Nicole P. Satiembre **March 12, 2024**

Cebu Technological University-Main Campus

Clean Code

Chapter 4

4. Comments

We should try to make our codes as self-documenting as possible. The only use of comments is to express things we fail to express in our codes but their use is not a cause for a celebration. Every time we catch ourselves needing to write comments, we must try to think through other better ways we can write our code. Comments are bad because codes constantly evolve while comments remain static and old. Truth can only be found in one place: the code. Only the code can truly tell you what it does.

Comments Do not make up for Bad Code. Another reason for writing comments is bad code and we tend to save it by writing comments. We shouldn't be satisfied with this, instead, we must fix our mess. Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments.

Explain Yourself in Code. Instead of writing comments, as much as possible, we must try to explain ourselves in our code, this can be done by using functions and choosing appropriate variable names.

Good Comments. Comments are necessary or beneficial however, the only truly good comment is the comment you found a way not to write.

Legal Comments. There may be times when we will need to write comments for legal purposes. For example:

// Copyright (C) 2003,2004,2005 by Object Mentor, Inc. All rights reserved.
// Released under the terms of the GNU General Public License version 2 or later

Comments like this should not be contracts or legal tomes. Where possible, refer to a standard license or other external document rather than putting all the terms and conditions into the comment.

Informative Comments. Informative comments can be useful. For example, a comment lets us know that the regular expression is intended to match a time and date that were formatted with the SimpleDateFormat.format function using the specified format string. In such instances like these, comments become a big help.

Explanation of Intent. In some cases, comments don't just provide useful information about the use case of the code but it also provides the intent behind a decision.

Clarification. Sometimes comments are used to clarify obscure arguments or return values. However, there is a substantial risk of incorrect clarifying comments. That is why, before we write a comment, take care that there is no better way, and then take even more care that they are accurate.

Warning of Consequences. Comments can also be used to warn other programmers about the certain consequences of running a code. It could be a warning to not run an expensive calculation unless you have some time to kill. In this case, a warning is highly appreciated.

TODO Comments. Sometimes, programmers list out their to-do lists as comments. It might be a reminder for them but cannot do the task as of the moment. Whatever it may be, it is not an excuse to leave bad code in the system. We don't want our code to be littered with TODOS so it is advisable to scan through them regularly and eliminate the ones you can.

Amplification. A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

Javadocs in Public APIs. When writing APIs, it is necessary that we also write good documentation for it. But keep in mind to be careful as it has the potential of becoming misleading, nonlocal, and dishonest as any other kind of comment.

Bad Comments. Bad comments are comments that are made to make-up for poor code or justifications for insufficient decisions.

Mumbling. If we decide to write a comment, we should spend the time necessary to make sure it is the best comment we can write. Our comment shouldn't be a mumble of what we feel. Any comment that forces you to look in another module for the meaning of that comment has failed to communicate and is not useful.

Redundant Comments. If a comment is not more informative than the code; it does not justify the code, or provide intent or rationale; it is not easier to read than the code; it is less precise than the code then that surely is a redundant comment. They serve no documentary purpose at all. They only clutter and obscure the code.

Misleading Comments. A subtle bit of misinformation, couched in a comment is harder to read than the body of code. Sometimes, even with the best intentions, programmers can make inaccurate comments. Comments like these are misleading and could make the lives of programmers hard.

Mandated Comments. Having a rule that requires every function to have a Javadoc or for every variable to have a comment is just plain silly. They only clutter up the code, propagate lies, and lend to general confusion and disorganization.

Journal Comments. Sometimes people add a comment to the start of a module every time they edit it. These comments accumulate as a kind of journal or log, of every change that has ever been made. These long journals do more harm than good as they only clutter up and obscure the code. They should be completely removed.

Noise Comments. When comments just restate the obvious and provide no new information, it is just plain noise. Eventually, the comments begin to lie as the code around them changes. We must be determined to clean our code out of noise.

Scary Noise. Even some Javadocs can become noisy if they don't serve any purpose at all, just pure redundancy.

Don't Use a Comment When You Can Use a Function or a Variable. As much as possible, we don't want to write comments. We can limit writing comments by making our codes self-documenting by choosing appropriate variable names.

Position Markers. Sometimes, programmers like to mark a particular position in a source file. Even I, do this as well when coding with C. However they only clutter up the code and they should be eliminated.

Closing Brace Comments. Sometimes we want to mark our closing functions especially if there is a long line of code. However, they seem very redundant. Instead, we would want to make our functions shorter and encapsulated so they are easier to read.

Attributions and Bylines. Although it might be good to write comments to remember who contributed the code, in reality, however, they tend to stay around for years and years, getting less accurate and relevant.

Commented-Out Code. It is very common to do this, even I, have done this thing as well. However, this only messes up the code and obscures it. A good thing to note is, if you don't need it right now then remove it.

HTML Comments. HTML in source code comments is a disgrace. It makes the comments hard to read even with an IDE.

Nonlocal Information. When writing a code, we must localize it, describing the codes near it. We don't have to write down systemwide information in the context of a local comment.

Too Much Information. We shouldn't put interesting historical discussions or irrelevant descriptions of details into our comments. It doesn't serve any purpose at all and is only redundant.

Inobvious Connection. We must make the comment explicit and obvious in describing what it meant to say. We don't want to be confusing. A comment shouldn't need an explanation.

Javadocs in Nonpublic Code. Javadocs are useful for public use APIs. However, they become redundant in codes that are not intended for public consumption. Creating Javadoc pages for the classes and functions within a system is typically not beneficial, and the additional formality of Javadoc comments is often nothing more than unnecessary clutter and a source of distraction.