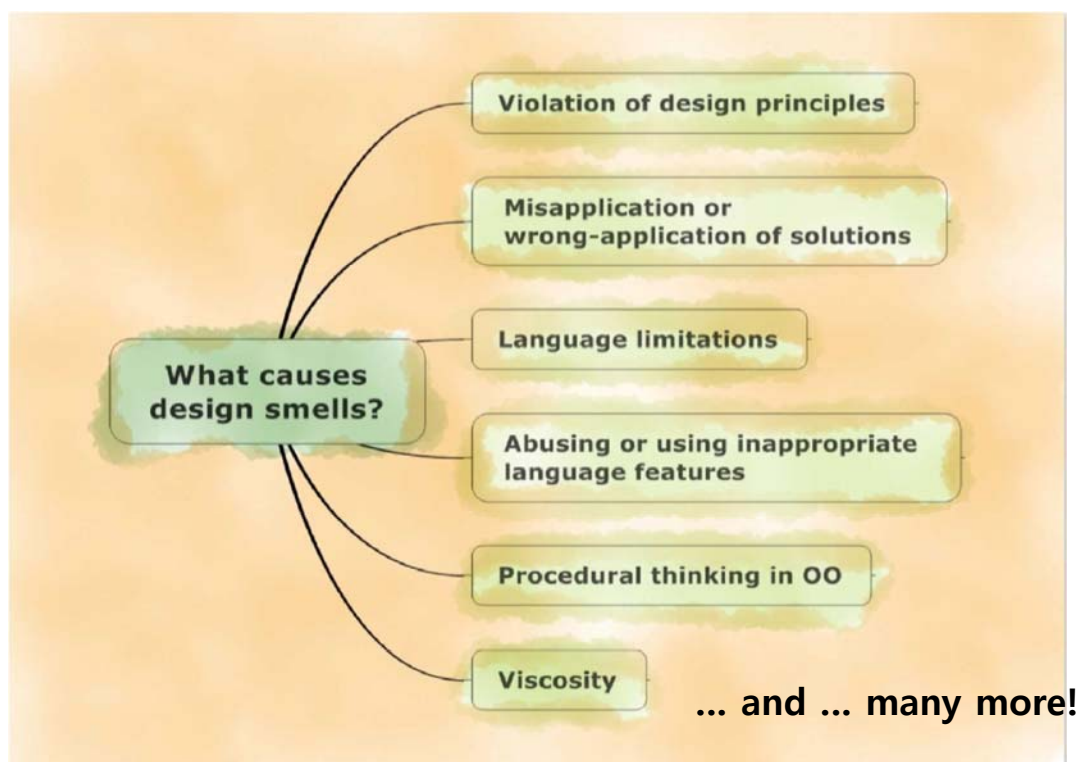


# DESIGN PRINCIPLE-BASED REFACTORING

1

## What Causes Design Smells?



2

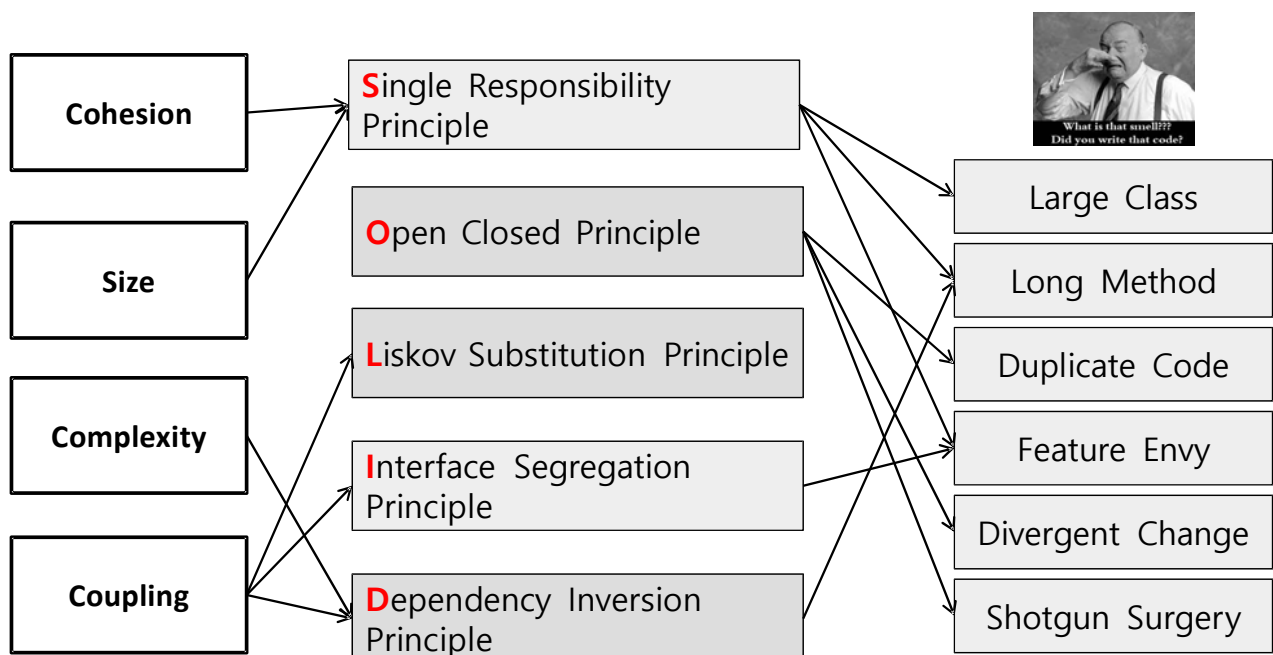
# Design Smells as Violations of Fundamental Principles

- What do smells indicate?
