Mohammadmahdi Farrokhy

Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 10: Clarity

Simple Design Rule

An important point for having clean code is to have clarity in the design. Based on Beck's opinion, design is simple if it

- passes all tests.
- has no duplication.
- is expressive.
- has minimal classes and methods.

These rules are prioritized in order of importance.

Rule #1

- Passes all tests.

We start by designing a system that works as expected. However, even if a system functions well theoretically, it's all in vain if we can't easily verify it. A testable system should be thoroughly tested and consistently pass those tests.

Increasing the number of tests gets us closer to testable components, leading to the required confidence for building better system design. It also opens up the opportunity to apply principles like **S.O.L.I.D** and **design** patterns to our code.

Refactoring

To change code's structure, without changing its logic. Afterward, we run all the tests again to ensure our modification did not disrupt the logic.

Rule #2

- No Duplication

Duplication means additional effort, additional risk and unnecessary complexity. Different forms of code duplication are:

  - Lines with exact same code
  - Lines with similar codes, but different parameters

Rule #2
BAD CODE

```java
public void client() {
    JButton button1 = new JButton("1");
    button1.setBounds(0, 0, 50, 60);
    button1.setBackground(Color.white);
    button1.setBorder((BorderFactory.createLineBorder(Color.white)));

    JButton button2 = new JButton("2");
    button2.setBounds(100, 100, 50, 60);
    button2.setBackground(Color.white);
    button2.setBorder((BorderFactory.createLineBorder(Color.white)));

    JButton button3 = new JButton("3");
    button3.setBounds(200, 200, 50, 60);
    button3.setBackground(Color.white);
    button3.setBorder((BorderFactory.createLineBorder(Color.white)));
}
```

Rule #2
GOOD CODE

```java
private static final int BUTTON_WIDTH = 50;
private static final int BUTTON_HEIGHT = 60;
private static final Color BORDER_COLOR = Color.white;
private static final Color BACK_COLOR = Color.white;

public static void client() {
    JButton button1 = createButton("1", 0, 0);
    JButton button2 = createButton("2", 100, 100);
    JButton button3 = createButton("3", 200, 200);
}

private static JButton createButton(String text, int x, int y) {
    JButton button = new JButton(text);
    button.setBounds(x, y, BUTTON_WIDTH, BUTTON_HEIGHT);
    button.setBackground(BACK_COLOR);
    button.setBorder(BorderFactory.createLineBorder(BORDER_COLOR));
    return button;
}
```

Rule #3

- Expressive

The major cost of a software project is attributed to its long-term maintenance. To minimize the cost and potential modifications issues, we have to understand system's behavior and functionality. The code should clearly reflect the author's intentions. A well-written code can be easily understood, leading to significant reduction of onboarding time and maintenance cost.

Rule #4

- Minimal Classes And Methods

The goal is to keep the system compact at its highest level. So we need minimum number of small classes and methods.