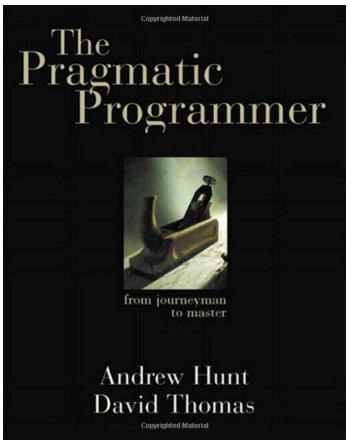
DESIGN PRINCIPLE-BASED REFACTORING: DRY

Don't Repeat Yourself (DRY)

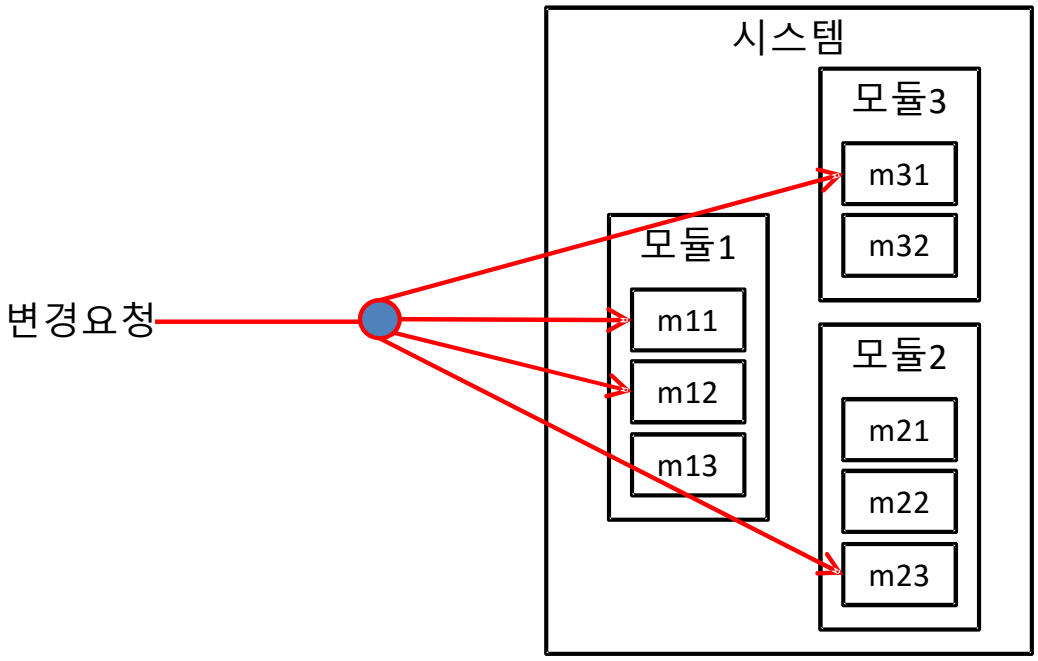
- The Pragmatic Programmer, by Andy Hunt and Dave Thomas, 2010.
- The DRY principle states that these small pieces of knowledge may **only occur exactly** once in your entire system.
- Every piece of knowledge must **have a single, unambiguous, authoritative representation** within a system.
- When you 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

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.

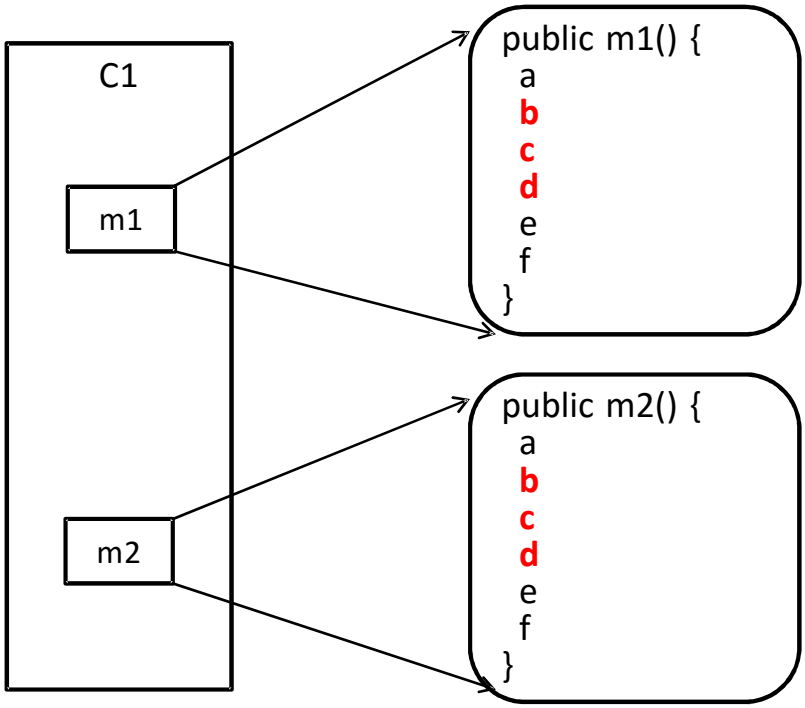


Refactoring Techniques

Smell	Refactoring
Code duplicate across methods	Extract method
Similar codes among methods	Parameterize method
Isolate and encapsulate object creation knowledge	Factory method Pattern
similar codes in subclasses	Template method Pattern