  - Violations of fundamental design principles
- Connection between smells and principles helps identify *cause of the smell* and *potential refactoring* as well.

3

## Many Concepts

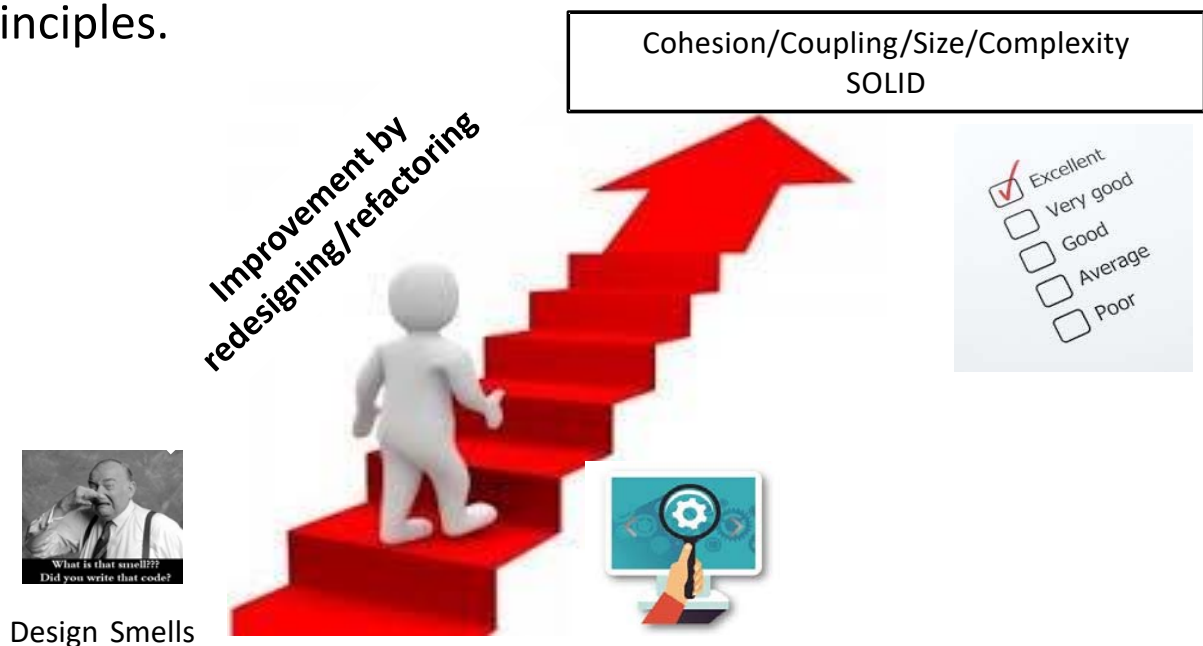
- Concerned with design understandability/maintainability



