5 RULES FOR COMMENTS TO DOC YOU OUT OF PROBLEMS

Overview: Commenting on code can seem like a chore but documenting changes and their purpose is key to helping others understand your code. These five rules serve as a handy guideline to help you make the most of your comments and make sure they're seen and understood by others.

Mention comments and documentation to a group of developers and you are likely to get a “heated discussion.” Documentation may be a good concept, but it is rarely even done, let alone done right, and can never be expected to be fully up to date. When we get to comments, it seems to be more a matter of degree. We see the need, but how many should we put in and how much should we have to rely on them?

If you struggle with commenting, you are really struggling with your logic.

We suggest following these 5 simple rules to form a guideline for using comments.

1. Your documentation should reside in your code, not outside. Having the documentation in the form of comments means that it travels with the code. There won’t be any hunting for it or questions about whether it is the latest. The only exception to this rule is when documentation on the usability of the software is written. This can reside outside the code, in the form of a Confluence page, development manual page, etc.

2. Comment on the line close to action. Comments need to be right where the code is. You shouldn't place comments lines above and expect someone to find it or realize it is related. When an update is done on a line, if the comment is far away, the comment may not be seen and therefore not updated. Keeping the comments and lines together makes understanding the code faster, and the comments are more likely to be current. It is important to remember that Integrated Development Environments (IDE) do not correlate comments with their corresponding code counterpart, hence why the location of the comments is important.

3.If logic relates to another section of code, comment there to relate. Something in the section you are in may have an impact in another part of the code. You need to form a bridge to make the connection visible. Building in impact analysis could actually point out problems with your code. If you feel the need for a lot of these comments, your structure could be bad. If your comments are trying to explain the code, then that simply signifies a need to refactor. If the code structure is already well established and refactoring cannot be done, it is important to comment on the way it should function when the refactoring is complete. This helps future developers understand why code smells exist.

4. For automated testing, a general rule is to leave comments in relation to what that test will do. If an automated test fails, then the individual in charge of maintaining these tests can use the comments to debug the primary process of the test and figure out what the issue is, minimizing the time to debug the issue. Comments give you the opportunity to search for keywords in the future and help you see where the code has deviated from the workflow of the given test, and whether the test case needs to be changed to reflect that. Write the general sense of what the test will do. Link your tests to the code, not the other way around.

5. Pay as much attention to the comments as to the code. Don't make comments an afterthought, don't ignore them in code reviews, and most importantly, continue to update them as the code continues to change.

It can be easy to overlook comments because of the concentration on code, but they are important. Following these five rules can help you make the most of comments and treat them as what they are-an investment in the future.