Code Commenting: How to properly comment code

M.T White

Abstract

A good codebase is a well-documented codebase. Many scientific programs are being developed by scientists as opposed to formally trained developers. As such, the need for scientists to know how to properly document code is growing. A well-documented codebase allows scientists to write better, easy to understand code that will ultimately free up resources that can be allocated for more productive tasks such as conducting their research as opposed to spending countless hours trying to modify a codebase that needs to meet new requirements. This document will give a brief tutorial on how to properly write code comments as well as API-level documentation that will make scientific software easier to understand and much more modifiable in the future.

1 Introduction

Programming has become so ingrained into modern society that a lot of software is written by developers that do not have any formal education in programming. As such, many developers who lack formal training in software development will usually document code poorly if at all. Poor code documentation is not only prevalent among non-formally trained developers it is also prevalent among formally educated developers.

If the Covid-19 crisis taught anything to software developers, it should be that a software system can be in place for many decades after it has been introduced. The pandemic revealed that states such as New Jersey, Kansas, and Connecticut were using COBOL-based systems that were 40 years old [1]. The key here is to understand that a software system can and will age and as the needs change for the organization or project that uses it so to will the software need to change. As such, the code should be maintainable and easily understandable. There are many best practices on how to write maintainable code; however, what is often the simplest and arguably the most effective is proper commenting and documentation on a codebase.

Scientific programming is no exception to this rule. Scientific software is often used to conduct research or help engineers develop new systems. This means that the underline requirements are fluid, as new information is uncovered or as requirements for the project change or evolve so too will the software also need to change to accommodate the new or modified requirements. As such, when discoveries arise, or new factors need to be considered the program will need to be easily modifiable to accommodate.

Code documentation is very important for scientific software. Scientific software can be complex and many of the developers are scientists and not formally trained software developers with backgrounds in software development. One study suggests that scientists spend up to 30% of their time developing scientific software [6]. The same study also suggests most of the scientists that develop scientific software are not formally trained [6]. As such, for a piece of scientific software to last, it must be modifiable, and for it to be modifiable it must be understandable. Therefore, proper code documentation is very important for these codebases. Since it is no secret that scientists and engineers alike favor the Python programming language the following will explore why code documentation for scientific software is important and how to document code properly using the Python programming language.

2 What is code documentation?

There are some developers that feel that code documentation is unnecessary, and that software should be written in such a way that the code documents itself [2]. For many, especially the author and the works cited in this article greatly disagree. Code documentation provides an easy way for developers, to understand the components of a codebase [3]. In other words, documentation helps developers understand what is going on in the codebase. It is going to be quicker for a developer new to a codebase to get up to speed when they can read easy to understand language over sifting through hundreds if not thousands of lines in codebases that can be any number of years old. For example, it is documented that a set of chemical engineers were tasked with modifying a poorly documented codebase to accommodate new requirements after the original authors left [5]. However, the documentation was so poor that parts of the software had to be re-written from scratch [5]. In other words, the engineering team had to allocate resources that were of better use actually working on their assigned task to the redevelopment of poorly documented software because new engineering factors were discovered, and the software could not be easily modified to accommodate due to the documentation.

So, what is code documentation? Code documentation is anything that will help people, including the developer understand what the code does [2]. When many developers think of documentation, they think that documentation is for other people. However, what many developers forget they are usually the ones who are going to support the code a few weeks or a few months down the road.

What should be documented?

What should and should not be documented is going to be largely dictated by the organization. A good organization will at the very least have some standards that relate to documentation. However, at the least, a decently documented codebase will have a short description of the classes, methods, and variables. Complex blocks of code should also be documented to help others understand what is going on. Generally, it is a good idea to add a short description to the class, methods, and variables like the following,

```python
#This class is for reading and writing XML
class XMLReadWriter:

    #Number of files to read
    NumberOfFilesToRead = 10;

    #This method reads an xml file
    def readXML(self):
        print("read xml file")

    #This method reads a xml file
    def writeXML(self):
        print("write xml file")
```

It is common to hear arguments that state things like getting and setter methods as well as private variables, and so on should not have to be commented. However, as stated before, how much or how little the code is documented is largely up to the organization.

2.1 Inline code comments

Comments are meant to be seen by developers working on the source code. Inline comments are only accessible from within the codebase and are similar to the text following the hashtag in the last example. In short, to see an inline comment the developer must have the uncompiled source code. An inline comment should be used as a short note to other developers about how a piece of code works. It is common to put an inline comment at the top of a method, variable, or logic block like the following:

```
#This is the rotation of the machine.
rotation = 32
```

2.2 API-Level documentation.

API level documentation works mostly the same way. However, for the API documentation to be of any use outside of the original authors all public attributes should be documented using a tool like Javadocs. If an attribute is public, it can be used outside of the class that it is declared in, and as such the outside developers need to know what that piece of code is intended for and how to use it. However, it is also common to provide API-level documentation for every attribute such as private variables, classes, interfaces, and so on.

In Python, a common tool used to create API-level documentation is called Docstrings. The documentation in a Docstring is called via the print command. For example, a Docstring for a method may look like the following:

```
def DocstringExample():
    """This is a docstring example"""
```

The Docstring is displayed with the following command,

```
print(DocstringExample.__doc__)
```

As can be seen, the Docstring is called with the print() method. In the parentheses, the method name is followed by .__doc__.