4

# Principle-based Refactoring

- Design should be evaluated and improved toward design principles.



5

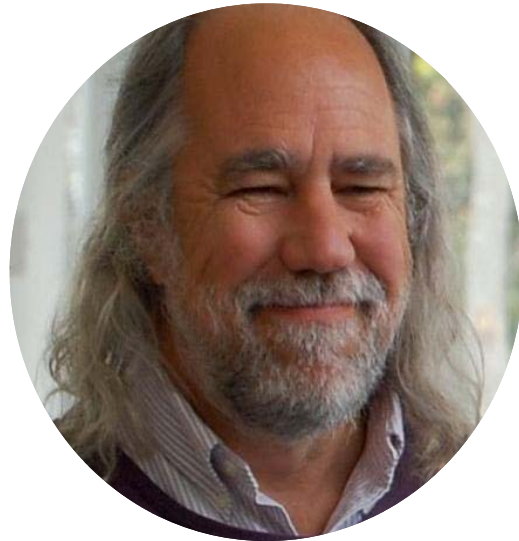
## OO Principles

- SRP - Single Responsibility Principle
- OCP - Open Closed Principle
- LSP - Liskov Substitution Principle
- ISP - Interface Segregation Principle
- DIP - Dependency Inversion Principle
- DRY - Don't Repeat Yourself
- SCP - Speaking Code Principle
- REP - Reuse/ Release Equivalency Principle
- CRP - Common Reuse Principle
- CCP - Common Closure Principle
- ADP - Acyclic Dependencies Principle
- SDP - Stable Dependencies Principle
- SAP - Stable Abstractions Principle
- TDA - Tell Don't Ask
- LOD - Law of Demeter
- SOC - Separation of Concerns
- Separate interface from implementation (Encapsulation)
- Program to an interface
- Favor composition over inheritance
- High cohesion and low coupling
- Protected Variation
- Indirection
- Single level of abstraction principle (SLAP)
- YAGNI - You ain't gonna need it
- KISS - Keep it simple and stupid
- ...

6

# Booch's Fundamental Principles

- Abstraction
- Encapsulation
- Modularization
- Hierarchy




7

## Yet Another Classification of Design Smells

Abstraction	Modularization	Encapsulation	Hierarchy
<ul style="list-style-type: none"><li>• Missing</li><li>• Imperative</li><li>• Incomplete</li><li>• Multifaceted</li><li>• Unnecessary</li><li>• Underutilized</li><li>• Duplicated</li></ul>	<ul style="list-style-type: none"><li>• Broken</li><li>• Insufficient</li><li>• Cyclically-dependent</li><li>• Hub-like</li></ul>	<ul style="list-style-type: none"><li>• Deficient</li><li>• Leaky</li><li>• Missing</li><li>• Unexploited</li></ul>	<ul style="list-style-type: none"><li>• Missing</li><li>• Unnecessary</li><li>• Unfactored</li><li>• Wide</li><li>• Speculative</li><li>• Deep</li><li>• Rebellious</li><li>• Broken</li><li>• Multipath</li><li>• Cyclic</li></ul>

