Refactoring Fundamentals

Code Smells: Change Preventers

Steve Smith
Ardalis.com
@ardalis

pluralsight
hardcore developer training

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 Change Preventers

- Touch many parts of the system
- Tight coupling
- Poor separation of concerns or responsibilities

http://flic.kr/p/5S2p2b

Change Preventers: Divergent Change

- A class is commonly changed in at least two different ways
- Indicates a violation of the Single Responsibility Principle

Refactor

- Extract Class

Tip

Any change to handle a variation should change a single class.

Kent Beck

Change Preventers: Shotgun Surgery

- Many small changes, all over the place
- Hard to find them all; easy to miss some

Refactor

- Move Method
- Move Field
- Inline Class
- (many others)

Ideally, there should be a one-to-one relationship between changes and classes.

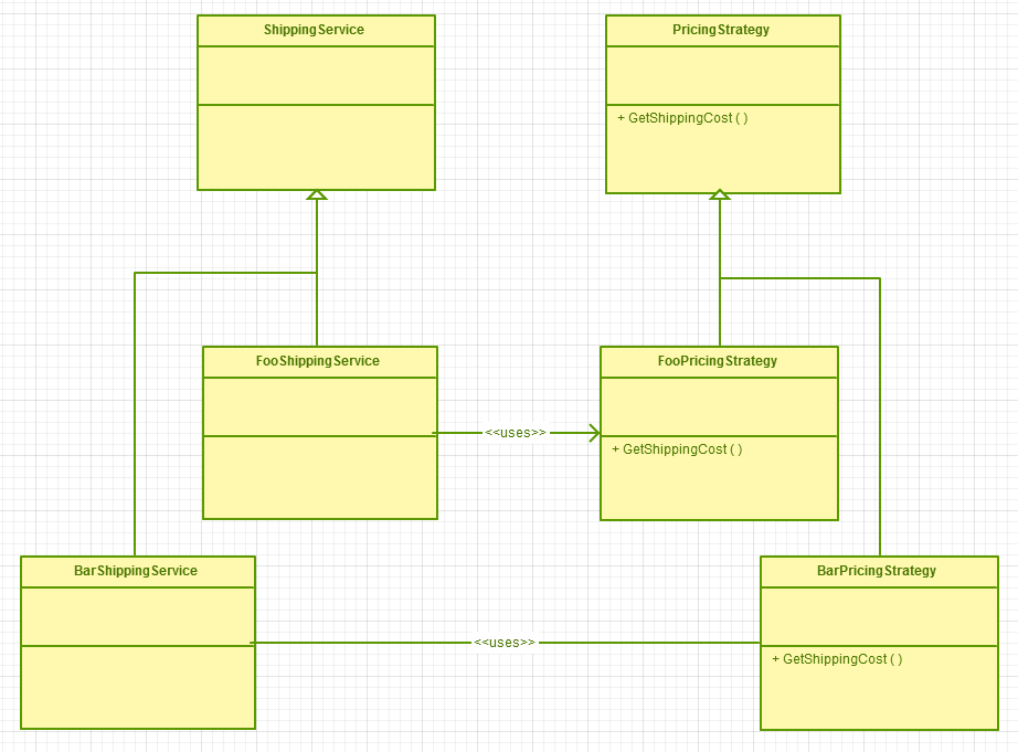
Also known as: Solution Sprawl

http://commons.wikimedia.org/wiki/File:Shotgun_wound-xray.JPG

Change Preventers: Parallel Inheritance Hierarchies

- **Every time you make a subclass of one class, you need a subclass of another**

- **Subclasses frequently share the same prefix**
  - FooShippingService
  - FooPricingStrategy
  - BarShippingService
  - BarPricingStrategy

