**LOUIE JAY A. CENIZA BSIT 4-1 FEBRUARY 20, 2024**

**(CHAPTER 11-20)**

# **CODE IN THE LANGUAGE OF THE DOMAIN**

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.

# **CODE IN THE LANGUAGE OF THE DOMAIN**

In this chapter it talks about how important you write the code that will reflect the concepts of the domain (where if you talk about code terms you usually say variables, functions, classes, and methods) where it improves the readability, and facilitates communication. among your collaborators instead of using technical jargon and complex data structures make it simple and clear like written in domain terms and clear relationship between entities.

# **CODE IS DESIGN**

In this chapter it talks about how the learning in the future learnings and construction became negligible, because of the fast-growing technology where it neglects the basic fundamentals of the flow of the design where it leads to the design quality's effectiveness.

# **CODE LAYOUT MATTERS**

In this chapter it talks about important layout or format of the code you have written where you can easily distinguish what are the unwanted parts and the parts that needs to be re-evaluate or re-written and, in this generation, there are many tools for lay outing your code designing your indentions and coloring your syntax to make it more appealing and you can visualize it if you have the same pattern on other code.

# **CODE REVIEWS**

In this chapter it talks how the importance of reviewing your code helps you to understand it more and improving your code quality and reducing the defects but suggests that the traditional approach of rigid, formal reviews led by architects or senior developers can be counterproductive. Instead, it advocates for a more collaborative and knowledge-sharing approach to code reviews.

Also, the primary goal of the code review should be to share knowledge, establish common coding guidelines, and promote collective code ownership. This can be achieved by involving team members in reviewing each other's code, rotating roles during review meetings, and assigning different areas of focus to reviewers. It says that making code reviews more enjoyable and engaging to motivate team members and ensure active participation. By keeping the review process constructive, involving both inexperienced and experienced team members, and maintaining coding conventions checked by tools, code reviews can become more effective and beneficial for the entire team.

# **CODING WITH REASON**

In this chapter it talks about challenges of reasoning about software correctness and proposes a middle path between formal proofs and informal reasoning: semi formally reasoning about correctness. instead of making it whole code where it has many functionalities in one function, dividing the code into short section or what we called functions is much more efficient and clearer making the code easy to read.

Also, it discusses that coding practices that facilitate reasoning and improve code quality. These practices include avoiding complex control flow structures like go to statements, minimizing the use of modifiable global variables, preferring immutable objects, making code readable and self-documenting, keeping functions short and focused, limiting the number of function parameters, and maintaining narrow interfaces to reduce communication between units of code.

By practicing and engaging in reasoned discussions about code, developers can improve their understanding of the codebase and enhance its quality, ultimately benefiting the entire team.

# **A COMMENT ON COMMENTS**

In this chapter it talks about how the is a good practice where you create a comment of outer and inline comments are important when you are implementing a code system, where it helps you easily distinguished which part and what is this part is for by making comment you can also save time finding them and make the other collaborators understand how this line of codes functions on your code system.

Also, you need to be clear and concise of what you are commenting, don’t make an excessive commenting which can clutter the code and make it harder to read. Instead, comments should provide relevant explanations to clarify the code without obscuring it.

# **COMMENT ONLY WHAT THE CODE CANNOT SAY**

In this chapter it talks about how important a thoughtful and purposeful commenting in your code system, it gives you clarity and concise explanation to the code. Comments should add genuine value to the codebase, providing insights or information that the code itself cannot convey. Comments that are incorrect or redundant can be more harmful than helpful, as they can distract programmers and lead to misunderstanding. Therefore, comments should be treated with the same care and scrutiny as code.

Also, you should only use comments when necessary to convey information that cannot be expressed through code alone.

# **CONTINUOUS LEARNING**

In this chapter it talks about the importance of continuous learning due to the unpredictable evolving of our technology, where you will be left behind if you don’t go with the flow of what are the trends in our industry, you need to continue learning so that your knowledge still remains relevant and marketable.

Also, it provides a list of strategies where you can learn on your own because you can only educate yourself, not with the help of anyone else so explore new technology and gain hands on experience through practice, implementing and learning beyond technical skills such as understanding the business domains and improving your productivity and quality.

# **CONVENIENCE IS NOT -ILITY**

In this chapter it talks about how importance of designing a good APIs where it needs to prioritize efficiency, consistency, and elegance over a flawed approached known as the argument of convenience where it involves bundling multiple functionalities or options into a single API method where it is more convenient for developers that often leads to decreased code readability and usability and can be result in confusion and ambiguity for those who are using the API.

Instead, emphasizing the importance of providing a well-thought-out vocabulary that allows for expressive and easy-to-understand code that that prioritizes clarity, expressiveness, and usability, even if it requires more time and effort to develop.

# **DEPLOY EARLY AND OFTEN**

In this chapter it talks about importance of prioritizing deployment and installation processes early in a project rather than leaving them as an afterthought. Deploying the application can help you to understand the users and your application better and refine and evolve it throughout the development cycle and allows you for easier debugging and identification than creating and assumptions in the code.

It tells us that deployment and development process can be treated at the same level of importance when developing an application, by prioritizing deployment early and continuously refining your application it can be more successful in delivering the process.