Refactoring for  
Software Design Smells  
Managing Technical Debt



MK  
Girish Suryanarayana,  
Ganesh Samarthiyam, Tushar Sharma  
Forewords by Grady Booch and Stéphane Ducasse

8

# Some Important Principles

Principle	Description
<b>SCP</b>	<u>The code should communicate its purpose.</u>
<b>Cohesion, SRP</b>	A class should have <u>one, and only one, reason to change</u>
<b>Coupling</b>	A class should <u>minimize its dependency</u> on others
<b>DRY</b>	<u>Do not write the same or similar code</u> more than once.
<b>DIP</b>	High-level concepts <u>shall not depend on low-level concepts/implementations.</u>

9

## Design Principles and Bad Smells

- Bad smells describes the situation where design principles are not satisfied

Principle	Bad smells
SCP	Comment, Deeply Nested Condition, Error Code
Cohesion, SRP	Long Method Large Class, Divergent Change Feature Envy, Primitive Obsession, Refused Bequest
Coupling	Long Parameter List, Control Couple, Message Chains Fat Interface
DRY	Duplicate Code, Dependent Logic
DIP	Inappropriate Intimacy, Switch Statements

10

# Speaking Code Principle (SCP)

- The code should communicate its purpose.

"Any fool can write code that a computer  
can understand.

Good programmers write code that  
humans can understand"

-- Martin Fowler, 1999.

11

## SCP-related Bad Smells

- Comments: Comments in the code could indicate that the code communicates its purpose insufficiently.

```
// performs the transaction
public void execute() {
    // get references to bank database and screen
    BankDatabase bankDatabase = getBankDatabase();
    Screen screen = getScreen();

    // get the total balance for the account involved
    double totalBalance = bankDatabase.getTotalBalance( getAccountNumber() );

    // display the balance information on the screen
    screen.displayMessageLine( "\nBalance Information:" );
    screen.displayMessage( "\n - Total balance:  " );
    screen.displayDollarAmount( totalBalance );
    screen.displayMessageLine( "" );
}
```

12

# SRP: A class should have one, and only one reason to change

Operations in the class should be closely related to one subject area → *strong cohesion*

Shift the meaning a little bit, then operations in the class should be closely related to one responsibility.

In the context of the SRP, a responsibility is defined as “a reason for change” or “axis of change”

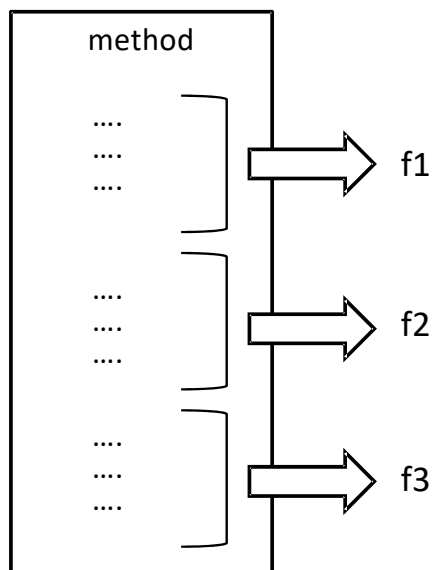
If you can think of more than one motive for changing a class, you are violating the SRP.

Employee
doAsEmployee() genReport() saveToDB()

13

## SRP-related Bad Smells

- **Long method:** A long methods implements have more than one functions.

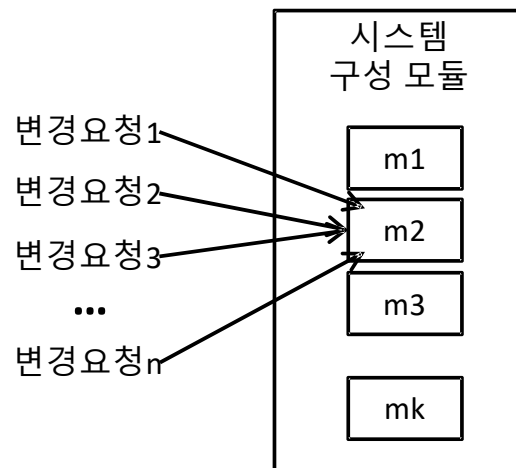


Method can be changed by three independent reasons.

