Joshua Lee, Farooq Khan, Rudra Patel, Thomas McNamara, Vignesh Yampally

Requirements Workshop

1. Provide an example of five hypothetical non-functional requirements for this system. Be sure to include the specific type of requirement discussed in class, with each requirement coming from a unique category.

Usability – Comments must be differentiated from the rest of the code.
Reliability – Will be efficient and accurate 90% of the time.
Performance – Documentation will be generated within 5 seconds.
Supportability – Our system will be updated with new languages that are created.
Implementation/Constraints – Written with Python.

2. Provide an example of five hypothetical functional requirements for this system.

1.) Comments will be tailored to the user's functions.
2.) The comments must be at least one sentence long.
3.) Will not overwrite existing documentation.
4.) Be able to handle multiple functions at one time.
5.) Will ignore variables.

3. Think of a specific task required to complete each of the functional requirements and non-functional requirements mentioned above (10 total). Estimate the amount of effort needed to complete this task using function points (i.e., using the values here). Briefly explain your answer.

1.) Code must be written to change the format of the comments. (4 function points)
2.) Testing must be done to ensure that this ratio is met. (2 fp)
3.) Testing must be done to ensure that the code runs in this timeframe. (2 fp)
4.) The code must be revisited when new languages become popular. (10 fp)
5.) Python must be downloaded on the developer's computer. (1 fp)
6.) Code must be written to analyze the body of the function. (7 fp)
7.) Tests must be run to ensure that all generated comments are at least one sentence long. (2 fp)
8.) Tests must be run to ensure that the code can handle files of any length. (2 fp)
9.) Tests must be run to ensure that the code can generate comments for multiple functions. (2 fp)
10.) Code must be written that can differentiate between a comment and a function. (4fp)

4. Write three user stories from the perspective of at least two different actors. Provide the acceptance criteria for these stories.

- Johnny is on a software team of 5 people, where he is developing code that will be used by the whole team. He needs to do it in a timely manner, so he utilizes our code documenter to write comments on his code for his team to understand what he is doing. The acceptance criteria are that the comments must be descriptive and tailored to every one of his functions.

- Caroline has written some code for a school project. The project is very long, and Caroline is unsure whether or not she has documented all of her functions. Instead of checking through each file individually (which would be very time consuming), she decides to use our tool to generate documentation where she may have forgotten. Acceptance Criteria: Comments must be generated for all functions while not overwriting functions that already have complete documentation.

- Bob needs to send his code over to a member on his team, but it is poorly documented. For the other person to understand his code he utilizes the code documentation tool to quickly generate comments before sending it over. The acceptance criteria would be that the person receiving the code is able to understand the documentation that was generated.

5. Provide two examples of risk that could potentially impact this project. Explain how you would mitigate these risks if you were implementing your project as a software system.

- There could be security risks with sensitive files that include user data. We could implement an authentication token for security.

- The tool could malfunction and not work as intended and corrupt original files. We will implement error handling by throwing exceptions and showing warning messages when the script fails.

6. Describe which process your team would use for requirements elicitation from clients or customers, and explain why.

Our team will use brainstorming and prototyping since those processes would work the best with our product. Gathering stakeholders and collecting ideas would be important to learn what stakeholders value, and prototyping would help us come up with a functional product.

Requirements Analysis

Requirements analysis is the process of understanding the requirements for a software application.
This deliverable will include 5:

- Fully dressed use cases
- Model (use case or sequence diagrams) for representing each use case

First Use Case: User wants comments on one function while coding

1. Preconditions:

User has one function implemented

2. Main Flow:

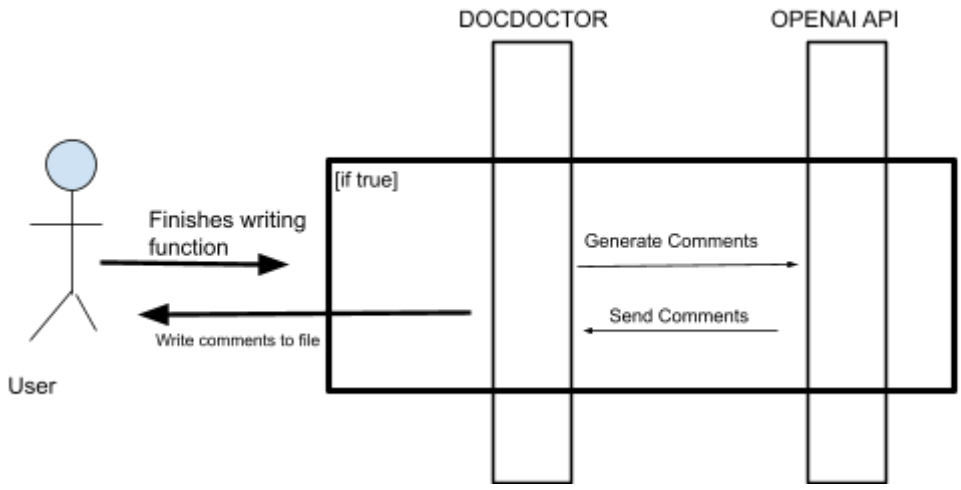
The User will finish typing up one function in their Python/Java file in realtime. Then, DocDoctor will see the function is completed [S1] and generate comments [S2]. Finally, comments would be attached [S3].

