Louis Potterie M2 ITF

TP#6: February 09 th – Design Patterns, Refactoring, Code Smells

Exercise 1 :

1. What is clean code? What is the role played by refactoring one's code?

Clean code is code easily readable, easy to understand and to maintain. To write clean code will increase the scalability of the program. The role to refactoring code is to improve the quality of the code and decrease the technical debt.

2. Do you think you can "over-refactor" and do too much? How?

"Over-refactor" is for me like over-optimization, with an over-optimization that could recreate complexity not necessary.

3. What is a code smell? Why should you bother?

Code smells are indicators of problems that can be process during refactoring. We should take care of code smells because they are hints to deeper problems or bad coding practices.

4. Can you identify a few code smells in the original GuildedRose?

- Duplicate code:
  - Ex: the increase/decrease quality function is duplicate for each if statement in the Update Quality function.
- Switch Statements:
  - Ex: Update quality function contains a complex sequence of if statements, Refactoring.guru advises to think about polymorphism
- Lazy class:
  - Ex: Item.cs contains only attributes and the basic methods for accessing them (getters/setters)

5. What are some refactoring techniques you could have used in the GuildedRose?

- Extract method:
  - Can be use for Duplicate code and extract this duplicate code into a dissociate function
- Replace Conditional with Polymorphism:
  - Can be apply on to simplify the complex if-statements
- Replace List with Object:
  - We can replace the List representing the inventory into an Object Inventory
- Replace Nested Conditional with Guard Clauses
  - Eliminate nested conditional of Update quality function

Exercise 2: Now get to the other part of this website, under “Design Patterns”. You’ll discover incredible mechanism used to build robust software that are used through-out the world.

1. In simple terms, and a few sentences at most, what’s a design pattern?

Since the beginning of IT, lot of developers has solved problems and issues. A design pattern is a template solution to a common IT problem. The design patterns are tested and approved for problem already met.

a. When should you use one?

When we encounter a common problem, we should use design patterns because we already know that is a good way to resolve it. And also, it will provide a solution understood by other developers.

b. When shouldn’t you?

We should not use design patterns when a simple algorithm resolves the problem, because use a design pattern in a simple situation will complexify for nothing your code.

2. Why is this a good idea? Think about scalability for instance

With a builder pattern that easy to implement different object configuration and allow you to be scalable with different characteristics that you want to add to a same base product. We can have a personalized product with just adding a function add in the builder

3. Find an original idea to implement a decorator pattern (not the one from the website). Can you think of any limitations? No need to code here.

I think we can apply decorator patter to a car range offer. For example, we have a basic car called Csharpiscool, this basic car has 3 doors, no air conditioning, basic seat, 90ch motor. Chsharpiscool exists with 5 doors but with the same other characteristics. We have also an upgrade version of Chsharpiscool with a motor of 120 ch and air conditioning but still with the same chassis and other elements.

Example :