14

# SRP-related Bad Smells

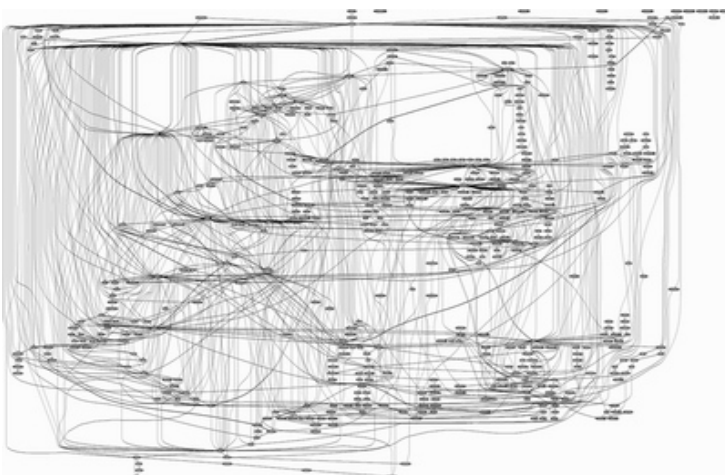
- **Divergent change:** when one class is commonly changed in different ways for different reasons.



15

## Coupling

- Degree of interdependence between two modules.
- Highly coupled systems are harder to understand and maintain.



How to achieve low coupling

- Eliminate unnecessary relationships
- Reduce the number of necessary relationships

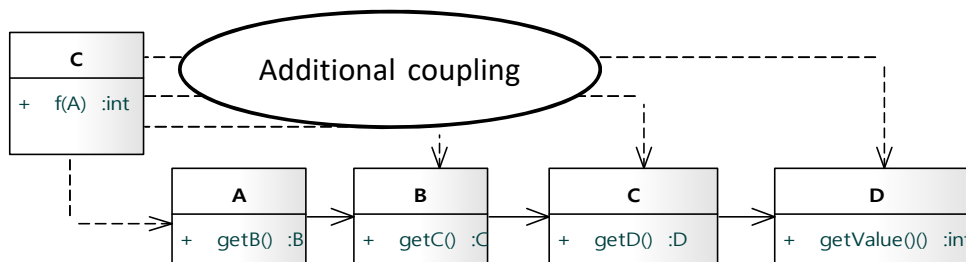
16



# Coupling-related Smells

- **Message chains:** You see message chains when a client asks one object for another object, which the client then asks for yet another object, which the client then asks for yet another object, and so on

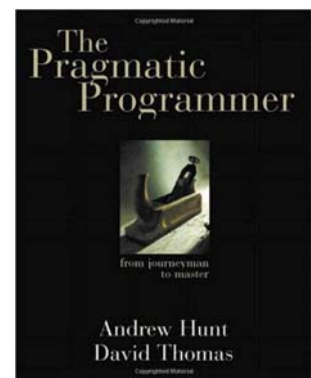
```
class C {  
    public int f(A a) {  
        return a.getB().getC().getD().getValue() ;  
    }  
}
```



17

## Don't Repeat Yourself (DRY)

- The Pragmatic Programmer, by Andy Hunt and Dave Thomas, 2010.
- The DRY principle states that these small pieces of knowledge may **only occur exactly** once in your entire system.
- Every piece of knowledge must **have a single, unambiguous, authoritative representation** within a system.
- When you find yourself writing code that is similar or equal to something you've written before, take a moment to think about what you're doing and don't repeat yourself.

