The design and architecture of the software will be based on the principles of throwaway prototyping, allowing for rapid iteration and development of the application. The testing and validation process will include both functional and non-functional testing to ensure that the software meets the ISO 25010 standards for Functional suitability, Performance efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability.

Development Process

In this, the researchers will use the throw-away prototyping methodology. This involves of data gathering, analysis, design, implementation, integration and maintenance.

It will focus on creating a functional prototype as quickly as possible to test the viability of design ideas to create a solution and served its purposed to meet more permanent solution to developed.

Throw-away prototyping will be start at data gathering. In this, the researchers need to gather the requirements for the document similarity analysis system. This includes determining the features that are required, such as the ability to compare image and PDF files, and algorithms that will be used to compare documents. These requirements may include the types of documents that will be analyzed, the metrics used to measure similarity, and the expected output format.

After that, next is the analysis. This involves analyzing the requirements for the document similarity analysis system, which includes identifying the key features and functionalities that the system should have.

This analysis helps to ensure that the system will meet the needs of its users.

The design would refer to the overall architecture of the system, the algorithms that will be used for document similarity analysis, and the user interface design. It will allow user to analyze the study and attempt before implementing to help users meet the needs.

In implementation, it develops a quick prototype of the system using Python. This should include the basic functionality of the system, such as uploading and processing PDF and image files, and displaying the similarity score. It tests the prototype to ensure that it meets the requirements and functions as intended and identify any issues or areas for improvement in the prototype.

Once the prototype of the document similarity analysis system using Python for image and PDF files has been developed, it needs to be integrated with other systems and technologies to ensure that it functions correctly.

This involves performing a series of tests to verify that data can be transferred between the systems, that the system behaves as expected, and any potential issues or bugs are identified and resolved.

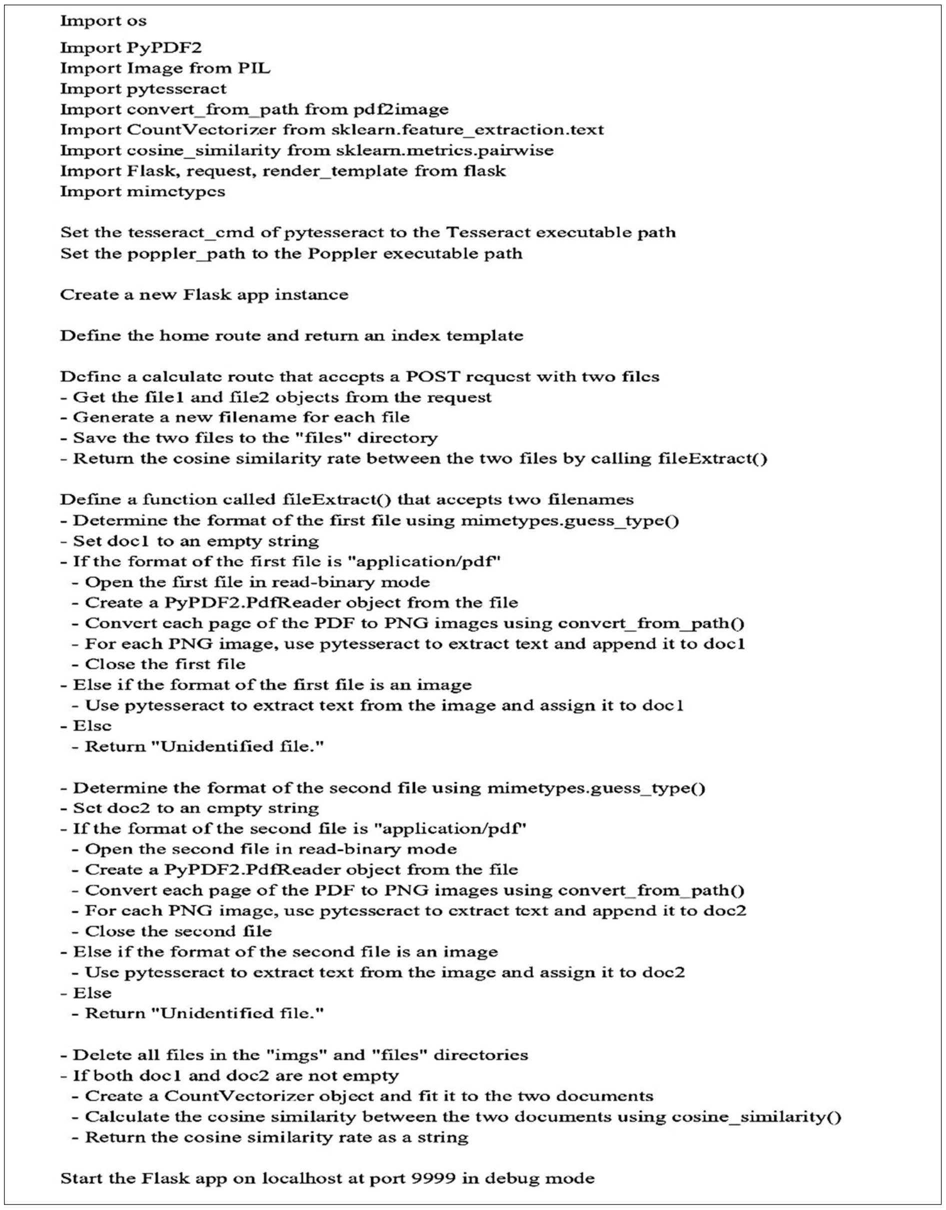
By thoroughly testing the integration, it is possible to ensure that the document similarity analysis system functions correctly within the overall system architecture and meets the requirements of the users.

