuration files.
   - **Needs**:
     - A tool that can quickly search through large logs, configuration files, or system outputs for errors or specific patterns.
     - Integration with other system tools or automation to improve their workflow.
     - They might require the ability to scale the system to handle large datasets or perform batch searches on multiple files.
5. **Educational Institutions (Learning/Teaching Users)**
   - **Characteristics**:
     - Teachers, instructors, or students who are learning about programming, data processing, or text manipulation.
     - They may use the Keyword Search System as an example for educational purposes, learning how to read files and search for specific terms in Python.

- **Needs**:
  - A simple, well-documented tool that can be used to demonstrate basic file handling and string manipulation in Python.
  - Clear instructions and easy-to-understand code that can serve as a learning resource.

These user classes reflect a broad spectrum of individuals who may benefit from the Keyword Search System. While the system is primarily designed for simplicity and ease of use, it also has the potential to meet the needs of more technically advanced users, offering flexibility for different contexts and use cases.

### 2.2.1 Tech Enthusiasts

**Characteristics**:

- Users passionate about technology, programming, and exploring new tools.
- Have a good understanding of computing and enjoy experimenting.

**Needs**:

- **Exploration**: Study and modify the system's code.
- **Customization**: Add features like advanced search or support for more file types.
- **Integration**: Use the system with other tools or in larger projects.
- **Learning**: Appreciate clear documentation and opportunities to expand knowledge.

### 2.2.2 Casual Shoppers

**Characteristics**:

- Non-technical users looking for a simple solution.
- Primarily use the system for quick, basic searches without much setup.

**Needs**:

- **Simplicity**: Easy-to-use, straightforward tool for keyword searches.
- **Efficiency**: Fast results with minimal effort.
- **Accessibility**: Works well without complex configurations.

### 2.2.3. Favored User Class

**Characteristics**:

- Users who benefit the most from the Keyword Search System.
- Typically consist of researchers, content managers, and students who need to search through text files regularly.

**Needs**:

- **Efficiency**: Quick keyword search in large text files.
- **Accuracy**: Reliable results to locate relevant information.
- **Ease of Use**: Simple interface for minimal interaction.

### 2.2.4. Alignment with User Needs

The Keyword Search System is designed to meet the following user needs:

- **Efficiency**: Provides quick, accurate search results for users dealing with large text files.
- **Simplicity**: Easy to use for non-technical users, with a minimal command-line interface.
- **Flexibility**: Allows tech enthusiasts and developers to customize or integrate the system into larger workflows.
- **Reliability**: Ensures consistent performance for users across different environments and use cases.

## 2.3 Operating Environment

The Keyword Search System operates in the following environments:

- **Hardware**: Can run on any standard computer (laptop or desktop) with sufficient memory and processing power to handle text files, even large ones.
- **Software**: Requires Python (version 3.x) to run. It can be used on any operating system with Python installed, including Windows, macOS, and Linux.
- **Dependencies**: No additional libraries or complex software are required beyond Python itself. The system uses built-in Python functions for file handling and text search.
- **User Interface**: Operates through a command-line interface (CLI), with user inputs for the file and keyword to search.

### 2.3.1 Hardware Platform

The Keyword Search System can run on standard desktop or laptop computers with the following hardware specifications:

- **Processor**: Minimum of a 1 GHz processor (recommended for better performance).
- **Memory**: At least 1 GB of RAM (recommended 4 GB or more for larger files).
- **Storage**: Requires minimal disk space (a few MBs) for the Python runtime and text files.
- **Peripherals**: Keyboard and screen for user input and output in the command-line interface.

### 2.3.2 Operating Systems and Versions

The Keyword Search System is compatible with the following operating systems:

- **Windows**: Windows 7 or later, with Python 3.x installed.
- **macOS**: macOS 10.9 (Mavericks) or later, with Python 3.x installed.
- **Linux**: Any distribution supporting Python 3.x (e.g., Ubuntu, Fedora, Debian).

No additional software or setup is required beyond Python installation. The system is platform-independent as long as Python is available.