3. Subflows:

[S1] DocDoctor will recognize the end of a function implementation
[S2] DocDoctor will utilize OpenAI's API to generate comments
[S3] DocDoctor attaches the comments to the Python/Java file

4. Alternative Flows:

[E1] No function is available



Second Use Case: User wants comments on multiple functions while coding

1. Preconditions:

User has multiple functions implemented

2. Main Flow:

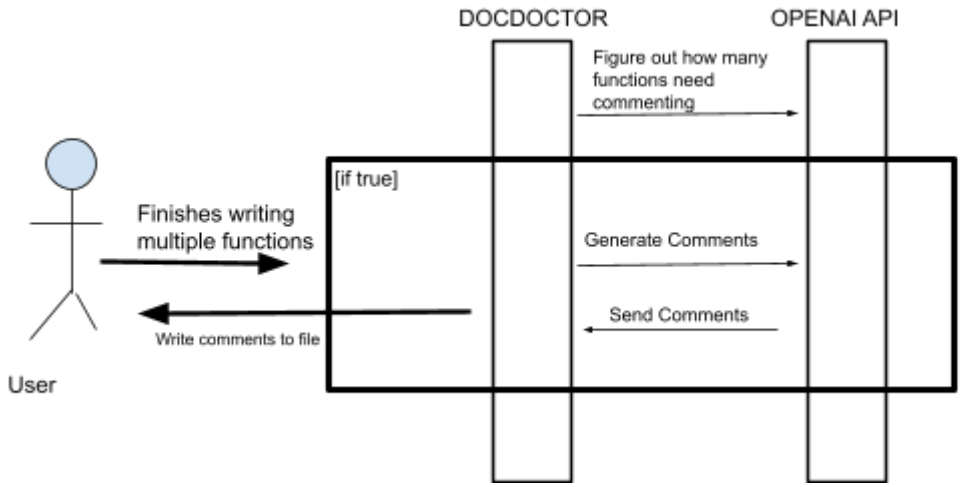
The User will finish writing multiple functions in their Python/Java file in realtime. Then, DocDoctor will see how many functions need comments to be generated [S1]. Then, it will talk to OpenAI's API and generate comments [S2]. Finally, comments would be attached [S3].

3. Subflows:

[S1] DocDoctor will figure out how many functions need comments
[S2] DocDoctor will utilize OpenAI's API to generate comments
[S3] DocDoctor attaches the comments to the Python/Java file

4. Alternative Flows:

[E1] No function is available



Third Use Case: Batch Processing for Existing Projects

1. Preconditions:

User has an existing project with multiple files, each containing several functions without documentation.

2. Main Flow:

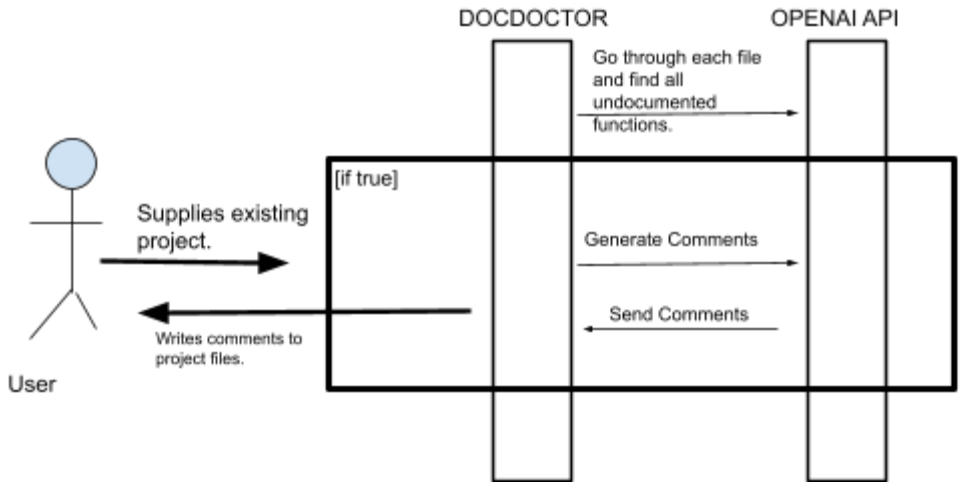
The User selects an entire project or specific directories for documentation [S1]. DocDoctor scans all files to identify functions without comments [S2], generates comments using OpenAI's API [S3], and attaches these comments appropriately [S4].

