Agile Software Development (TCS 855)

Unit-III Agile Software Design and PROGRAMMING

Refactoring Techniques

Prof.(Dr.) Santosh Kumar

Department of Computer Science and Engineering

Graphic Era Deemed to be University, Dehradun

Refactoring Techniques

1. Composing Methods
2. Moving Features between Objects
3. Organizing Data
4. Simplifying Conditional Expressions
5. Simplifying Method Calls
6. Dealing with Generalization

1. Composing Methods

- Much of refactoring is devoted to correctly composing methods. In most cases, excessively long methods are the root of all evil. The vagaries of code inside these methods conceal the execution logic and make the method extremely hard to understand—and even harder to change.
- The refactoring techniques in this group streamline methods, remove code duplication, and pave the way for future improvements.

Extract Method

- **Problem:** You have a code fragment that can be grouped together.
- **Solution:** Move this code to a separate new method (or function) and replace the old code with a call to the method.

Inline Method

- **Problem:** When a method body is more obvious than the method itself, use this technique.
- **Solution:** Replace calls to the method with the method's content and delete the method itself.

Extract Variable

- **Problem:** You have an expression that's hard to understand.
- **Solution:** Place the result of the expression or its parts in separate variables that are self-explanatory.

Moving Features between Objects

- Even if you have distributed functionality among different classes in a less-than-perfect way, there is still hope.
- These refactoring techniques show how to safely move functionality between classes, create new classes, and hide implementation details from public access.

e.g. Move Method

Problem: A method is used more in another class than in its own class.

Solution: Create a new method in the class that uses the method the most, then move code from the old method to there. Turn the code of the original method into a reference to the new method in the other class or else remove it entirely.

Similarly others….

Move Method
Move Field
Extract Class
Inline Class

Hide Delegate
Remove Middle Man

Introduce Foreign Method
Introduce Local Extension

3. Organizing Data

- These refactoring techniques help with data handling, replacing primitives with rich class functionality. Another important result is untangling of class associations, which makes classes more portable and reusable.

e.g. **Change Value to Reference**

Problem: If you have many identical instances of a single class that you need to replace with a single object..

Solution: Convert the identical objects to a single reference object.

Similarly others….

- Change Value to Reference
- Change Reference to Value
- Duplicate Observed Data
- Self Encapsulate Field
- Replace Data Value with Object
- Replace Array with Object

4. Simplifying Conditional Expressions

- Conditionals tend to get more and more complicated in their logic over time, and there are yet more techniques to combat this as well.

e.g. Consolidate Conditional Expression

Problem: You have multiple conditionals that lead to the same result or action

Solution: Consolidate all these conditionals in a single expression.

Similarly others….

- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Decompose Conditional

5. Simplifying Method Calls

- These techniques make method calls simpler and easier to understand. This, in turn, simplifies the interfaces for interaction between classes**.**

e.g. Add parameters

Problem: A method doesn't have enough data to perform certain actions**.**

Solution: Create a new parameter to pass the necessary data.

Similarly others….

Add Parameter
Remove Parameter
Rename Method
Separate Query from Modifier
Parameterize Method

6. Dealing with Generalization

- Abstraction has its own group of refactoring techniques, primarily associated with moving functionality along the class inheritance hierarchy, creating new classes and interfaces, and replacing inheritance with delegation and vice versa.

e.g. Pull Up Field

Problem: Two classes have the same field.

Solution: Remove the field from subclasses and move it to the superclass.

Similarly others….

Pull Up Field
Pull Up Method
Pull Up Constructor Body
Push Down Field
Push Down Method