AutoDoc: Enhancing Documentation Efficiency in Software Engineering

Final Report By Doc Doctors

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

Abstract

"AutoDoc," is an innovative documentation tool that is designed to automatically generate and update code documentation in response to new code commits. By integrating this code documenter directly into the development workflow, AutoDoc can help promote enhanced communication and collaboration among software engineering teams. Its primary goal is to enhance overall software quality and increase the efficiency of code and developers, ensuring that documentation evolves in tandem with the codebase. This report addresses the difficulties of keeping documentation current within fast-paced software development environments, both large and small. It will talk about the already existing documentation tools to compare what is different and show the implementation process as well as deployment and maintenance of it.

Introduction

Maintaining up-to-date documentation in a workplace that is constantly expanding, whether it be a personal project or even a government contract, can be challenging. There are more chances than ever to have more errors within code as everyone is trying to be productive and push new code and innovate new things, however, this very thing can cause developers to fall behind. A lot of code that is pushed is usually not documented well, or in some cases not documented at all.

This causes outdated or misaligned code to still be within databases and repositories and no one knows what to do.

This disconnect can lead to several different problems with decreased productivity and inefficiency. When time could be spent working on new things, developers often find themselves trying to decipher old pieces of code to try to understand and see if it is even necessary. Fortunately, AutoDoc is a proposed solution to these definitive problems. By introducing a system that can document code in real-time, there can be more production output and efficiency increases throughout all software engineering processes.

AutoDoc, unlike manually commenting all of the code, is tailored perfectly for development workflow. It can be run as soon as a function or class is finished and currently works well with Python and Java. By running AutoDoc as soon as a function is finished, developers no longer have to spend more time commenting and can just move on while AutoDoc understands and translates the comments into readable and understandable format. This immediate process will show the application of any such project and will be a pivotal tool to enhance the development process from team collaboration all the way to deployment.

Related Work

As research for AutoDoc was conducted, numerous other similar tools were studied such as GitHub CoPilot, JavaDoc, and Doxygen. There are many other tools that offer easy API creation and many more. These frameworks were there to assist in generating documentation for large bases of code. Alongside the implementation research, there was more research done on other

automated testing tools which were necessary in ensuring quality documentation and reliable performance. Other static document generation, such as Swimm and Doxygen, were helpful resources that aided in understanding the process of transferring data points and extracting pre-existing code documentation.

With the existing tools being as advanced as they are, there is still a gap in their ability to progress as fast as needed in a software development workplace. This is where AutoDoc is crucial. Unlike these other tools, the real-time documentation generation allows for this gap in current technology to be reached. This allows workplaces to change to any adaptation and it ensures everyone understands it and no one is left behind.

Implementation Process

Agile Methodology was applied during this project as it was in strong alignment to the nature of this documentation development. Agile is known for adaptive planning and development and AutoDoc needed to be an always-improving model. Agile's emphasis on incremental development was a good choice for regular updates of the development process. It allowed the production to be flexible with new requirements and changes.

Testing was also done on a small scale. This testing of the code to document personal code allowed group members to submit code and check and push it on to others to ensure that the process was going smoothly. Any incorrect matching resulted in better adoption of OpenAI's API and slight adjustments to the source code.

Through this methodology, the development of this tool ran smoothly with many changes that helped better the end product. Plans to improve upon it in the future are set with new phases and tests coming through to meet the demands of software development environments. This now ensures that the AutoDoc remains a versatile tool for all developers to utilize to output high-quality code.

Low-Level Design

The Factory Method will be used for AutoDoc due to its ability to enhance flexibility, scalability, and adaptability to different coding languages. The AutoDoc currently works well with both Python and Java. The Factory Method uses an interface for creating objects, classes, and subclasses. For AutoDoc, a document generator tool could be the main surface that can define each method for documentation generation. Within each method, the subclasses of the factory would implement specific documentation, for example, either PythonDocGenerator or JavaDocGenerator.

By using this approach, it is easier to implement more languages due to the change of individual subclasses. Some benefits of using this pattern include the flexibility of change. Instead of changing all of the code to add another language like C, we are able to add a singular subclass. This also helps people who may work on implementing this to not understand the full scope and only need to understand their portion of the code. AutoDoc will utilize OpenAI's API to generate the comments once it asks the user for language selection. This also exemplifies how easy it may be to add more languages without having to redo major parts of the code.

