Nicole P. Satiembre BS Computer Engineering-Intern February 20, 2024

Cebu Technological University-Main Campus

97 Things Every Programmer Should Know
Chapters 11-15

16. A Comment on Comments-Cal Evans

The writer starts by sharing a relatable experience about having a low grade on a coding activity where he thought all the functionality was working and nothing seemed to be wrong with his code. He then carried on to say that his teacher commented “No comments?” into his output. That left him with an unforgettable reason about the importance of commenting on your code.

Comments help you and other programmers in the team understand the code. It should be a common decency to comment out parts of your code as a guide. Modern languages have a tool akin to Javadoc that will parse properly formatted comments to automatically build an API document. This can be a very good start but it is also important to note the explanations about what the code is supposed to do making sure that the comments you write are relevant and not obscure.

17. Comment Only What the Code Cannot Say-Kevlin Henney

This chapter is a good addition to the previous chapter. However it is important to explain parts of our code by making relevant comments, we must also avoid the temptation of commenting too much.

If we comment out things that are self-explainable or very obvious, it doesn’t come off as helpful anymore, rather it becomes a noise. Comments that do not add value to the code can make the code lines longer and more confusing. Commented-out code is not executable code, so it has no useful effect for either reader or runtime. It also becomes stale very quickly. Comments should be added only when the code cannot explain itself. If you find parts of the code where you need to explicitly discuss it through comments, it is advisable to consider making the variable and function names self-explanatory. Instead of commenting sections in long functions, extract smaller functions whose names capture the former sections’ intent. Try to express as much as possible through code.

For final thoughts, the comments that we write should add some value for the reader, otherwise, it is waste that should be removed or rewritten.

18. Continuous Learning-Clint Shank

With the help of technology, development becomes rapid and it is distributed globally. We then realize that a lot of people are capable of doing our job. This means strong market competition. The only way to stay in the game is to keep learning so we can have marketable skills. Not doing so will cause us to be obsolete and one day we might no longer be needed.

It sounds unfair especially when we have spent money and time earning our degree only to realize that our skills have gone obsolete as time passes by. We cannot argue this reality. Some employers are generous enough to provide training to broaden our skill set but that may not always be the case. The reality is that we are responsible for our future. We shouldn't wait for somebody to teach us. In this world of technology, information is wide and only a few clicks away. It is therefore our efforts that are needed to keep ourselves updated with the latest skills.

As a programmer myself, I learned this earlier and started to teach myself by reading documentation, and blogs, joining forums, asking experienced people, and watching YouTube videos to help me learn new skills. The skills that I have today were not taught in school. I learned them all by myself. With the industry that we work in, rapid development is our enemy but we should ride the waves by continuously learning.

19. Convenience Is Not an -ility - Gregor Hohpe

This chapter is about the argument of "convenience". Most experienced programmers have learned that a good API follows a consistent level of abstraction, exhibits consistency and symmetry, and forms the vocabulary for an expressive language. Alas, being aware of the guiding principles does not automatically translate into appropriate behavior. Sometimes we want to use shortcuts like combining two separate calls into one making the method invocation harder to read. Oh, I remember the first chapter, "Act with Prudence and consider the consequences."

The metaphor of API as a language can guide us toward better design decisions in these situations. An API should provide an expressive language, which gives the next layer above sufficient vocabulary to ask and answer useful questions. For example, we prefer to say run instead of walk(true), even though it could be viewed as essentially the same operation, just executed at different speeds. A consistent and well-thought-out API vocabulary makes for expressive and easy-to-understand code in the next layer up.

For the final thoughts, we shouldn't fall into the temptation of lumping a few things together into one API method. Instead of convenience, this causes more trouble and, makes the code hard to maintain.

20. Deploy Early and Often - Steve Berczuk

As a novice programmer, I used to do the common bad practice of deploying the project only upon completion. Debugging the deployment and installation processes is often put off until close to the end of a project. The result is that the team has no experience with the deployment process or the deployed environment until it may be too late to make changes.

I have learned that having an installation process for a project will give us more time to evolve the process as we move through the product development cycle. It is also important to regularly run tests on the installation process to check for the assumptions that we may have missed in the code that relies on the development or test environments. As a personal experience, delaying the tests and deployment process causes the work to become complicated and causes more bugs. That is why it is important to regularly deploy and debug project parts so we can catch and fix errors early on without risking the whole program and time.