

Refactoring, Code Smells, Coupling and Cohesion



Dario Campagna

Head of Research





Code Smells

Symptoms of a problem



Code Smells

A code smell is a surface indication that usually corresponds to a deeper problem in the system.

- Quick to spot
- Provide feedback on our decisions
- Don’t always indicate a problem worth solving



Categories of code smells

Bloaters
- Long Method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps

Object-orientation abusers
- Switch Statements
- Temporary Fields
- Refused Bequest
- Alternative Classes with Different Interfaces

Couplers
- Feature Envy
- Inappropriate Intimacy
- Message Chains
- Middle Man

Change preventers
- Divergent Change
- Shotgun Surgery
- Parallel Inheritance Hierarchies

Dispensables
- Lazy Class
- Data Class
- Duplicated Code
- Dead Code
- Speculative Generality
- Comments

Have a look at https://refactoring.guru/refactoring/smells.



Primitive Obsession

Use of primitive types instead of small objects for simple tasks.

- Replace data value with object
- Replace type code with class
- Replace array with object
- …

```
1  package it.esteco.pos;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  public class Sale {
7
8      private Display display;
9      private final Map<String, String> pricesByBarcode;
10
11     public Sale(Display display, HashMap<String, String> pricesByBarcode) {
12         this.display = display;
13         this.pricesByBarcode = pricesByBarcode;
14     }
15
16     public void onBarcode(String barcode) {
17         if ("".equals(barcode)) {
18             display.setText("Scanning error: empty barcode!");
19         } else{
20             if (pricesByBarcode.containsKey(barcode)) {
21                 display.setText(pricesByBarcode.get(barcode));
22             } else {
23                 display.setText("Product not found for " +
24                         barcode);
25             }
26         }
27     }
28 }
```



Feature Envy

A method accesses the data of another object more than its own data.

- Move method
- Extract method

```
1  public class Coordinate
2  {
3    public int X {get; set}
4    public int Y {get; set}
5  }
6
7  public class PositionUpdater
8  {
9    public Coordinate MoveUp(Coordinate coordinate)
10   {
11     return new Coordinate{X = coordinate.X, Y = coordinate.Y + 1};
12   }
13 }
```



Message Chains

A message chain occurs when a client requests another object, that object requests yet another one, and so on.

- Hide delegate
- Extract method
- Move method

```
master.getModelisable()
  .getDockablePanel()
    .getCustomizer()
      .getSaveItem()
       .setEnabled(Boolean.FALSE.booleanValue());
```



```
master.allowSavingOfCustomizations();
```



Comments

The best comment is a good name for a method or class.

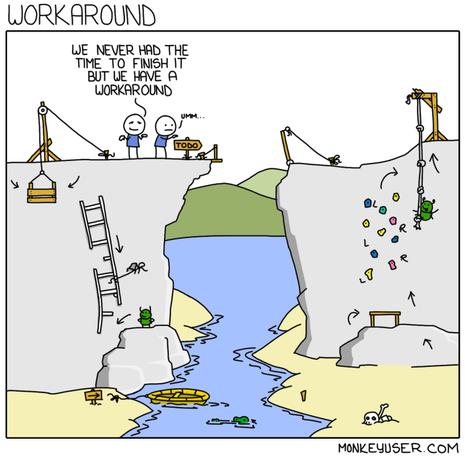
- Extract variable
- Extract method
- Rename method

```
public void Play(char symbol, int x, int y) throws Exception {
    //if first move
    if (_lastSymbol == ' ') {
        //if player is X
        if (symbol == 'O') {
            throw new Exception("Invalid first player");
        }
    }
    //if not first move but player repeated
    else if (symbol == _lastSymbol) {
        throw new Exception("Invalid next player");
    }
    //if not first move but play on an already played tile
    else if (_board.TileAt(x, y).Symbol != ' ') {
        throw new Exception("Invalid position");
    }

    // update game state
    _lastSymbol = symbol;
    _board.AddTileAt(symbol, x, y);
}
```