3. Subflows:

[S1] User selects the project/directory through the DocDoctor interface.
[S2] DocDoctor analyzes each file to detect undocumented functions.
[S3] DocDoctor utilizes OpenAI's API for each undocumented function found.
[S4] DocDoctor attaches generated comments above the corresponding functions.

4. Alternative Flows:

[E1] Selected project/directory does not contain any functions.
[E2] All functions are already well documented.



Fourth Use Case: Customizable Commenting Styles

1. Preconditions:

User preferences for comment styles are set in DocDoctor settings.

2. Main Flow:

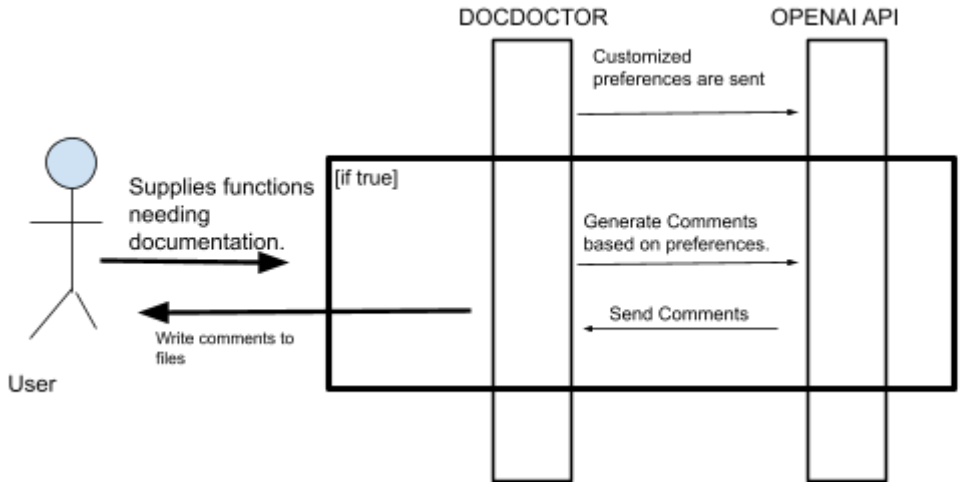
Before generating comments, DocDoctor checks the user's preferences for commenting style [S1], generates comments according to these styles [S2], and then attaches them to the code [S3].

3. Subflows:

[S1] DocDoctor retrieves the user's predefined commenting style preferences.
[S2] Comments are generated in line with the preferred style (e.g., verbose, concise, Javadoc, or Python docstrings).
[S3] Styled comments are attached to the appropriate places in the code.

4. Alternative Flows:

[E1] User has not set any preferences for commenting styles.



Fifth Use Case: Language-Specific Commenting

1. Preconditions:

The project includes files in multiple programming languages.

2. Main Flow:

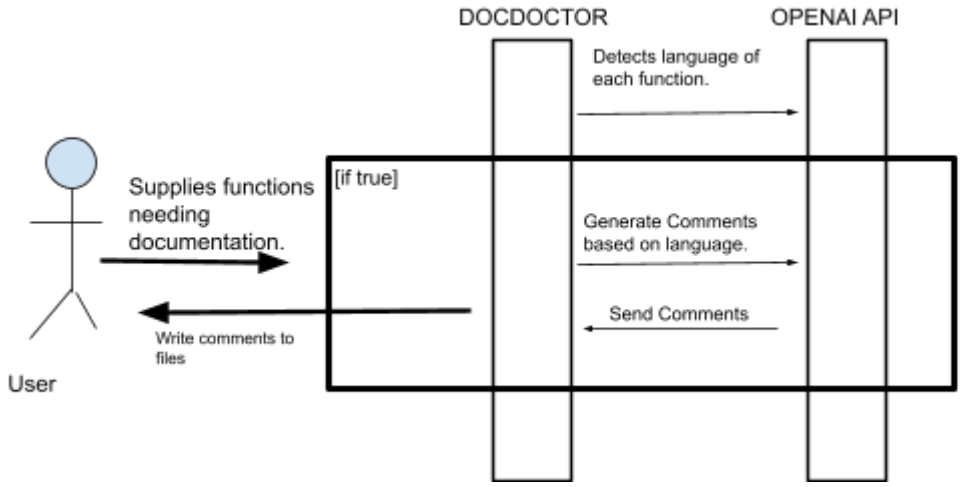
User selects files or directories for documentation [S1]. DocDoctor identifies the programming language of each file [S2], generates language-appropriate comments [S3], and attaches them accordingly [S4].

