Columbus

AI-BASED AUTOMATED DOCUMENTATION GENERATION AND CODE EXPLANATION

Documentation is hard.

But why does it have to be so?

Author
ANURAN ROY

Date
1ST APRIL 2023

Agenda

- The problem with documentation
- AI Tools to the rescue
- The flaws in the "rescue operation"
- Columbus - A possible way out
- What Columbus will explore
- Monetising Columbus
- Moving forward

</>

The problem with documentation

WHY IS DOCUMENTATION HARD?

Documentation is never complete.

DOCUMENTATION ALWAYS FAILS TO CAPTURE THE ENTIRE SCENARIO

Yes, you read that right. Documentation always tends to fail to capture the entire scenario.
What about documentation of major products like Spring, Django and Selenium? Yep, even them. Spring Boot's documentation is a personal favourite of mine, yet there are flaws that I often find, as I mention below.

CONTEXT IS NEVER CAPTURED FULLY IN DOCUMENTATION

When was the last time you got to know about the possible usage scenarios of a completely unknown function without referring to external examples or digging through the code? Don't stress yourself too much, it's probably **Never**. Some languages like Java try to capture this self-explanatory notion through usage of expressive in-built names, and others try to follow suit through various naming conventions (Yes PEP and ECMA, I am talking about you), but even this is often clumsy and drains productivity.

An example

```
public bool isEqualsIgnoreCaseOrSubstring(String sampleString, String inputString) {
    if (inputString.equalsIgnoreCase(sampleString) || inputString.contains(sampleString)) {
        return true;
    }
    return false;
}
```

This is a simple function that returns whether the string "sampleString" is equal to (ignoring case), or is a substring of another string "inputString".

From here, the developer can do only the following:

- Keep on the same naming conventions, notwithstanding however clumsy writing the code becomes, along with inevitable productivity hits that only mounts with time.
- Capture the context with simpler names, with some mechanism for documentation.

Capturing the context in documentation

CONTEXT TO THE RESCUE

Capturing the context is a brilliant idea.

HOW IT IS DONE?

At the end of the day, huge codebases all boil down to the first class members of (almost) any programming language - functions and its related members. An ideal function is a complete blackbox that abstracts all details, enabling the user to be concerned with only what data it takes in and returns, insulating all internal mechanisms. Thus, documenting the context of a function is an important first step in coverage of the complete context.

SCALING THE SAME CONCEPT

At present, there are automated tools that do this well at the level of functions, and apply it for all functions. Tools like Javadoc (for Java), Doxygen (language-agnostic), Pdoc (for Python) do this, generating the documentation of individual functions with mainly the following fields: input format, output format, and a standard description of the operations. But this has a caveat: How do you determine a description suitable enough?

But no use of a brilliant idea with poor execution.

SO WHAT NOW?

AI Based tools to the rescue.

AI TOOLS ARE TRAINED TO ACTUALLY CAPTURE CONTEXT

Thanks to different modern AI-based architectures like Transformers and RNNs, AI can accurately capture context through attention mechanism, and render a suitable description through the context. Also, vector databases are a proof that data can be stored in neural networks in an extremely compressed format, comprising of numbers. Large Language Models (like GPT) are a demonstration of the same.

DETERMINING THE CONTEXT BECOMES THE MAIN CONCERN

Now the main concern becomes determining the context that needs to be captured. A lot of tools still rely on the aforementioned conventions and model their tools exactly after that, or extend it somewhat keeping the basis of the documentation same - just the problem evolves into finding a better and more concise function description, but that is only half-solved.

</>

COLUMBUS - A possible way out

SOLVING FOR THE HARDEST PART OF DOCUMENTATION

About Columbus

COLUMBUS IS NOT A SINGLE PRODUCT, IT'S A PRINCIPLE

COLUMBUS stands for **COntextual Long User-generated eMBddings for codebaSe** , and is the approach I propose here for generating user documentation. It captures context through the use of the following mechanism:

- **Tracing code calls**: Inspecting the code call stack of the entire project, or whatever segment the user chooses to, through entrypoints in code (eg.: `// columbusEntry()` for Java).
- Generating sequences consisting of code calls and adding them to a vocabulary specific to a particular project, or with a user-defined scope.
- Using one or more language models to trace and describe the connection between the code calls (say input types, output types of every members, ways it can be called and what each type of call means, etc.)

BREAKING DOWN IN LAYMAN'S WORDS

Columbus scans every line of the project, generating a mental sketch in itself of how the different components of the code base are connected. From the mental sketch that it generates, it describes how the entire code base works from both top-down & bottom-up approaches. This lets the reader gain comprehensive understanding from simply reading the documentation which now isn't unlike a research paper or a textbook!

Monetization Opportunities

GENERATING REVENUE FROM A RADICAL IDEA

Possible Revenue Model(s)

DOCUMENTATION AS-A SERVICE (DAAS) FOR IDES AND CODE EDITORS

As a plugin, **COLUMBUS** can be used for generating revenue through **DaaS** model, not very different than GitHub CoPilot, which specifies usage quota and available models for use based on subscriptions. It can include a free trial as well to get users hooked to the idea of auto-generating documentation on the level of dissertations and research papers, providing a comprehensive view of code from maximum possible perspectives.

RAPIDAPI MODEL FOR TEXTIFY AI MARKETPLACE

We can also take the RapidAPI approach, providing testable endpoints with API keys and letting users upload limited amount of files/text, to get a generated set of files/documentation strings from the API. This can be an add-on to the previously described model as well.

The potential of impact of good documentation

Team productivity gains

+>124%

Code correctness

+15x

Possible market size (in bn $)

20+