Pola Desain Perangkat Lunak

[Week] 13 – Criticisms of Design Pattern & Antipattern
Prepared by: Tifanny Nabarian

Criticisms of Design Pattern

Overview

- In this chapter, I present some of the criticisms of design patterns. Reading about the criticisms can offer real value.
- If you think critically about patterns before you design your software, you can predict your “**return on investment**” to some degree.
- Design patterns basically help you benefit from another people’s experience. This is often called ***experience reuse***.

Criticism #1

- The way you write program in today's world is different and the facilities that you have nowadays are much more compared to old days of programming.
- So, when you extract patterns based on some old practices, you ***basically show additional respect to them***.

Criticism #2

- Design patterns are based on some of the key principles, and one of them is to *identify the code that may vary and then separate it from rest of the code*. It sounds very good from theoretical perspective.
- But in real world implementations, who guarantees you that your judgment is perfect? **Software industry always changes**, and it needs to adapt with **new requirements/demands**.

Criticism #3

- The pattern that is giving you the satisfactory results today, can be a **big burden to you in the near future** due to the **“continuous changes”** in the software industry.

Criticism #4

- Designing a software is basically an art. And there is **no definition or criteria for best art**.

Criticism #5

- **Design patterns give you the idea** but not the implementations (like libraries or frameworks).
- **Each human mind is unique**. So, each engineer may have his/her own preferences for implementing a similar concept, and that can create chaos in a team.

Criticism #6

- At the end, design patterns basically help **you to get benefit from others experience**. You are getting their thoughts, you come to know how they encountered the challenges, **how they tried to adapt new behaviors in their systems**, and so forth.
- **But you start with the assumption that a beginner or relatively less-experienced person** cannot solve a problem better than his/her seniors.
- In some specific occasions, **a relatively less experienced person can have a better vision than his seniors**, and he can prove himself more effective in the future.

Antipattern

Overview

- The original idea of design patterns came from building architect Christopher Alexander. He shared his ideas for the construction of buildings within well-planned towns.
- Gradually, **these concepts entered into software development** and they gained popularity through the leading-edge software developers like Ward Cunningham and Kent Beck.
- In 1994, the idea of design patterns **entered mainstream object-oriented software development** through an industry conference on design patterns, known as **Pattern Languages of Program Design (PLoP).**

Overview

- Undoubtedly, **these great design patterns helped (and are still helping) programmers to develop the high-quality software**. But *people started noticing the negative impacts also*.
- A common example is that many **developers wanted to show their expertise without the true evaluation** or the consequences of these patterns in their specific domains. As an obvious side effect, **patterns were implanted in the wrong context, which produced low-quality software**, and ultimately caused big penalties to the developers and their companies.

Overview

- So, the **software industry needed to focus on the negative consequences of these mistakes, and eventually, the idea of antipatterns evolved**. Many experts started contributing to this field, but the first well-formed model came through Michael Akroyd's presentation, "**AntiPatterns: Vaccinations against Object Misuse**." It was the antithesis of the GoF's design patterns.
- **AntiPatterns alert you about common mistakes that lead to a bad solution**. Knowing them helps you take precautionary measures. The proverb "**prevention is better than cure**" very much fits in this context.

AntiPattern

- ***Over Use of Patterns***: Developers may try to use patterns at any cost, regardless of whether it is appropriate or not
- ***God Class***: A big object that tries to control almost everything with many unrelated methods. An inappropriate use of the mediator pattern may end up with this antipattern.
- ***Golden Hammer***: **Mr. X believes that technology T is always best**. So, if he needs to develop a new system **(that demands new learning), he still prefers T, even if it is inappropriate**. He thinks, "I do not need to learn any more technology if I can somehow manage it with T."

AntiPattern

- ***Shoot the Messenger***: You are already under pressure and the program deadline is approaching. There is a smart tester who always finds typical defects that are hard to fix. So, at this stage, you do not want to involve him because he will find more defects and the deadline may be missed.
- ***Swiss Army Knife***: Demand a product that can serve the customer's every need. Or make a drug that cures all illnesses. Or design software that serves a wide range of customers with varying needs. It does not matter how complex the interface is.

Thank You!