## 3. System Features

- **Keyword Search**:
  - Users can search for a specific keyword within a text file. The system returns all occurrences of the keyword along with their positions in the file.
- **File Handling**:
  - The system allows users to load and read text files from local storage. It handles both small and large text files efficiently.
- **Case Sensitivity**:
  - The system allows for case-sensitive and case-insensitive searches, depending on the user's preference.
- **Multiple File Support (Future Feature)**:
  - Potential to extend the system to handle multiple files simultaneously for batch searches.
- **Error Handling**:
  - If an invalid file or keyword is provided, the system gracefully handles errors and notifies the user.
- **Simple User Interface**:
  - Command-line interface (CLI) for easy interaction, with minimal commands needed to perform searches.
- **Extensibility**:

- The system is designed to be extended, allowing developers to add advanced features like regular expression search, support for different file formats, or integration with other tools.

4. Data Requirements

- **Input Data**:
  - **Text Files**: The system requires text files (.txt) as input, which must be accessible from the user's local storage. The file must contain readable text in a standard encoding (e.g., UTF-8).
- **Search Data**:
  - **Keywords**: The system needs a keyword or phrase provided by the user to search within the text files. This data is case-sensitive unless specified otherwise.
- **Output Data**:
  - **Search Results**: The system outputs the keyword search results, including the positions where the keyword is found in the text, along with line numbers or indices.
- **Storage Requirements**:
  - The system does not require a large amount of storage but needs sufficient space to store the input files and Python environment. Text files should be stored locally on the user’s machine or in an accessible directory.
- **Future Data Needs**:
  - As the system expands (e.g., for handling multiple file formats), it may require additional libraries or access to databases or cloud storage for managing large datasets.

4.1 Logical data model

The logical data model for the Keyword Search System is simple and focuses on managing text files and keywords for search operations. Below is a conceptual representation of the system's data structure:

1. **Text Files (Input Data)**
   - **Attributes**:
     - `file_name`: The name of the text file.
     - `file_content`: The content of the text file, stored as lines of text (list of strings).
   - **Description**:
     - Text files are the primary data source for the search. The file's content is read line by line to search for keywords.
2. **Keywords (Search Data)**
   - **Attributes**:
     - `keyword`: The string or word the user wants to search for.
     - `case_sensitive`: Boolean indicating whether the search is case-sensitive.
   - **Description**:
     - The keyword data is used to search through the text file content. The search can either be case-sensitive or case-insensitive based on user input.
3. **Search Results (Output Data)**
   - **Attributes**:
     - `matches`: A list of match positions where the keyword is found.
     - `line_number`: The line number where the keyword occurs.
     - `position_in_line`: The position of the keyword in the line.
   - **Description**:
     - The system outputs the results, showing where the keyword was found in the file along with the corresponding line number and position within that line.
4. **Error Messages (Error Handling)**
   - **Attributes**:
     - `error_type`: The type of error (e.g., file not found, invalid input).
     - `message`: A brief description of the error.
   - **Description**:
     - If the user provides invalid data (like a non-existent file), the system returns an appropriate error message to guide the user.

This logical data model keeps the system straightforward and efficient, focusing on text data, keywords, and search results. As the system evolves, this model can be expanded to support additional data sources or more complex features.

# 5. External interface requirements

**1 User Interface**:

- **Command-Line Interface (CLI)**:
  - Users will interact with the system through a text-based CLI.
  - The interface will require simple input commands such as the file name and the keyword to search.
  - Output will be displayed directly in the terminal, showing search results or error messages.

**2 Hardware Interfaces**:

- **Storage Access**:
  - The system requires access to local file storage to read the text files. It will work on any computer with sufficient disk space to store text files and the Python environment.
  - No specialized hardware is required, but the system should have enough RAM to handle larger text files.

**3 Software Interfaces**:

- **Python Environment**:
  - The system depends on Python 3.x to execute. It should be installed on the user's machine to run the Keyword Search System.
- **File System**:
  - The system relies on the operating system’s file system to access, read, and search through text files.

