24 Refactoring

Contents

Qué es la refactorización

*Martin Fowler defines as "a change made to the internal structure of the software to make it **easier to understand** and **cheaper to modify** without changing its observable **behavior**".*

En nuestro caso particular:

- *A big refactoring is a recipe for disaster.* —Kent Beck :slack_call: @Elmo

CHECKLIST: Reasons to Refactor

pag. 609 Code Complete.

1. Code is duplicated
2. A routine is too long
3. A loop is too long or too deeply nested
4. A class has poor cohesion
5. A class interface does not provide a consistent level of abstraction
6. A parameter list has too many parameters
7. Changes within a class tend to be compartmentalized
8. Changes require parallel modifications to multiple classes

9. Inheritance hierarchies have to be modified in parallel
10. Related data items that are used together are not organized into classes
11. A routine uses more features of another class than of its own class
12. A primitive data type is overloaded
13. A class doesn't do very much
14. A chain of routines passes tramp data
15. A middle man object isn't doing anything
16. One class is overly intimate with another
17. A routine has a poor name
18. Data members are public
19. A subclass uses only a small percentage of its parents' routines
20. Comments are used to explain difficult code
21. Global variables are used
22. A routine uses setup code before a routine call or takedown code after a routine call
23. A program contains code that seems like it might be needed someday

CHECKLIST: Summary of Refactorings

pag. 616 Code Complete

Data Level Refactorings

1. Replace a magic number with a named constant.
2. Rename a variable with a clearer or more informative name.
3. Move an expression inline.
4. Replace an expression with a routine.
5. Introduce an intermediate variable.
6. Convert a multi-use variable to a multiple single-use variables.
7. Use a local variable for local purposes rather than a parameter.
8. Convert a data primitive to a class.
9. Convert a set of type codes to a class.
10. Convert a set of type codes to a class with subclasses.
11. Change an array to an object.
12. Encapsulate a collection.
13. Replace a traditional record with a data class.

Statement Level Refactorings

1. Decompose a boolean expression.
2. Move a complex boolean expression into a well-named boolean function.

3. Consolidate fragments that are duplicated within different parts of a conditional.
4. Use break or return instead of a loop control variable.
5. Return as soon as you know the answer instead of assigning a return value within nested if-then-else statements.
6. Replace conditionals with polymorphism (especially repeated case statements).
7. Create and use null objects instead of testing for null values.

Routine Level Refactorings

1. Extract a routine.
2. Move a routine's code inline.
3. Convert a long routine to a class.
4. Substitute a simple algorithm for a complex algorithm.
5. Add a parameter.
6. Remove a parameter.
7. Separate query operations from modification operations.
8. Combine similar routines by parameterizing them.
9. Separate routines whose behavior depends on parameters passed in.
10. Pass a whole object rather than specific fields.
11. Pass specific fields rather than a whole object.
12. Encapsulate downcasting.

Class Implementation Refactorings

1. Change value objects to reference objects.
2. Change reference objects to value objects.
3. Replace virtual routines with data initialization.
4. Change member routine or data placement.
5. Extract specialized code into a subclass.
6. Combine similar code into a superclass.

Class Interface Refactorings

1. Move a routine to another class.
2. Convert one class to two.
3. Eliminate a class.
4. Hide a delegate.
5. Replace inheritance with delegation.
6. Replace delegation with inheritance.
7. Remove a middle man.

8. Introduce a foreign routine.
9. Introduce a class extension.
10. Encapsulate an exposed member variable.
11. Remove Set() routines for fields that cannot be changed.
12. Hide routines that are not intended to be used outside the class.
13. Encapsulate unused routines.
14. Collapse a superclass and subclass if their implementations are very similar.

System Level Refactorings

1. Duplicate data you can't control.
2. Change unidirectional class association to bidirectional class association.
3. Change bidirectional class association to unidirectional class association.
4. Provide a factory routine rather than a simple constructor.
5. Replace error codes with exceptions or vice versa.

CHECKLIST: Refactoring Safely

pag. 630 Code Complete
w

1. Is each change part of a systematic change strategy?
2. Did you save the code you started with before beginning refactoring?
3. Are you keeping each refactoring small?
4. Are you doing refactorings one at a time?
5. Have you made a list of steps you intend to take during your refactoring?
6. Do you have a parking lot so that you can remember ideas that occur to you mid-refactoring?
7. Have you retested after each refactoring?
8. Have changes been reviewed if they are complicated or if they affect mission-critical code?
9. Have you considered the riskiness of the specific refactoring, and adjusted your approach accordingly?
10. Does the change enhance the program's internal quality rather than degrading it?