Code smells and refactoring

CO5225 – Andrew Muncey

Contents

- Definition
- Code smells
  - Duplicated code
  - Long method
  - Large class
  - Long parameter list
  - Temporary fields
  - Data class

- Refactoring
  - Composing Methods
    - Extract method
    - Inline method
    - Inline Temp
    - Introducing explaining variable
  - Organising Data
    - Encapsulate Field
    - Encapsulate Collection
    - Replace type code with class
  - Making Method calls simpler
    - Remove setting method
    - Hide method
    - Rename method
    - Add Parameter
    - Remove Parameter

Code Smell

- “Symptoms of poor design and implementation choices” (Tufano et al., 2015, p.403)
- “A surface indication that usually corresponds to a deeper problem in the system” (Fowler, 2006)

Refactoring

- “Changing a software system in such a way that it **does not alter the external behavior** of the code yet improves its internal structure. It is a disciplined way to clean up the code that minimizes the chances of introducing bugs. In essence when you refactor you are **improving the design of the code after it has been written**” (Fowler, 1999, p.xvi)

Code Smell – Duplicated code

- The same structure of code appears more than once
  - in the same function
  - In two different methods in the same class
  - In subclasses that share a parent

Code Smell – Long method

- The longer function is, the harder it is likely to be to understand what it does and how it does it
- It’s also likely to be harder to test

Code Smell – Large class

- Class tries to do too much
  - Common culprit are GUI classes
- Possible symptom: too many instance variables

- Remember classes should adhere to Single Responsibility Principle

Code Smell – Long Parameter List

- Passing too many parameters to a method
- E.g. Redundant parameters that are instance variables

Code Smell – Temporary Field

- An instance variable is only set in certain circumstances
  - Whilst a method runs
  - Because only certain instance of the class use it

Code Smell – Data class

- Classes with fields, getters and setters and nothing else
  - Setters might not be needed
  - Collections might be fully modifiable from outside the class

Refactoring - Extract Method

- Solution
  - Extract code into a clearly named method that describes its purpose
- Possible issues
  - Extracted code modifies more than one variable from original method

Refactoring – Introduce Explaining Variable

- You have a complicated expression
  - Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose
- Solution
  - Declare a final temporary variable, set the result to part of the complex expression
  - Replace that part of the expression with the temporary variable
  - Repeat as needed

Refactoring – Inline Method

- A method’s body is just as clear as [or clearer than] its name
- Solution
  - Put the method’s body into the body of its callers and remove the method

Refactoring – Inline Temp

- You have a temporary variable that is assigned once with a simple expression, and it is getting in the way of other refactorings
  - Replace all references to that temporary variable with the expression
- Solution
  - Check no side effects of the assignment
  - Make it final (val) and compile to check it really is only assigned once
  - Replacement references to temporary variable with expression

Refactoring – Encapsulate Field

- There is a public field
  - Make It private and provide accessors
- Solution
  - Add getters and setters
  - Replace usages of the field with calls to the getter/setter as appropriate
  - Make the field private

Refactoring – Encapsulate Collection

- A method returns a collection
  - Make it return a read only view and provide add/remove methods
- Solution
  - Create methods to add and remove items from the collection
  - Initialise the instance variable to an empty collection
  - The existing method that returns the entire collection should return an unmodifiable collection (rename if required)
  - Find methods that use the setter and replace with calls to the new add items method
  - Review the use of getter methods and extract / move code as required (e.g. to object with the collection)

Refactoring – Replace Type code with class

- A class uses a numeric type code (e.g. ints) to represent types
  - Replace the number with a new class
- Solution
  - Create a class for the type code (e.g. an enum)
    - Use the existing int values for the underlying enum value types
  - Review all uses of the existing type code and switch to use the enum
  - Remove the original implementation of the type codes

Refactoring – Remove setting method

- A field should be set at creation time and never altered
  - Remove any setting method for that field
- Solution
  - Check the setter is only called in the constructor
  - Modify the constructor to access the instance variable directly
  - Compile and test
  - Remove the setting method and make the field final
  - Compile

Refactoring – Hide method

- A method is not used by any other class
  - Make the method private
- Solution
  - Consider each method and make it private if you can
  - Compile

Refactoring – Rename method

- The name of a method does not reveal its purpose
  - Change the name of the method
- Solution
  - Check to see if the method is inherited
  - Rename the method in each case to accurately describe its purpose

Refactoring — Add Parameter

- A method needs more information from its caller.
  - Add a parameter for an object that can pass on this information.
  - Don’t if the information can be obtained from another parameter
- Solution
  - Check to see if the method is inherited
  - Make a new method with the new parameter and copy over the body
  - Compile
  - Change body of old method to call the new method
  - Compile and test
  - Modify all references to the old method to call the new method;
  - Compile and test after each reference is changed
  - Delete the old method

Refactoring — Remove Parameter

- A parameter is no longer used by the method body.
  - *Remove it.*
- Solution
  - Check to see if the method is inherited
  - Make a new method without the old parameter and copy over the body
  - Compile
  - Change body of old method to call the new method
  - Compile and test
  - Modify all references to the old method to call the new method;
  - Compile and test after each reference is changed
  - Delete the old method

Unit testing and refactoring

- If you generate unit tests as you develop a project and then need to refactor, running the unit tests can help ensure that your refactoring has been carried out correctly.
- In some cases, unit tests will need to be modified or new ones created.

Reading

- Tufano, Michele; Palomba, Fabio; Bavota, Gabriele; Oliveto, Rocco; Di Penta, Massimiliano; De Lucia, Andrea; Poshyvanyk, Denys (2015). *When and Why Your Code Starts to Smell Bad*. IEEE/ACM 37th IEEE International Conference on Software Engineering. pp. 403–414. ACM
- Fowler, M. 1999. *Refactoring: Improving the design of existing code*. Addison Wesley: Crawfordsville, IN.