**4 Communication Interfaces**:

- **File Input/Output**:
  - The system accepts input files via local file paths, and results are output in the terminal.
- **Error Handling**:
  - In case of errors (e.g., missing file, invalid input), the system will notify the user with appropriate messages through the terminal.

**5 External Systems**:

- The Keyword Search System operates as a standalone tool and does not require integration with external systems, databases, or APIs. However, it can be extended to interact with more complex systems in future versions.

This ensures the system is simple and efficient, with minimal dependencies on external systems.

## 5.2 Software Interfaces

The Keyword Search System interacts with the following software components:

1. **Python 3.x**:
    - The system is built using Python 3.x, so it requires a Python interpreter installed on the user's machine to execute. It utilizes Python's built-in libraries, including `os` for file handling and `re` for advanced search (if added in the future).
    - **Requirements**: Python 3.6 or later.
2. **File System**:
    - The system relies on the operating system's file system to read text files. It accesses files using standard file I/O operations (open, read, close).
    - Supported on all platforms with Python (Windows, macOS, Linux).
3. **External Libraries (Optional)**:
    - **None**: The basic version of the system does not require any external libraries. However, in future releases, libraries like `regex` for advanced search or `tkinter` for GUI (if implemented) might be used.
4. **Command-Line Interface (CLI)**:
    - The system communicates with the user via the CLI. It uses Python's `input()` function to receive data and `print()` to display results or errors.

These software interfaces ensure compatibility with all systems that support Python.

### 5.2.4 Non-Functional Requirements

1. **Performance**:
    - The system should handle files of up to 100 MB efficiently, with minimal lag during keyword searches. For larger files, performance may vary depending on hardware.
2. **Reliability**:
    - The system should consistently return correct search results and handle errors gracefully, ensuring no crashes or unexpected behavior.
3. **Usability**:
    - The system should be simple to use, requiring only basic knowledge of how to run a command in a terminal. The interface should be clear and prompt the user for necessary inputs.
4. **Scalability**:
    - The system should be easily extendable to handle additional features like searching across multiple files or supporting other file formats (e.g., PDF, DOCX).
5. **Security**:
    - The system should ensure that no sensitive data (such as file paths or search keywords) is exposed during the search process. Since the system operates

locally, there are no direct security risks, but safe handling of files should be ensured.

6. **Maintainability**:
    - The code should be modular and well-documented to allow for easy updates, bug fixes, and feature additions. Future enhancements should be simple to integrate.
7. **Portability**:
    - The system should be cross-platform, running on any operating system (Windows, macOS, Linux) as long as Python is installed. No platform-specific dependencies are required.
8. **Availability**:
    - The system should be available for use whenever needed, with no server or network dependencies since it operates offline.

These non-functional requirements focus on ensuring that the system is efficient, easy to use, and adaptable for future needs.

# 6. Quality Attributes

6.1 **Performance**:

- The system should process keyword searches quickly, even for moderately large text files (up to 100 MB). The performance can degrade with very large files, but it should remain responsive for typical use cases.

6.2 **Usability**:

- The system should be simple and intuitive, requiring minimal user input (file path and keyword). The command-line interface should provide clear prompts and outputs, ensuring a smooth user experience.

6.3 **Reliability**:

- The system should consistently produce correct search results without crashes. Error handling should be implemented to gracefully manage issues such as file not found or invalid input.

6.4 **Scalability**:

- The system should be designed to accommodate future feature additions, such as searching through multiple files or supporting additional file types, without significant changes to the core functionality.

6.5 **Maintainability**:

- The code should be modular, well-organized, and documented, making it easy for developers to update or add new features. Clear code structure helps in troubleshooting and future improvements.

6.6 **Portability**:

- The system should be cross-platform, capable of running on Windows, macOS, and Linux without requiring platform-specific adjustments.

6.7 **Security**:

- The system should ensure that user inputs, such as file paths and keywords, are handled securely to avoid potential security risks. Since the system operates locally, it minimizes exposure to external threats.

6.8 **Availability**:

- The system should be available for use anytime as it operates offline, with no dependencies on external services or network connections.