The maintenance, the researchers continue to maintain and improve it based on user feedback and changing requirements.

Using the throw-away prototyping methodology it quickly tests new features or changes before implementing them in system.

Requirement Analysis

Pseudocode



```
Import os
Import PyPDF2
Import Image from PIL
Import pytesseract
Import convert_from_path from pdf2image
Import CountVectorizer from sklearn.feature_extraction.text
Import cosine_similarity from sklearn.metrics.pairwise
Import Flask, request, render_template from flask
Import mimetypes

Set the tesseract_cmd of pytesseract to the Tesseract executable path
Set the poppler_path to the Poppler executable path

Create a new Flask app instance

Define the home route and return an index template

Define a calculate route that accepts a POST request with two files
- Get the file1 and file2 objects from the request
- Generate a new filename for each file
- Save the two files to the "files" directory
- Return the cosine similarity rate between the two files by calling fileExtract()

Define a function called fileExtract() that accepts two filenames
- Determine the format of the first file using mimetypes.guess_type()
- Set doc1 to an empty string
- If the format of the first file is "application/pdf"
  - Open the first file in read-binary mode
  - Create a PyPDF2.PdfReader object from the file
  - Convert each page of the PDF to PNG images using convert_from_path()
  - For each PNG image, use pytesseract to extract text and append it to doc1
  - Close the first file
- Else if the format of the first file is an image
  - Use pytesseract to extract text from the image and assign it to doc1
- Else
  - Return "Unidentified file."

- Determine the format of the second file using mimetypes.guess_type()
- Set doc2 to an empty string
- If the format of the second file is "application/pdf"
  - Open the second file in read-binary mode
  - Create a PyPDF2.PdfReader object from the file
  - Convert each page of the PDF to PNG images using convert_from_path()
  - For each PNG image, use pytesseract to extract text and append it to doc2
  - Close the second file
- Else if the format of the second file is an image
  - Use pytesseract to extract text from the image and assign it to doc2
- Else
  - Return "Unidentified file."

- Delete all files in the "imgs" and "files" directories
- If both doc1 and doc2 are not empty
  - Create a CountVectorizer object and fit it to the two documents
  - Calculate the cosine similarity between the two documents using cosine_similarity()
  - Return the cosine similarity rate as a string

Start the Flask app on localhost at port 9999 in debug mode
```