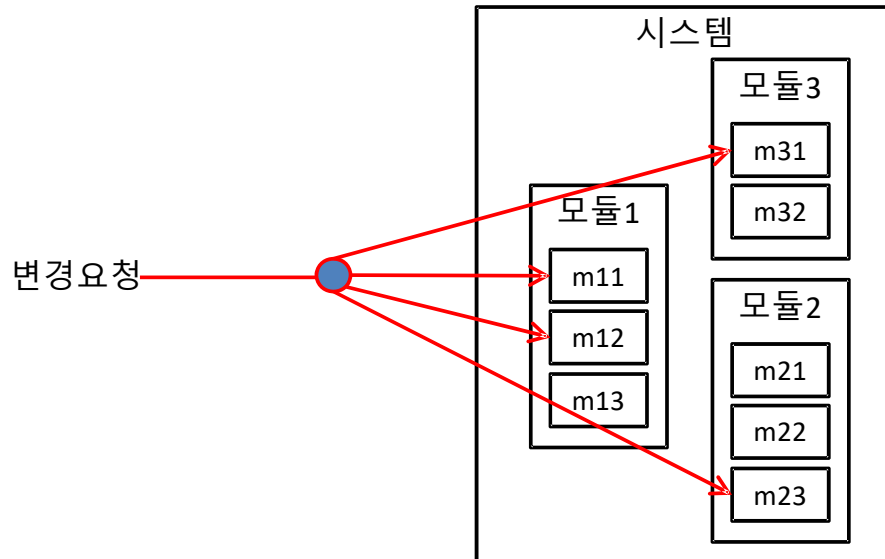


Once and Only Once  
Single Source of  
Truth  
Single Point of Truth

18

# DRY-related Smells

- **Shotgun surgery:** When you have the same thing expressed in two or more places, you have to remember to change the others, if you change one.



19

## DIP: The Dependency Inversion Principle

*High level modules should not depend upon low level modules. Both should depend on abstractions.*

*Abstractions should not depend upon details. Details should depend upon abstraction.*

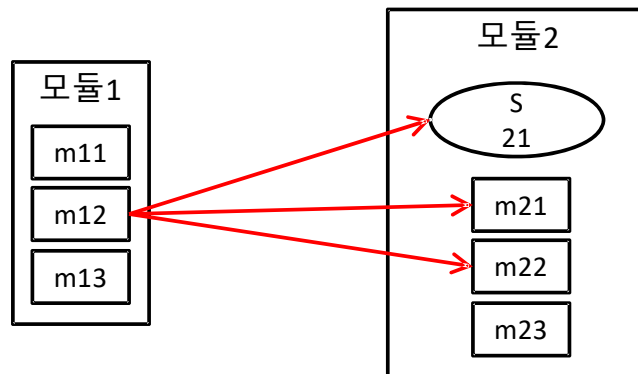
OCP states the **goal**; DIP states the **mechanism**.

A base class in an inheritance hierarchy should not know any of its subclasses.

Modules with detailed implementations are not depended upon, but depend themselves upon abstractions.

# DIP-related Smells

- **Inappropriate intimacy:** Sometimes classes become far too intimate and spend too much time delving in each others' private parts



```
Person kent = new Person();
Set s = new HashSet();
s.add(new Course ("Smalltalk Programming", false));
s.add(new Course ("Appreciating Single Malts", true));
kent.initializeCourses(s);
```

21

## Design Principles and Design Patterns

- Refactoring towards a pattern can satisfy certain design principle(s) in removing bad smells.

Principle	Patterns
SCP	-
Cohesion, SRP	Decorator Pattern, Observer Pattern
Coupling	Façade Pattern
DRY	Factory Method Pattern, Abstract Factory Pattern, Template Method Pattern
DIP	Iterator Pattern, Composite Pattern, Strategy Pattern, Command Pattern, State Pattern

22

# The Rules

- Apply Common Sense
- Don't get too dogmatic / religious
- Every decision is a tradeoff
- All other principles are just that
  - Guidelines
  - “best practices”
  - Consider carefully if you should violate them - but, know you can.