Nicole P. Satiembre **March 4, 2024**

Cebu Technological University-Main Campus

Clean Code

Chapter I

1. Clean Code

There Will Be a Code. People often argue that programmers won't be needed anymore because they will be replaced with automated codes generated through the inputs of project requirements, needs, and specifications. Modern AI applications can generate fast code just by writing a few prompts and written code will become a thing of the past. However, the opposite is true. The elimination of code is implausible since code encapsulates the intricacies of requirements. There reaches a point where these details cannot be overlooked or simplified; they must be explicitly defined. Crafting requirements in such precision that a machine can carry them out constitutes programming. In essence, such a detailed specification is, in fact, code. This will never happen. Only humans have the intuition and creativity capable of creating successful systems from the vague feelings of their customers.

Bad Code. The experience of a company from the late 80s is recounted, where initially successful software suffered due to neglected code quality. The author discusses the common challenge of dealing with bad code, referring to it as "wading" through a difficult terrain of tangled and senseless programming. The author acknowledges the universal tendency to postpone code cleanup for another day, highlighting the trap of thinking "later equals never" as coined by LeBlanc's law. This passage serves as a reminder of the enduring importance of prioritizing clean and maintainable code in software development.

The Total Cost of Owning a Mess. As the mess builds, the productivity of the team continues to decrease. Within a year or two, teams that initially progressed rapidly on a project may experience a significant slowdown. Any alteration to the code results in the disruption of two or three other components. No adjustment is straightforward. Each addition or modification to the system necessitates a comprehensive understanding of the existing complexities, only leading to the introduction of more intricacies. As time elapses, the disorder accumulates to such an extent that it becomes unmanageable. The complexity becomes vast and insurmountable, making it impossible to tidy up the mess in any way.

The Grand Redesign in the Sky. The story teaches us that when a team rushes through coding and the code becomes messy, it can lead to big problems. Eventually, the team rebels and asks for a complete redo. Even though the managers may not want to spend resources on a new design, they agree because productivity is suffering. A special team starts fresh, but they need to match the old system's features and keep up with its changes. This process can take a really long time, and even after it's done, the new system might still need fixing. So, the lesson is, if you've been through a situation like this, you know that taking the time to keep your code clean is not just a good idea – it's crucial for your professional success.

Attitude. As experts in our field, we must uphold high professionalism in the work that we do. When encountering a messy code and it took more time than we expected for it to finish, we have lots of explanations for it. We complain that the requirements changed in ways that thwart the original design. We bemoaned the schedules that were too tight to do things right. We blather about stupid managers, intolerant customers, useless marketing types, and telephone sanitizers. We may want to play the blame game but the real fault is us. We are being unprofessional. The managers and marketers look to us for the information they need to make promises and commitments; and even when they don't look at us, we should not be shy about telling them what we think. We must be upfront about the possible consequences of the task we are assigned instead of being complicit. It is unprofessional for programmers to bend to the will of managers who don't understand the risks of making messes.

The Primal Conundrum. All developers already know that code messes slow them down yet the pressure of deadlines causes them to risk messes. True professionals know that the second part of the conundrum is wrong. You will not make the deadline by making the mess. Indeed, the mess will slow you down instantly and will force you to miss the deadline. The only way to make the deadline—the only way to go fast—is to keep the code as clean as possible at all times

The Art of Clean Code. We already know that writing messy code is a big impediment, thus, as much as possible we would want to write clean code. But we cannot write clean code if we don't know what clean code is like. writing clean code is not just about following rules but involves the disciplined application of numerous small techniques guided by a developed sense of "cleanliness," often referred to as "code-sense." While some individuals may possess this instinct naturally, others need to consciously cultivate it. Code-sense allows a programmer not only to distinguish between good and bad code but also provides a strategy for transforming messy code into clean, efficient code.

What is Clean Code?

Bjarne Stroustrup, inventor of C++ and author of The C++ Programming Language. Clean code is efficient and elegant. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Bad code tempts the mess to grow! When others change bad code, they tend to make it worse.

Grady Booch, author of Object Oriented Analysis and Design with Applications. Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control. Like a good novel, clean code should expose the tensions in the problem to be solved. Our code should be matter-of-fact as opposed to speculative. It should contain only what is necessary.

"Big" Dave Thomas, founder of OTI, godfather of the Eclipse strategy. Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways of doing one thing. Dave ties cleanliness to tests! Code, without tests, is not clean. No matter how elegant it is, no matter how readable and accessible, if it hath not tests, it is unclean.

Michael Feathers, author of Working Effectively with Legacy Code. Clean code is code that has been taken care of. Someone has taken the time to keep it simple and orderly. They have paid appropriate attention to details. They have cared.

Ron Jeffries, author of Extreme Programming Installed and Extreme Programming Adventures in C#. Simple rules of code: Runs all the tests; Contains no duplication; Expresses all the design ideas that are in the system; Minimizes the number of entities such as classes, methods, functions, and the like. To emphasize: no duplication, one thing, expressiveness, tiny abstractions. Everything is there.

Ward Cunningham, inventor of Wiki, inventor of Fit, and coinventor of eXtreme Programming. The motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code. Clean code is characterized by predictability and clarity. When reading a routine, if the code aligns with your expectations and is easily understandable, it indicates cleanliness. Taking it a step further, code can be deemed beautiful when it not only meets expectations but also appears as if the programming language was tailor-made for the specific problem at hand. In essence, the measure of clean code goes beyond mere functionality and extends to the elegance with which it aligns with the inherent nature of the programming language and the problem it aims to solve.