Refactoring Fundamentals

Code Smells: The Couplers

Steve Smith
Ardalis.com
@ardalis

In This Course

- ~~What is Refactoring?~~
- ~~Why do it?~~
- ~~What's the process?~~
- ~~What are some tools that can assist with it?~~
- ~~What is a *Code Smell*?~~
- What are some examples of Code Smells?
- What are some common refactorings?
- How does one apply them correctly?

Organizing Code Smells

- **Taxonomy proposed by Mäntylä, M. V. and Lassenius, C.**
  - http://www.soberit.hut.fi/~mmantyla/ESE_2006.pdf

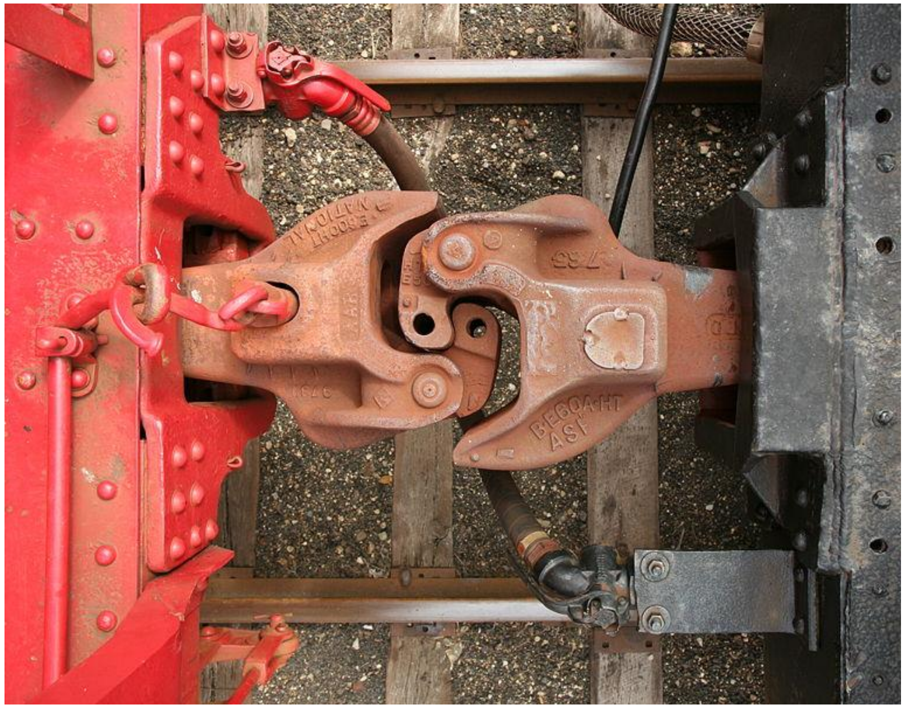
- **Organization of Code Smells into 5 Groups**
  - ~~The Bloaters~~
  - ~~The Object-Orientation Abusers~~
  - ~~The Change Preventers~~
  - ~~The Dispensables~~
  - The Couplers

- **I've added three more:**
  - ~~The Obfuscators~~
  - Environment Smells
  - Test Smells

Code Smells: The Couplers

- **Smells related to coupling**
  - Introduce high coupling
  - Result from attempting to avoid coupling

- **Most of these could also be considered OO abusers**



http://commons.wikimedia.org/wiki/File:Train_coupling.jpg

The Couplers: Feature Envy

- Ideally, object orientation packages data and behavior together
- Characterized by calling getters
- Keep together things that change together
- Some patterns are break this rule
  - Strategy
  - Visitor

Feature Envy

Refactor

- Move Method
- Extract Method

Feature Envy

```
public class Rental {
private Movie _movie;
public decimal GetPrice()
{
    if (_movie.IsNewRelease)
    {
        if (_movie.IsChildrens)
        {
            return 4;
        }
        return 5;
    }
    if (_movie.IsChildrens)
    {
        return 2;
    }
    return 3;
}
}
```