High-Level Design

The AutoDoc uses the Pipe and Filter pattern to structure all the documentation effectively. This pattern was chosen due to its productive processing of large amounts of data and error handling through various phases. The system of AutoDoc has several filters, each having a specific function so that smooth data handling and code are commenting.

The initial phase is called the Code Entry phase. In this phase, the source code is run through many inputs such as local and remote repositories, and prepares it for better understanding. Next, the code is then filtered to identify if it is the correct language that was inputted by the user or just identify it completely. This phase is responsible for parser analysis. In this phase, the filter structures the code by matching functions and classes and organizes them in a format such that everything is understandable and readable to the computer. In the Analyzer phase, the system then reviews all parsed code to check for existing documentation and either adds onto it or rewrites it. The Documentation Generator is when AutoDoc writes down all the documentation. Next, the output manager is like a helper function in the sense that it helps output the documentation correctly. In Pipe and Filter patterns, there tends to be a conflict between managing proper output production, so this filter is necessary to ensure that all the code is correct, nothing has changed and the documentation is quality checked.

With such implementation of this pattern, it is simple to scale this production through new languages and more features. Filters can be added seamlessly to run through more programming languages and other filters could be added to export as different file formats too. This AutoDoc implementation makes it a versatile tool. The scalability is also effective in ensuring that any

filter can be changed at any time, without disrupting other filters, to be more efficient. For example, the Parser filter can be implemented in order to handle more data inputs. Alongside this, the Pipe and Filter method provides a great error-handling tool. The filters that are placed, and ones that can be added, ensure that no data is corrupted and that all the documentation is of great quality.

Testing

To test AutoDoc we created multiple programs in different coding languages, some that needed documentation and some that did not. For example, we created a calculator program in Java and Python, with simple add, subtract, and multiplication functionality. We used these files to test the consistency and functionality of AutoDoc, as the documentation across both languages should be relatively similar. We also used methods with varying parameters and return types to make sure that all aspects of our program were being documented accurately. Additionally, to test that AutoDoc does not do more than it should, we tested it with files that did not need any documentation. Such as files with no functions to be documented, or functions with existing documentation. We would run these tests expecting that no documentation would be added to the files.

Deployment and Maintenance

For the deployment and maintenance of this codebase if we were to release it would be relatively easy because of the current structure of our code. We would want to extensively test the codebase before the actual deployment. We would continue working on our frontend, and deploy our code as both an application and a website. This way both users on mobile devices as

well as computers would be able to utilize our service. As for maintenance, we want to keep adding different programming languages that we can support for the documentation. As new languages are developed, we want to stay with the current most popular languages. Our biggest priority when it comes to maintenance is making sure that our current baseline works as expected and seeing if any small details can be added to improve our service.

Conclusion

AutoDoc is a tool that can greatly improve code documentation and its efficiency by automating many processes in real time. This tool builds the gap between current technology and improves both the software quality and overall team and collaboration productivity. Throughout this development process, agile methodologies were enabled allowing for the flexible implementation of many different phases and changes. The Factory and Pipe and Filter pattern, provided for the scalability to accommodate better features and more languages. Overall, AutoDoc is an amazing tool that will continue to be developed to support all types of software developments as it does have many limitations still. Future work will focus on refining the

Despite its achievements, AutoDoc has limitations, including its current support for a limited range of programming languages. Future work will focus on expanding this range and integrating advanced AI to refine the contextual accuracy of documentation. AutoDoc sets a robust foundation for ongoing enhancements in automated documentation technology.

References

[1] “15-410 Coding Style and Doxygen Documentation,” 15-410 coding style and Doxygen documentation, https://www.cs.cmu.edu/~410/doc/doxygen.html (accessed Apr. 30, 2024).

[2] Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

[3] Dima, A. M., & Maassen, M. A. (2018). From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. Journal of International Studies (2071-8330), 11(2)

[4] P. Rehan, “I just improved my code documentation using SWIMM.,” Medium, https://pinjarirehan.medium.com/i-just-improved-my-code-documentation-using-swimm-60e8195790a6 (accessed Apr. 30, 2024).