Nicole P. Satiembre BS Computer Engineering-Intern February 19, 2024

Cebu Technological University-Main Campus

“97 Things Every Programmer Should Know”

I. Act With Prudence - Seb Rose

An important best practice for programmers that I have learned and I guess should be thoughtfully considered every time we work on a project, especially at the beginning stages, is to know that whatever we take, act with prudence, and consider the consequences. As developers, we often face the conundrum between “Doing it right” versus “Doing it quickly”, especially when a project has tight deadlines and a lot of features need to be implemented.

As a novice programmer myself, I have fallen into this same trap by using premade modules, libraries, or functions instead of creating one on my own in hopes of finishing the project quickly. Doing things quickly without considering the best practices often causes a “Technical Debt”. This could compromise the speed or the size of the program. Though it gives a huge short-term advantage in terms of speed, often, in the long run, these shortcuts in the code make it harder to add features or refactor the code and are a breeding ground for defects. The longer we leave it the worse it gets and by the time comes, it gets harder to fix it and demands more time and risks. I have encountered this same relatable problem before when working on my projects. The longer I ignored the problem, the more bugs are showing up. It seemed like a domino of problems and it cost me more time to fix it. It is very frustrating! I have learned things the hard way.

Although we always want to follow the best practices in all of our projects, no methodology is a perfect one-size-fits-all. In circumstances when there is a need to do it, it is best to make a tracking log and fix it quickly. Finally, as words to live by that I have learned, applicable to every programmer starting a project: “Pay off technical debts as soon as possible”.

II. Apply Functional Programming Principles - Edward Garson

In this chapter, I have learned the importance of Mastering the Functional Programming Paradigm principles. In our programs, it is our goal to attain a higher degree of referencial transparency – that is the functions consistently yield the same results regardless of where they are invoked given the same parameters. Function evaluation depends less on the side effects of a mutable state because Functional Programming often utilizes immutable states. One leading advantage of functional programming is fending off defects often observed in imperative codes with multivariable, where some values are not expected in a particular situation.

On the other hand, in my personal experience, I have observed that whenever we leave our codes for some days and look back at them again, it could be hard to remember the things we have written. Understanding our imperative code is not as easy as it was when we were writing it. That is why visibility semantics is crucial to limit defects caused by unorganized mutability-designed systems. Functional Programming offers better readability which is a huge advantage for an ever more important purpose: maintainability. Functional programming plays a crucial role in future tech development because of its modularity. Modularity breaks down large and complex projects into simpler modules. You can test the modules separately, which lessens the amount of time spent on unit testing and debugging.

With the smart test-driven design of functional programming, there will be fewer defects, and often will be easier to debug because it is easier to track errors in these designs. However, this paradigm is not always an optimal solution in all cases. For example, in object-oriented designs, this style is better fitted with domain model development than with user-interface development. Otherwise, it is still a good new way to approach solving problems.

III. Ask What Would the User Do? (You are not the user) - Giles Colborne

One particular mistake we programmers make is to assume that everyone thinks the same. But we don't. This is often why it is hard for programmers to put themselves in the feet of the users. This is termed by psychologists as false consensus bias. Users don't think like programmers and they spend less time using computers thus they don't really have much of an idea in terms of troubleshooting patterns or usability patterns that most programmers are acquainted of.

In terms of understanding the needs of the user, we programmers would often fall into the spiral of guessing what the users would probably do. However, the best way to find out what a user thinks is to observe one. When we let them try a new technology, we don't want to interrupt them, instead, we watch carefully and see why they do and why they don't do certain things. By carefully observing the users, we can then deduce that the users do the same patterns; they try to complete tasks in the same order and often make mistakes in the same areas– and that area is where we want to focus on solving. As programmers, often when we get stuck about something, we look around for ways. However, when users are stuck around, they narrow their focus, which is exactly why tooltips are more helpful than menus, and it's best to locate them near the problem area, providing an obvious way to solve a problem.

Lastly, an important note for programmers is to observe the users as it is the best way to understand them rather than constantly guessing what they want and what they don't. It is also a faster and more direct way of learning how to give the best user experience to our users.

Automate your Coding Standards- Filip Van Laenen

At the beginning of a project, we often have this clear goal in mind to follow good coding practices for our fresh project repositories. However, during the middle towards the end of the project, these coding standards seem to be abandoned one at a time. This is sometimes because a team member didn't follow the instructions well or other members understood things differently. Sometimes, a team member would want to do their way. Others may have wanted to follow the best practice but when due dates are critical, they often take the risk. When the time comes for the project to be delivered, the code is a huge bulk of mess.

Following a coding standard without automation can be a hassle and idle but why should we bother to do so anyway? One important thing is to prevent programmers from writing anti-patterns that could lead to bugs. It is not a good idea to have individual coding patterns for each programmer working in a team, that could lead to messy code that is a breeding ground for errors that are hard to trace. There are helpful tools we can use that help automate coding standards. Such tools are code formatting tool which helps to format code everytime we compile. Another is to use the Static Code Analysis tool to scan the code for unwanted anti-patterns. It is also important to learn how to configure these tools and breaking the build test in case the coverage is too low. We must practice these coding standards diligently especially for the important tasks.

Of course, there will always be an exception to the rules. As for the things that we cannot automatically flag or fix, we can consider them as supplementary guidelines to our coding standards which our team member may or may not follow. Finally, the coding standards must be dynamic and not static. As the project evolves, the needs of the projects change and what may have seemed smart at the beginning of the project may not be necessarily smart a few months later. Our coding standards need to be updated and adaptive to evolving project requirements.