Feature Envy

```
public class Movie
{
    public bool IsNewRelease { get; set; }
    public bool IsChildrens { get; set; }
    public string Title { get; set; }

    public decimal GetPrice()
    {
        if (IsNewRelease)
        {
            if (IsChildrens)
            {
                return 4;
            }
            return 5;
        }
        if (IsChildrens)
        {
            return 2;
        }
        return 3;
    }
}
```

Feature Envy

```
public class Movie
{
    public bool IsNewRelease { get; set; }
    public string Title { get; set; }
     public virtual decimal GetPrice()
    {
        if (IsNewRelease)
        {
            return 5;
        }
        return 3;
    }
}

public class ChildrensMovie : Movie
{
    public override decimal GetPrice()
    {
        if (IsNewRelease)
        {
                return 4;
        }
        return 2;
    }
}
```

The Couplers: Inappropriate Intimacy

- **Classes that know way too much about one another**
- **Keep class honest by going through clean interfaces**
- **Watch out for:**
  - Inheritance
  - Bidirectional relationships

Refactor

- **Move Method**
- **Move Field**
- **Change Bidirectional Association to Unidirectional**
- **Extract Class**
- **Replace Inheritance with Delegation**

The Couplers: Inappropriate Intimacy

G8: Too Much Information

- **Avoid wide and deep interfaces**
- **Small interfaces reduce coupling**
- **Limit your code's surface area**
  - Fewer methods
  - Fewer variables
  - Fewer instance variables

Tip Concentrate on keeping interfaces very tight and very small. Keep coupling low by limiting information.

Clean Code

Inappropriate Intimacy and the Law of Demeter

- **Law of Demeter**
  - A given object should assume as little as possible about the structure or properties of anything else (including its own subcomponents)

- **A method *m* on object *O* may only invoke methods on**
  - *O* itself
  - *m*'s parameters
  - Any object created within method *m*
  - *O*'s direct component objects (fields and properties)
  - Global variables and static methods

The Paperboy and the Wallet

- **Customer**
  - Wallet { get; }
- **Wallet**
  - AddMoney(decimal deposit)
  - RemoveMoney(decimal debit)
  - TotalMoney { get; }

http://flic.kr/p/bf3UxM

http://www.ccs.neu.edu/research/demeter/demeter-method/LawOfDemeter/paper-boy/demeter.pdf

Paperboy Class

```
public void GetPaidByCustomer(Customer customer)
{
   decimal payment = 2.00;
   var wallet = customer.Wallet;
   if(wallet.Total > payment)
   {
     wallet.RemoveMoney(payment);
   }
   else
   {
     // come back later to get paid
   }
}
```

Customer Class (refactored)

```
public class Customer
{
  private Wallet _wallet;
  public decimal RequestPayment(decimal amount)
  {
    if(_wallet != null && _wallet.Total > amount)
    {
      _wallet.RemoveMoney(amount);
      return amount;
    }
    return 0;
  }
}
```

Paperboy Class

```
public void GetPaidByCustomer(Customer customer)
{
   decimal payment = 2.00;
   decimal amountPaid = customer.RequestPayment(payment);
   if(amountPaid == payment)
   {
     // say thank you and provide a receipt
   }
   else
   {
     // come back later to get paid
   }
}
```

The Couplers: Indecent Exposure

- Sometimes classes or methods are public but shouldn’t be
- Violates encapsulation
- Can lead to Inappropriate Intimacy

Refactor

- Encapsulate Classes with Factory

The Couplers: Message Chains

- **Occur when a client asks an object for another object**
  - Then asks that object for another object
  - Then asks that one for yet another object

- **Another example of a Law of Demeter violation**

- **Couples the client to the structure of the navigation**

Refactor

- **Hide Delegate**
- **Extract Method**
- **Move Method**

The Couplers: Middle Man

- Sometimes delegation goes too far
- Hiding direct access to dependent objects is generally good…
- Until it seems like that's all the class is doing

Refactor

- Remove Middle Man
- Inline Method
- Replace Delegation with Inheritance

The Couplers: Tramp Data

- Data passed only because something else needs it
- Might be ok, if consistent to current abstraction

Refactor

- Remove Middle Man
- Extract Method
- Extract Class

The Couplers: Artificial Coupling

- **Avoid creating coupling in your code structures where it isn't necessary for the abstraction being used**

- **Examples**
  - General enums in specific classes
  - General static methods or variables in specific classes

Refactor

- **Move Method**

Tip

Things that don't depend upon each other should not be artificially coupled.

