Doc Doctors Final Project Presentation

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

Problem

Challenges in Documentation:

- Keeping documentation up-to-date with rapid code changes is a common challenge in software development.
- Manual documentation is time-consuming and can become a secondary priority, leading to outdated or incomplete information.
- Inconsistencies in documentation can arise from multiple developers working on the same codebase.

Impact on Teamwork and Collaboration:

- Outdated documentation can cause miscommunication within teams, leading to errors and reduced efficiency.
- New team members may struggle with onboarding due to the lack of clear and accurate documentation.
- The overall quality of software can suffer without thorough and up-to-date documentation, impacting collaboration and the end product.

Solution

AutoDoc: Automating Documentation

1. Automation: AutoDoc leverages AI to automatically generate documentation for new code commits.

2. Minimal Effort: Reduces the manual effort of documentation, ensuring it keeps pace with development cycles.

3. Efficiency: AutoDoc streamlines the documentation process, allowing developers to focus on coding rather than documenting.

4. Consistency and Clarity: Ensures consistent and clear documentation across all new commits, improving codebase readability.

5. Collaboration: Facilitates better team understanding and collaboration by providing up-to-date documentation for every change.

Related Work

Swimm, Doxygen, SwaggerUI, Redoc, DapperDox, RapiDoc

Static Analysis Documentation Generators

Swimm - tool for internal documentation, i.e. documentation that helps software teams manage and transfer knowledge about how their code works and why it was built the way that it was.

Doxygen - extracts existing documentation from source code and other files and generates it in a structured format.

API Documentation Generator Tools

Swagger UI - visualize and interact with API resources without accessing the relevant implementation logic.

Redoc - open source tool for generating API documentation according to the OpenAPI specification.

DapperDox - open source, out-of-the-box solution for API documentation.

RapiDoc - interactive API documentation tool that lets you create visually appealing documents

Related Works Cont.

Discussion Presentation Topics -

GitHub Copilot - relies on existing documentation to generate functions.

Assessing the Quality of GitHub Copilot's Code Generation

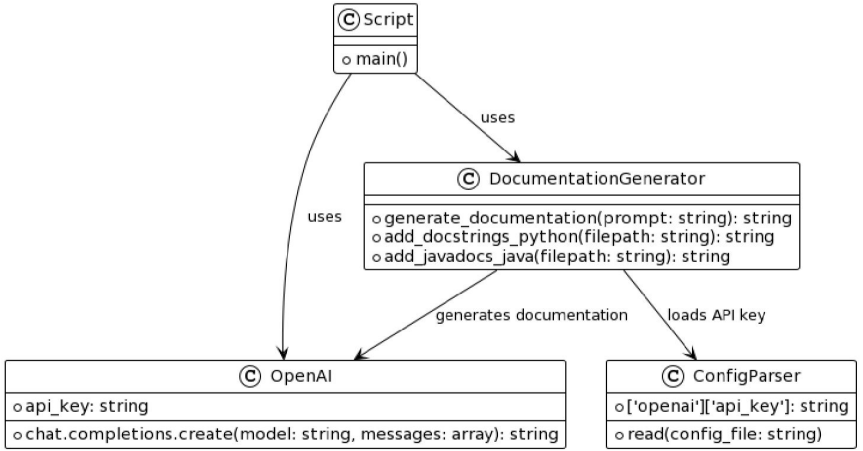
An Empirical Evaluation of GitHub Copilot's Code Suggestions

ChatGPT - generates documentation if provided functions, security risks with copy pasting sensitive company data. Requires manual user input, not automated.

Relevant Concepts From Class

1. High Level Design - We used the Pipe and Filter Pattern for our structure which proved to be useful for our specific use cases. For example, we needed to check what format the code is in, what language, and then start processing, understanding, and documenting. These were our data ‘pipes.’ The filters we aimed to implement were code entry, language detectors, our parser, and generator.

2. Low Level Design - We aimed to use the factory method design for AutoDoc because it enhances flexibility and scalability in creating documentation for various programming languages, which is what our tool aims to do. This pattern is a part of the creational pattern family, and focuses on creating objects in a manner suitable to the situation



Relevant Concepts Cont.

