**LOUIE JAY A. CENIZA BSIT 4-1 FEBRUARY 19, 2024**

**(CHAPTER 5-10)**

# **BEAUTY IS IN SIMPLICITY**

In this chapter it talks how simplicity is essential for achieving key qualities in code such as readability, maintainability, and speed of development. it suggests that regardless of individual background, simplicity is foundational in defining what constitutes beautiful code. They argue that simple, it focused on methods where even though your system features are complex the individual parts need to be simple that leads to maintainable, testable code.

# **BEFORE YOU REFACTOR**

In this chapter it talks about think before you act, same as the act with prudence, you need to test and take notes of the strong and weak points of this current code don’t let your ego control you where you think that your idea is better than the current system and it says that you need to minimize the restructuring of the current code avoid restructuring the code, in some cases where you don’t understand you want to rewrite it that leads to some bugs and errors that are not in the existing system. If you’re restructuring you need to change it little by little and test it one by one don’t test it when you’re done with the whole thing it will result to massive bugs and errors in which it is so time consuming.

Also, don’t think that better technology tools and equipment are better outcomes it is not the reason that you need to restructure the code.

# **BEWARE THE SHARE**

In this chapter it talks about the potential pitfalls of indiscriminate code sharing in software development. it entails the experience of a new developer, eager to implement principles of code reuse they had learned, only to face criticism during a code review for introducing shared libraries. Through this experience, it highlights the importance of understanding the context in which code is shared. They emphasize that while code reuse can be beneficial in certain contexts, it can also introduce unintended dependencies and increase maintenance costs if not carefully managed. The chapter serves as a cautionary tale, reminding developers to critically evaluate the implications of code sharing decisions and to prioritize maintaining the independence and flexibility of different parts of the system.

# **THE BOY SCOUT RULE**

In this chapter it talks about applying the boy scout rule to your systems code, regardless of what module or who made the mess it can greatly affect the system code and improve it. if we just all follow the same rules it would get better as they implement. It says that it’s not a required to follow but maybe you can help a little by cleaning the mess, and yes, we all know that a mess in your code is not usually a programmer should do but sometimes caring on others should improve the collaboration of your team.

# **CHECK YOUR CODE FIRST BEFORE LOOKING TO BLAME OTHERS**

In this chapter it talks how you likely to blame others if there was a mistake on your system's code, before you blame others you need to question your assumption if its right and you can get help from others like explaining the current problem you’re facing and try to think or solve it again and again if it’s associated with your problem and fix it.

# **CHOOSE YOUR TOOLS WITH CARE**

In this chapter it talks how the importance of carefully choosing tools for your web development, consider the tools your using might affect the development which can lead to architectural mismatch between the application and tools that will make the code more complex.

Also, choosing tools can be frustrating cause it has many configurations and depending too much on a specific vendor can constrained your maintainability, performances, ability to evolve, price, etc., and when you’re using a free software even though its free, you may need to buy commercial support, which is not necessarily going to be cheap.