# **ACT WITH PRUDENCE**

In this chapter I’ve learned that you need to think before you act or do something, time management, Consider Long-Term Implications, Risk Management, most of the time is that you choose the easiest and fastest way where you just need to code where it is working but there’s so many bugs and loopholes and say to yourself that you can improve it later on (what you called a technical debt which is like a loan).

Also, when you choose the easiest and fastest way it has many consequences during the development or in the process of development where you’ll have to analyze or repeat the original problem ordo the code again to make it work as what the client says or other options like you need to correct the feature as soon as possible if you’re going to choose the easiest and fastest way to generate the said feature or system of what the client says before it turns out to be a big problem.

# **APPLY FUNCTIONAL PROGRAMMING PRINCIPLES**

In this chapter I’ve learned that mastering the functional programming you can greatly improve the quality of the code you write in other contexts this is where you need to learn the referential transparency where it enhances code predictability and maintainability, so that you don’t need to face some of mutable states in which you need the help of visibility semantics to see the defects or errors on your program.

However, the mutable state problem has been addressed with astute test-driven in being sure to “Mock Roles, not Objects”, typically has better responsibility allocation with more numerous, smaller functions that act on arguments passed into them, rather than referencing mutable member variables. There will be fewer defects, and furthermore they will often be simpler to debug, because it is easier to locate where a rogue value is introduced in these designs than to otherwise deduce the particular context that results in an erroneous assignment. This adds up to a much higher degree of referential transparency on other hand it is not an optimal approach in all situations.

# **ASK, "WHAT WOULD THE USER DO?" (YOU ARE NOT THE USER)**

in this chapter emphasize the understanding of the user’s perspective and what feature you will do on your system where you need to observe the users of what they are doing to the system or similar to your system that you are developing not guessing what are the features that the users want and need. Users tend to exhibit similar patterns of behavior when interacting with the same system where you can improve the usability and user satisfaction.

# **AUTOMATE YOUR CODING STANDARD**

In this chapter it talks about how you should format or looks good in the eye to other collaborators, when you create a project, you need to do the same pattern it applies to all collaborators in this project. We all know that its coding standard is challenging and boring, in some cases other collaborators tried to rebel and not practicing the code format on your project.

The purpose of coding standard is for ensuring consistency in code formatting and structure, preventing individual developers from "owning" code, and avoiding common antipatterns that lead to bugs. They make it easier to work in a project and maintain development speed.

Also, the automation coding standard has some tools where you can scan or automate the pattern to look the same to the other format codes.