Code SMELL (Shotgun Surgery)

# shotgun surgery

The Code Smells are similar in concept to Development-level Anti-patterns. Sometimes in our code, we introduce code smell unintentionally those makes our design fragile.

**Definition of Code smell**

Code smell, also known as a bad smell, in computer programming code, refers to any symptom in the source code of a program that possibly indicates a deeper problem.

**Martin fowler says:**

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

Code smell creates lot of problems while introducing new feature or maintains the codebase,

Often developer has to write repeatable code, breaking encapsulation, breaking abstraction etc.

If code smells are not corrected.

So always refactor code smell while developing.

In this article, we discuss one of the popular code smell “***SHOTGUN SURGERY***”

Shotgun surgery says, to introduce a small new change, a developer has to change many classes and methods, most of the time has to write duplicate codes which violate “**Don’tRepeatYourself**” principle.

**Cause of Shotgun surgery smell:**

1. Due to poor separation of concern.

2. Fail to understand the responsibility. Often due to misunderstanding (Single responsibility principle)

3. Not identifying the common behavior or behavior with a slight change.

4. Fail to introduce proper design pattern.

Consequences of Shotgun Surgery:

1. Lots of duplicates code

2. Taking more time to develop a small feature.

3. Unmaintainable code base.

**Refactoring strategy:**

1. We can do it by “**Move Method**”, “**Move Field**” or “**Inline class**”

We will discuss above strategies in another article.

Let see an example, where “Shotgun Surgery” smell is present

**package** com.example.codesmell;

**public** **class** Account {

**private** String type;

**private** String accountNumber;

**private** **int** amount;

**public** Account(String type,String accountNumber,**int** amount)

{

**this**.amount=amount;

**this**.type=type;

**this**.accountNumber=accountNumber;

}

**public** **void** debit(**int** debit) **throws** Exception

{

**if**(amount <= 500)

{

**throw** **new** Exception("Mininum balance shuold be over 500");

}

amount = amount-debit;

System.***out***.println("Now amount is" + amount);

}

**public** **void** transfer(Account from,Account to,**int** cerditAmount) **throws** Exception

{

**if**(from.amount <= 500)

{

**throw** **new** Exception("Mininum balance shuold be over 500");

}

to.amount = amount+cerditAmount;

}

**public** **void** sendWarningMessage()

{

**if**(amount <= 500)

{

System.***out***.println("amount should be over 500");

}

}

}

**package** com.example.codesmell;

**public** **class** ShotgunSurgery {

**public** **static** **void** main(String[] args) **throws** Exception {

Account acc = **new** Account("Personal","AC1234",1000);

acc.debit(500);

acc.sendWarningMessage();

//acc.debit(500);

}

}

If we pay attention in Account.java file, we can see every operation **debit(),** **transfer(),sendWarningMessage()** has one validation, account balance should be more than 500.

And we copy the same validation in every method because of we not able to identify the common validation, so we introduce a “Shotgun Surgery” code smell.

The real problem occurs when we add another criterion in validation logic that is if account type is personal and balance is over 500 then we can perform above operations.

In this scenario, we have to make change in all methods which is not intended so let’s see how to solve it

Create a common method call **isAccountUnderflow()** that will solve the problem, all validation related stuff goes there.

**Refactored code**:

**package** com.example.codesmell;

**public** **class** AcountRefactored {

**private** String type;

**private** String accountNumber;

**private** **int** amount;

**public** AcountRefactored(String type,String accountNumber,**int** amount)

{

**this**.amount=amount;

**this**.type=type;

**this**.accountNumber=accountNumber;

}

**private** **boolean** isAccountUnderflow()

{

**if**(amount <= 500)

{

**return** **true**;

}

**return** **false**;

}

**public** **void** debit(**int** debit) **throws** Exception

{

**if**(isAccountUnderflow())

{

**throw** **new** Exception("Mininum balance shuold be over 500");

}

amount = amount-debit;

System.***out***.println("Now amount is" + amount);

}

**public** **void** transfer(AcountRefactored from,AcountRefactored to,**int** cerditAmount) **throws** Exception

{

**if**(isAccountUnderflow())

{

**throw** **new** Exception("Mininum balance shuold be over 500");

}

to.amount = amount+cerditAmount;

}

**public** **void** sendWarningMessage()

{

**if**(isAccountUnderflow())

{

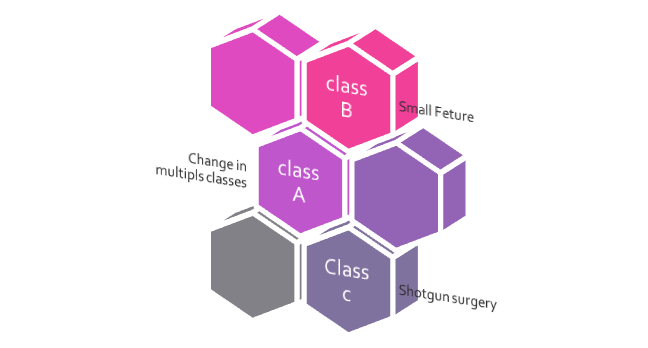
System.***out***.println("amount should be over 500");

}

}

}

**Picture :**





**ShotGun Surgery**