Clean Code

The Couplers: Hidden Temporal Coupling

- Structure code to enforce required order

Refactor

- Form Template Method
- Introduce Intermediate Results

Hidden Temporal Coupling

```
public void MakePizza
{
    PrepareCrust();
    AddToppings();
    Bake();
    CutIntoSlices();
}
```

Hidden Temporal Coupling

```
// template method in base class
public void MakeBakedGood()
{
   PrepareCrust();
   AddToppings();
   Bake();
   CutIntoSlices();
}

public class PizzaBakery : Bakery
{
   override PrepareCrust() { … }
   override AddToppings { … }
   override Bake { … }
   override CutIntoSlices { … }
}
```

Hidden Temporal Coupling

```
public SlicedPizza MakePizza (Dough dough)
{
    Crust crust = PrepareCrust(dough);
    ToppedDough toppedDough = AddToppings(crust);
    CookedPizza myPizza = Bake(toppedDough);
    SlicedPizza result = CutIntoSlices(myPizza);

    return result;
}
```

The Couplers: Hidden Dependencies

- Classes should declare their dependencies in their constructor
- Anything the class needs that isn't passed in via the constructor (or a parameter) is a *hidden dependency*
- Violate the Explicit Dependencies Principle
- Frequently consist of
  - Object instantiation (e.g. "new")
  - Non-stateless static calls

http://ardalis.com/new-is-glue

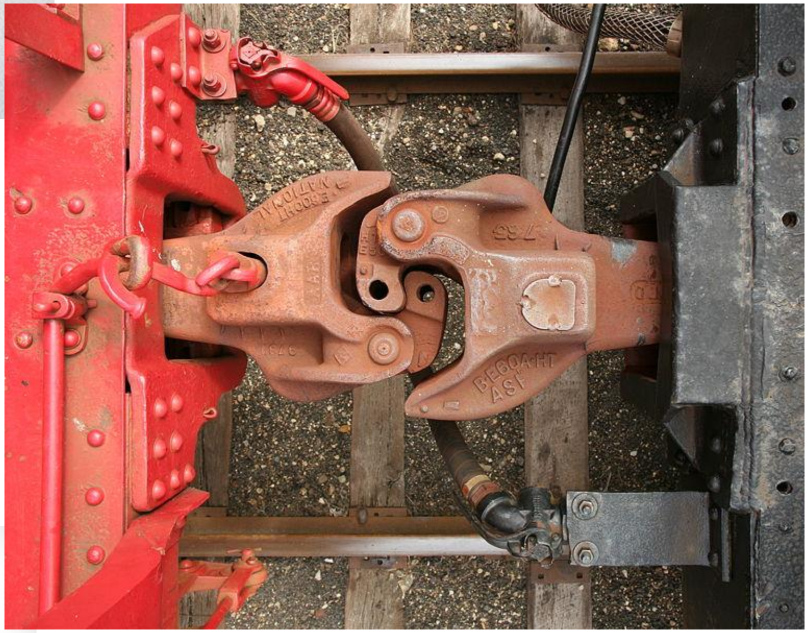
Refactor

- Replace Fixed Variable with Parameter
- Dependency Injection

http://flic.kr/p/aN4Zv

Summary

Feature Envy



References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Design Patterns Library http://bit.ly/SJmAX1

Books

Code Complete http://amzn.to/Vq5YLv

Clean Code http://amzn.to/YjUDI0

Refactoring http://amzn.to/110tscA

Web

The Paperboy, The Wallet, and The Law Of Demeter http://bit.ly/18lYFm

New is Glue http://ardalis.com/new-is-glue

Explicit Dependencies Principle http://deviq.com/explicit-dependencies-principle

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice

pluralsight
hardcore developer training