Comments can sometimes be useful



Define: Feature Tokens ($\mathcal{T}$) of codes in Table 3
Input: Program Codes ($\mathcal{C}$), $\mathcal{C} = \{c_1, c_2, c_3, \cdots, c_n\}$
Output: Tokenized SF ($\mathcal{TF}$) for all $\mathcal{C}$
for each program code $c_i \in \mathcal{C}$ **do**
    Refine Code $\mathcal{RC}_i \longleftarrow$ removeComments(c_i)
    Scan $\mathcal{RC}_i$ and **Select** $\mathcal{SF}$ in $\mathcal{RC}_i$ where $\forall \mathcal{SF} \in \mathcal{T}$
    Extract Selected Features ($\mathcal{EF}_i$) $\longleftarrow$ extractSelectedFeatures ($\mathcal{RC}_i$)
    for each extracted feature $f \in \mathcal{EF}_i$ **do**
        $\mathcal{TF}_{f \in \mathcal{EF}_i} \longleftarrow Token\ ID$
    end for
end for
Return $\mathcal{TF}_{c_i \in \mathcal{C}}$

Image from Watanobe, Y., Rahman, M.M., Amin, M.F.I. et al. Identifying algorithm in program code based on structural features using CNN classification model. Appl Intell 53, 12210–12236 (2023).

Explain why

To explain **why** something is being implemented in a particular way.

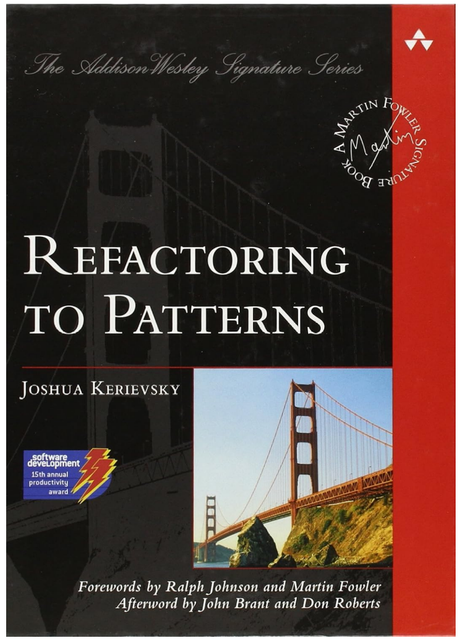
Citations

To **cite** an external knowledge source utilized in the implementation.

More Code Smells

Five additional code smells described in the book “Refactoring to Patterns”.

- Conditional Complexity
- Indecent Exposure
- Solution Sprawl
- Combinatorial Explosion
- Oddball Solution







Coupling and Cohesion

Metrics that (roughly) describe how easy it will be to change the behavior of some code.



Coupling

Measures the degree of interdependence between software components.

- Elements are coupled if a change in one forces a change in the other.
- We want to make changes in a component without impacting other components.
- We want coupling to be as low as possible, but not lower.



Cohesion

Measures how strongly related and focused the responsibilities of a software module are.

- Indicates a component’s functional strength and how much it focuses on a single point.
- Low cohesion results in behavior being scattered instead of existing in a single component.
- We want high cohesion.



LIFE Magazine (March 4, 1946)

Cohesion, coupling and code smells

- Divergent Change
- Feature Envy
- Inappropriate Intimacy
- Message Chains
- Middle Man
- Shotgun Surgery

High coupling

Indicators of possible high coupling.

- Data Class
- Lazy Class
- Middle Man
- Primitive Obsession
- Shotgun Surgery

Low cohesion

Indicators of possible low cohesion.



Smelly Tic Tac Toe

A TicTacToe implementation with quite a few code smells.

https://github.com/AgileTechPraxis/CodeSmells

- Start by identifying the smells.
- Then slowly refactor the code.