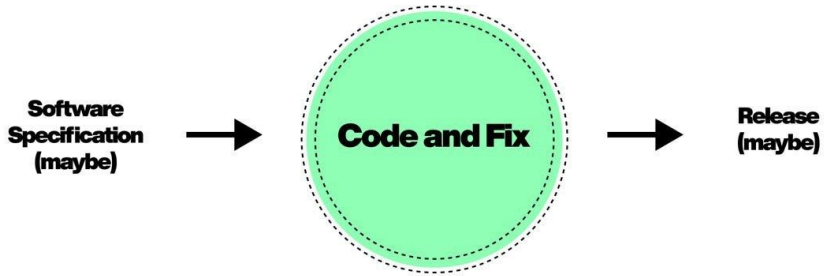
Code and Fix Software Development Model - Although this may be a controversial software process model due to lack of structure and planning, it worked for our project since it wasn't a large scale project that required an entire team working on the backend script at the same time.

Description from class slides:

Code-and-Fix

Based on a given a project spec

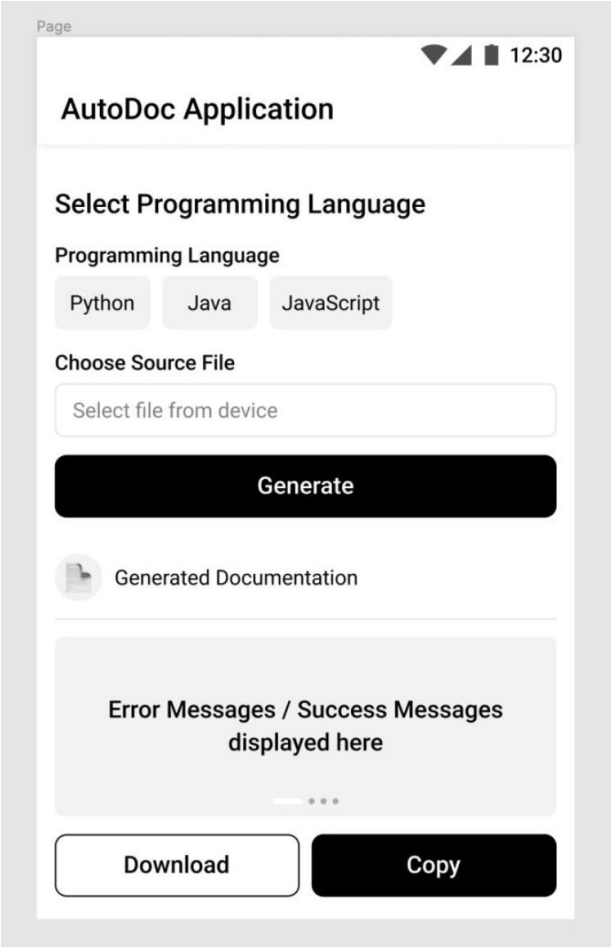
1. Write code
2. Improve it
3. GOTO 1



Future Work/Additional Features

Ideally, if we had more time, we would work on implementing a front end for our project whether it be a web application or a mobile application. We have a fully functional backend as of now, which was a huge step in the right direction for our product, but we did not have the time or resources available to start working on the frontend

The mock user interface on the right hand side is what our application would look like, and we would work on implementing additional features such as support for more languages,



Live Demo + Use Cases

Use Case 1 - Calculator.java

We will now test our AutoDoc script on a simple calculator file with an add and subtract method. It will take in the undocumented file and output a new documented file. Let's see how it performs.

Use Case 2 - Calculator.py

For our second use case, we have the same exact calculator class with add and subtract functions, however, it is a python file.

Implementing the python functionality for AutoDoc proved to be more difficult than implementing the java functionality, because python is more sensitive with indentation and doesn't have brackets for functions like java, so it is harder to identify where to insert the docstrings.

Use Case 3 - UserManager.java

Here's a more practical example with a Java file that AutoDoc could be useful for. This example simulates a simple user management system, featuring a UserManager class that handles operations like registering and updating user details in a system.

The methods perform operations like registering, fetching, updating, and deleting user information in a simulated database (a HashMap). This is a good example to test the detection and documentation of class structures, method functionalities, parameter usage, and error handling in AutoDoc.

Q&A