Nicole P. Satiembre **March 27, 2024**

Cebu Technological University-Main Campus

Clean Code

Chapter 12-Emergence

Getting Clean via Emergent Design.

A design is considered simple if it follows these rules:

- Rules all the tests
- Contain no duplication
- Expresses the intent of the programmer
- Minimizes the number of classes and methods

Simple Design Rule 1: Runs All the Tests

A good system design must act as intended, to verify this, it must be testable. Systems that aren't testable aren't verifiable. Arguably, a system that cannot be verified should never be deployed. Tight coupling makes it difficult to write tests. So, similarly, the more tests we write, the more we use principles like DIP and tools like dependency injection, interfaces, and abstraction to minimize coupling. Our designs improve even more.

Simple Design Rules 2-4: Refactoring

We can keep our codes clean by incrementally refactoring our codes. For each few lines of code, we add, we pause and reflect on the new design. We clean it up and don't forget to run tests to demonstrate that we haven't broken anything. We can increase cohesion, decrease coupling, separate concerns, modularize system

concerns, shrink our functions and classes, choose better names, and so on. This is also where we apply the final three rules of simple design: Eliminate duplication, ensure expressiveness, and minimize the number of classes and methods.

No Duplication

Duplication is the primary enemy of a well-designed system. It represents additional work, additional risk, and additional unnecessary complexity. Duplication manifests itself in many forms. Lines of code that look exactly alike and duplication of implementation.

Expressive

Most of us have worked with complex codes and have produced some ourselves. We may be able to understand our code at the time of writing it but other maintainers of the code aren't going to have so deep an understanding. The majority of the cost of a software project is in long-term maintenance. In order to minimize the potential for defects as we introduce change, we must be able to understand what a system does. This is why as much as we can, we should clearly express the intent of our codes.

Minimal Classes and Methods

In an effort to make our classes and methods small, we might create too many tiny classes and methods. So this rule suggests that we also keep our function and class counts low. Our goal is to keep our overall system small while we are also keeping our functions and classes small. Remember, however, that this rule is the lowest priority of the four rules of Simple Design. So, although it's important to keep class and function count low, it's more important to have tests, eliminate duplication, and express yourself.