Extract Method

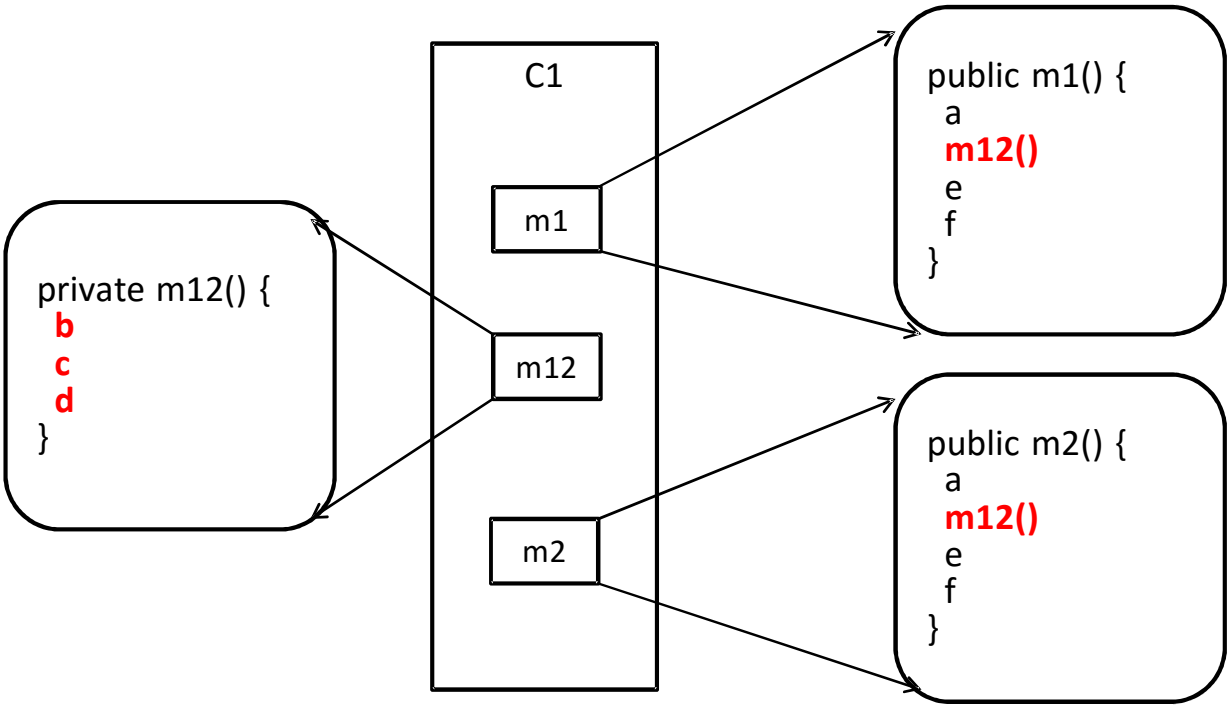
You have the same or similar codes



Try CPD in PMD and CCFinder for detecting clones

Extract Method

Extract the same or similar code into a method



Parameterize Method

Several methods do similar things but with different values

Create one method that uses a parameter for the different values

```
void tenPercentRaise () {
   salary *= 1.1;
}
void fivePercentRaise () {
   salary *= 1.05;
}
```

```
void raise (double factor) {
   salary *= (1 + factor);
}
```


Parameterize Method

Less obvious case; find similar, but different parts

```
public Dollars baseCharge() {
   double result = Math.min(lastUsage(),100) * 0.03;
   if (lastUsage() > 100) {
      result += (Math.min (lastUsage(),200) - 100) * 0.05;
   }
   if (lastUsage() > 200) {
      result += (lastUsage() - 200) * 0.07;
   }
   return new Dollars (result);
}
```

Parameterize Method

```
public Dollars baseCharge() {
   double result = usageInRange(0, 100) * 0.03;
   result += usageInRange (100,200) * 0.05;
   result += usageInRange (200, Integer.MAX_VALUE) * 0.07;
   return new Dollars (result);
}
private int usageInRange(int start, int end) {
   if (lastUsage() > start)
      return Math.min(lastUsage(),end) - start;
   else
      return 0;
}
```


DRY-related Patterns

- Factory Method Pattern
- Template Method Pattern