Every organization is going to have a different philosophy on what should be documented and how. However, there are some good and bad practices that when employed can be the difference between a long-lasting codebase and a codebase that will have to be re-written when the original author leaves. As such, the following section will be dedicated to what to document as well as good and bad documentation.

3 What comments should look like?

In lower-level, undergrad programming classes students are required to comment every line of code. Instructors do this to have the student explain the logic to the instructor in the code and have the student write their thought process out, so they understand it. However, in the real world commenting on every line of code is a terrible idea. An excess of comments can overwhelm other developers and cause them to miss important information.

3.1 A good comment

```
#This method determines if a person can buy wine or not
def ageCheck(age):
    if (age >= 21):
        print("can buy wine")
    else:
        print("cannot buy wine")
```

Consider the above code snippet. The snippet is a simple method that determines if a person can buy wine. The comment at the top of the method is short, simple, and is easy for anyone to understand. It is a single line and clearly states the purpose of the method. For this type of method, a developer wants to usually keep the remark to a line or two.

A comment can also be used to help explain the logic in a code block. Consider the example,

```
#This method determines if a person can buy wine or not
def ageCheck(age):
    #if a person is 21 or over they can buy wine
    if (age >= 21):
        print("can buy wine")
    else:
        print("cannot buy wine")
```

In this case, the new comment explains what the logic does. For a simple code block like this one, a comment would not usually be necessary. However, for logic like complex loops, mathematical computations, and so on it might be a good idea to comment on what the code is both doing and how it works. Every code block is different and as such, the developer must use their own discretion on how much or how little to document the code block.

3.2 What comments should not be.

A comment should not be a step-by-step play of the logic. As stated before, a developer should not get bogged down with commenting every line of code. The following is an example of a poorly commented method.

```
#This method determines if a person can buy wine or not
#The method will take in an input name age that will be used to store a
#person's age.
#If the age is greater than or equal to 21 the person can buy wines if they
are under 21 years of age
#the person will not be allowed to buy wine.
def ageCheck(age):
    if (age >= 21):
        print("can buy wine")
    else:
        print("cannot buy wine")
```

This code snippet is an example of a comment that will overwhelm the developer with useless information. Outside of the very first line that provides context for the method the rest can be easily deduced.

Another example of poorly documented code can be seen below.

```
#This method determines if a person can buy wine or not
def ageCheck(age):#define the method
    if (age >= 21): #check if the person is 21 or older
        print("can buy wine") #if the person is 21 or over print, they can
buy wine
    else:#else if they are younger than 21
        print("cannot buy wine")#print they cannot buy wine
```

This is the type of code commenting that is common in undergraduate programming courses. This type of code commenting is very poor because it clutters the code and provides no useful information. Anyone that is writing or modifying a Python program should be able to follow the code and understand what the lines in the method do. This is kind of contradictory to what was said earlier. In short, it was said before that code does not necessarily have to document itself, which is true; however, the code should be readable, and one should be able to, at the very least, have a basic understanding of what the code is doing by reading the code alone. In other words, comments are there to add context to code, tell other developers what the code does, and at times help guide other developers on how the code works.

There is a very fine line between properly documented code and code that is overly documented. Code that is under-documented can make code bases impossible to maintain and understand in the future while code that is over documented can clutter up the codebase and make it equally hard for other developers to extract meaningful information from the comments.   As with anything else, the commenting developer must use his or her own discretion.

In summary, every line of code does not have to be commented, in fact commenting every line of code is a bad practice. However, all methods should have an inline comment to give context as to what the method does. All variables should also have an inline comment to provide context. API documentation is important for codebases such as libraries and frameworks. API documentation is designed to be understandable to developers who consume the API and are not active in the development of the API. API documentation is usually more in-depth as opposed to inline comments and as such, it can be used in place of in-line comments. At the very least all accessible attributes that are in an API should have some API level documentation. As was stated time in again how much that is and is not documented is solely up to the organization and the developers. However, a well commented codebase will save time and energy for the current developer and any who will work on the codebase in the future. In short, well documented codebases free up resources that can best be put to use trying to accomplish the goal the software is assisting at as opposed to developing out the codebase.

References

[1]Gurchieck, K. (2020). *Desperate need for COBOL programmers during Covid-19 underlines importance of workforce planning.* https://www.shrm.org/hr-today/news/hr-news/pages/desperate-need-for-cobol-programmers-underlines-importance-of-workforce-planning.aspx

[2] Boersma, E. (2019) *Code documentation: The complete beginner's guide.* https://blog.submain.com/code-documentation-the-complete-beginners-guide/

[3] Tenny, T. (1988). Program readability: Procedures versus comments. *IEEE Transactions on Software Engineering*, *14*(9), 1271-1279.

[4] Keshav, V (2017). What is API documentation, and why it matters. *Swagger* https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/

[5] Prabhu P, Jablin TB, Raman A, Zhang Y, Huang J, et al. (2011) A survey of the practice of computational science. In: Proceedings 24th ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis. pp. 19:1– 19:12. doi:10.1145/2063348.2063374.

[6] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl and G. Wilson, "How do scientists develop and use scientific software?," *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 1-8, doi: 10.1109/SECSE.2009.5069155