3. Subflows:

[S1] Selection of files or directories by the user.
[S2] Language detection for each selected file.
[S3] Generation of comments using syntax and conventions suitable for the detected language.
[S4] Attachment of comments within the code, respecting language-specific documentation standards.

4. Alternative Flows:

[E1] The detected language is not supported by DocDoctor.



Process Deliverable

Our team will be using the code-and-fix SE Process Model

Code-and-Fix

GitHub repo link - https://github.com/rudyP123/CS3704-DocDoctors

source code -

```
# How to use
# python doc_generator.py python path/to/your_file.py
# python doc_generator.py java path/to/your_file.java
```

```python
import ast
import sys
import os
import configparser
from openai import OpenAI

def load_api_key(config_file='config.ini'):
    config = configparser.ConfigParser()
    config.read(config_file)
    api_key = config['openai']['api_key']
    return api_key

client = OpenAI(api_key=load_api_key())

def generate_documentation(prompt):
    try:
        response = client.chat.completions.create(model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ])
        return response.choices[0].message.content.strip()
    except Exception as e:
        print(f"An error occurred: {e}")
        return "Documentation generation failed."

def add_docstrings_python(filepath):
    output_filepath =
f"{os.path.splitext(filepath)[0]}_documented{os.path.splitext(filepath)[1]}"
    with open(filepath, "r") as file:
        lines = file.readlines()

    def get_function_start_line_numbers(lines):
        function_start_lines = []
        for i, line in enumerate(lines):
            if line.strip().startswith("def "):
                function_start_lines.append(i)
        return function_start_lines

    function_start_lines = get_function_start_line_numbers(lines)
    for start_line in function_start_lines:
```

```
        function_lines = []
        for line in lines[start_line:]:
            function_lines.append(line)
            if line.strip().startswith("def ") or "return" in line:
                break
        function_text = "".join(function_lines)
        description = "Provide a concise summary for this Python function."
        docstring = generate_documentation(description)
        docstring_formatted = f'"""{docstring}"""\n'
        lines.insert(start_line + 1, docstring_formatted)

    new_source = "".join(lines)
    with open(output_filepath, "w") as file:
        file.write(new_source)

    return output_filepath


def add_javadocs_java(filepath):
    """Reads a Java file, adds Javadoc comments to methods."""
    output_filepath =
f"{os.path.splitext(filepath)[0]}_documented{os.path.splitext(filepath)[1]}"

    with open(filepath, "r") as file:
        lines = file.readlines()

    new_lines = []
    for i, line in enumerate(lines):
        if (line.strip().startswith(("public", "private", "protected")) and "(" in line
and ")" in line and "{" in line
                and not any(keyword in line for keyword in ["class ", "=", ";"])):
            method_signature = line.strip()
            description = f"""Write a detailed but concise Javadoc comment for the
following
            Java method: {method_signature}"""
            javadoc = generate_documentation(description)
            new_lines.append(javadoc + "\n")
        new_lines.append(line)

    with open(output_filepath, "w") as file:
        file.writelines(new_lines)
    return output_filepath
```

```
def main():
    if len(sys.argv) < 3:
        print("Usage: script.py <language> <input_file_path>")
        sys.exit(1)

    language = sys.argv[1].lower()
    input_filepath = sys.argv[2]
    output_filepath = ""

    if language == "python":
        output_filepath = add_docstrings_python(input_filepath)
    elif language == "java":
        output_filepath = add_javadocs_java(input_filepath)
    else:
        print("Unsupported language. Only Python and Java are supported.")

    print(f"Documented file created: {output_filepath}")

if __name__ == "__main__":
    main()
```