**LOUIE JAY A. CENIZA BSIT 4-1 MARCH 5, 2024**

**(CHAPTER 71-80)**

# **READ THE HUMANITIES**

In this chapter it talks about the importance of software developers engaging with the humanities, such as literature, history, philosophy, and the arts. Also, integrating humanistic perspectives into software development education and practice.

By embracing interdisciplinary learning and drawing on insights from the humanities, developers can create technology that not only functions effectively but also reflect a deeper understanding of human values, experiences, and aspirations.

# **REINVENT THE WHEEL OFTEN**

In this chapter it talks about the importance of reinventing the wheel, or creating software components from scratch that already exist, is valuable for developers to gain a deeper understanding of fundamental concepts and core software implementations. Also, relying solely on existing code can limit a developer's understanding of how things work beneath the surface. By reinventing components like memory managers, data structures, or user interface elements, developers can gain intimate knowledge of their inner workings and improve their overall skills.

# **RESIST THE TEMPTATION OF THE SINGLETON PATTERN**

In this chapter it warns us against the overuse of the Singleton design pattern in software development. The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance.

Instead of relying on Singletons, use dependency injection and inversion of control principles. By passing dependencies explicitly to classes rather than accessing them globally, developers can improve the flexibility, testability, and maintainability of their code.

# **THE ROAD TO PERFORMANCE IS LITTERED WITH DIRTY CODE BOMBS**

In this chapter it talks about the importance of clean code in achieving high performance in software applications. Dirty code, which is poorly structured, inefficient, or difficult to understand, can severely impact performance and scalability.

By maintaining a focus on clean, efficient code, developers can avoid the pitfalls that lead to performance problems and ensure that their applications deliver optimal performance for users.

# **SIMPLICITY COMES FROM REDUCTION**

In this chapter it talks about the value of simplicity in software design and development. Simplicity is not just about having fewer lines of code or fewer features, but rather about reducing complexity to its essential elements while maintaining functionality and usability.

By embracing simplicity and reducing unnecessary complexity, developers can create software that is easier to understand, maintain, and extend over time.

# **THE SINGLE RESPONSIBILITY PRINCIPLE**

In this chapter it talks about the fundamental principle of software design aimed at improving code quality and maintainability. The principle states that a class should have only one reason to change, or in other words, it should have only one responsibility. the Single Responsibility Principle promotes code that is easier to understand, maintain, and extend by encouraging developers to design classes, functions, and modules with a single, well-defined purpose. This results in more modular, flexible, and resilient software systems.

# **START FROM YES**

In this chapter it encourages us in a positive and open-minded approach to problem-solving and collaboration. Rather than immediately dismissing ideas or proposals, "Start from Yes" suggests beginning with an attitude of acceptance and willingness to explore possibilities.

By embracing this mindset, teams can overcome obstacles more effectively, build stronger relationships, and achieve greater success in their endeavors.

# **STEP BACK AND AUTOMATE, AUTOMATE, AUTOMATE**

In this chapter it talks about the strategic use of automation in software development to enhance productivity, reliability, and efficiency. Also stating the importance of stepping back from manual tasks and investing time in automating repetitive processes.

Embracing automation is essential for modern software development teams to remain competitive and deliver value efficiently. By stepping back and prioritizing automation, developers can focus their time and energy on innovation and problem-solving, ultimately driving business success.

# **TAKE ADVANTAGE OF CODE ANALYSIS TOOLS**

In this chapter it talks about the proactive use of code analysis tools to improve software quality, maintainability, and reliability. It has benefits of leveraging automated analysis tools to identify potential issues and enforce coding standards throughout the development process.

It encourages developers to embrace code analysis tools as valuable assets in their toolkit. By taking advantage of these tools, teams can elevate their coding practices, reduce technical debt, and deliver higher-quality software products to their users.

# **TEST FOR REQUIRED BEHAVIOR, NOT INCIDENTAL BEHAVIOR**

In this chapter it talks about the importance of focusing testing efforts on verifying the essential behavior of software systems rather than incidental or implementation-specific details, tests should prioritize the validation of the intended functionality and requirements of the system, rather than becoming overly reliant on specific implementation details that may change over time.

You’ll need to develop your mindset that testing, prioritizes the validation of required behavior while remaining flexible to changes in the implementation. By focusing on essential functionality and maintaining test independence, developers can build robust test suites that provide meaningful validation of software systems.