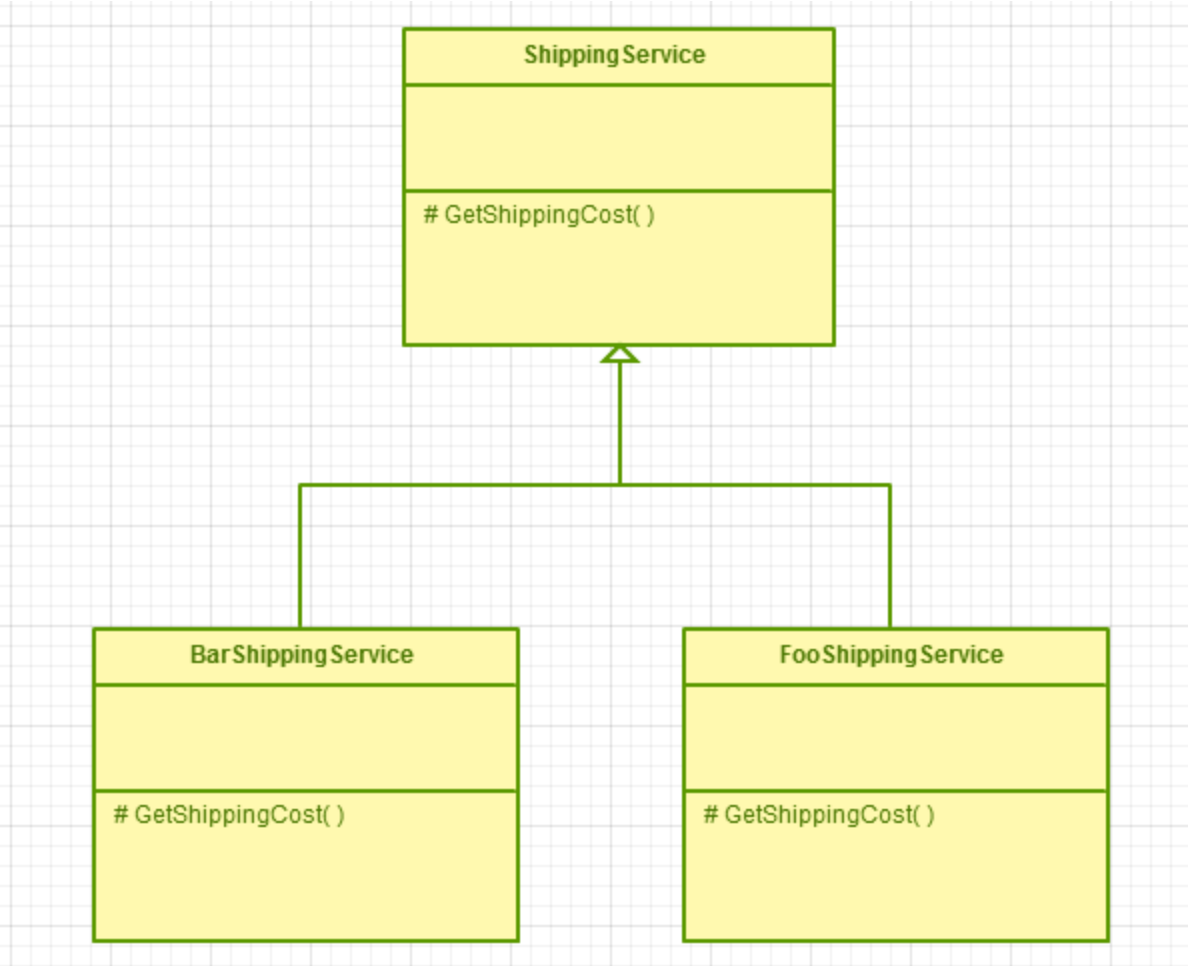


- **Special case of Shotgun Surgery**

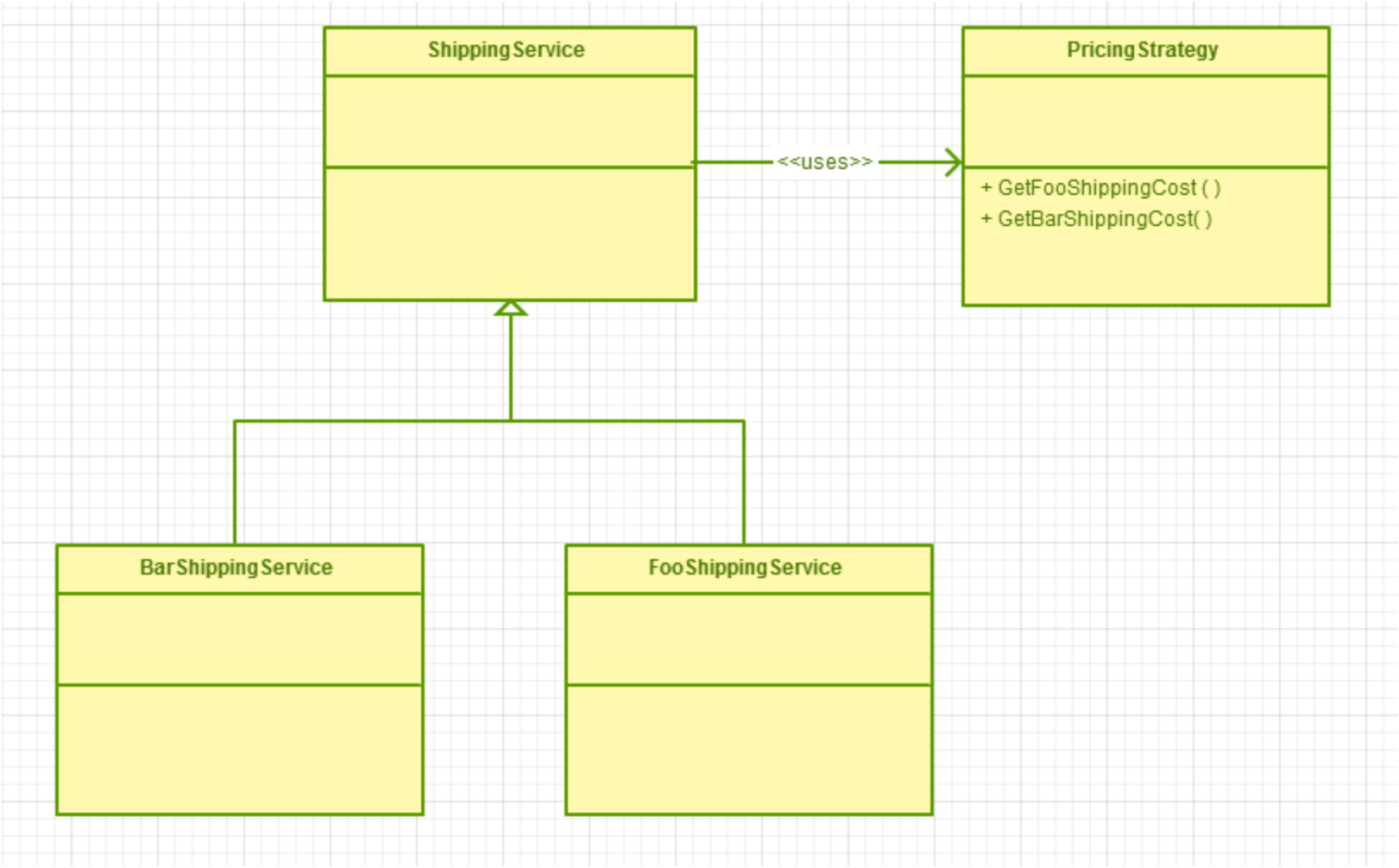
Refactor

- **Move Method**
- **Move Field**

Refactoring Parallel Inheritance Hierarchies



Refactoring Parallel Inheritance Hierarchies

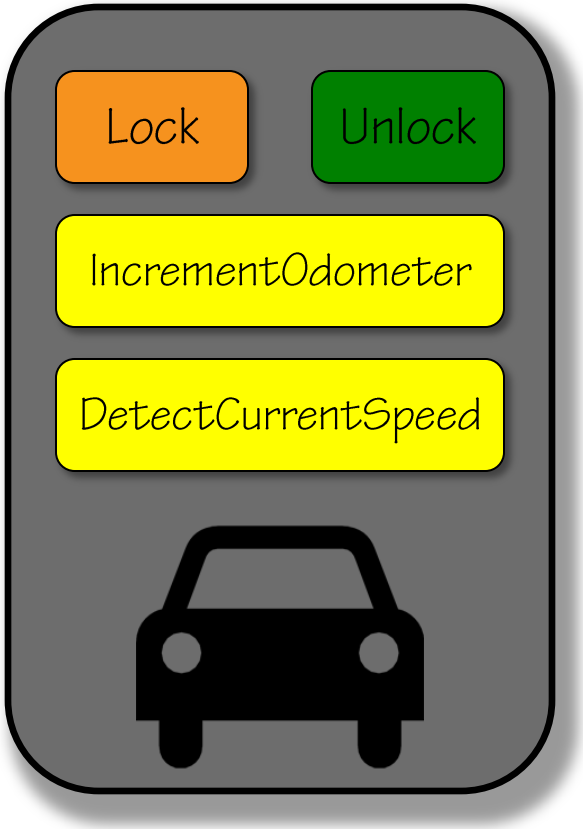


Change Preventers: Inconsistent Abstraction Level

- **Class interfaces should provide a consistent level of abstraction**
- **Often degrades over time with addition of expedient methods**
- **Functions**
  - Should use the same level of abstraction internally
  - It should be one level of abstraction below the operation defined by the function's name

Refactor

- **Move Method**
- **Extract Method**



Change Preventers: Conditional Complexity

- **Easily detected – Cyclomatic Complexity**
  - http://en.wikipedia.org/wiki/Cyclomatic_complexity

Code Metrics Results

Filter: None Min:

Hierarchy	Maintaina...	Cyclo...	Depth of I...	Class Cou...	Lines of Code
CodeSmells (Debug)	89	292	5	76	419
CodeSmells.Bloaters.LargeClass	90	100	2	14	117
CodeSmells.Bloaters.LongMethod	84	80	5	27	137
GildedRoseConsoleProgram	56	21	1	5	33
UpdateQuality() : void	44	19		3	28
Main(string[]) : void	67	1		4	4
GildedRoseConsoleProgram()	100	1		0	1

```
Items[i].Quality = Items[i].Quality + 1;

if (Items[i].SellIn < 6)
    if (Items[i].Quality < 50)
        Items[i].Quality = Items[i].Quality + 1;
```

Guest Opinion: Scott Hanselman

Conditional Condition Complexity

```
public void UpdateQuality()
{
   if(Items[i].Quality > 0
      && Items[i].Quality < 50
      && Items[i].Name.Contains(“Backstage passes”)
      && Items[i].SellIn < 11)
    {
      Items[i].Quality += 1;
    }
}
```

Conditional Condition Complexity

```
public void UpdateQuality()
{
   if(Items[i].Quality > 0
      && Items[i].Quality < 50
      && Items[i].Name.Contains(“Backstage passes”)
      && Items[i].SellIn < 11)
    {
      Items[i].Quality += 1;
    }
}

public void UpdateQuality()
{
   AdjustBackstagePassQuality(Items[i]);
}
```

Conditional Condition Complexity

```
public void UpdateQuality()
{
   if(Items[i].Quality > 0
      && Items[i].Quality < 50
      && Items[i].Name.Contains("Backstage passes")
      && Items[i].SellIn < 11)
    {
      Items[i].Quality += 1;
    }
}

public void UpdateQuality()
{
   if(ItemHasValidQuality(Items[i])
       && ItemIsBackstagePassNearingExpirationDate(Items[i]))
    {
      Items[i].Quality += 1;
    }
}
```

Conditional Condition Complexity

```
public void UpdateQuality()
{
   if(!ItemHasInvalidQuality(Items[i])
      && ItemIsBackstagePassNearingExpirationDate(Items[i]))
   {
     Items[i].Quality += 1;
   }
}
```

Tip

Avoid Negative Conditionals.

Robert C. "Uncle Bob" Martin

Change Preventers: Conditional Complexity

Refactor

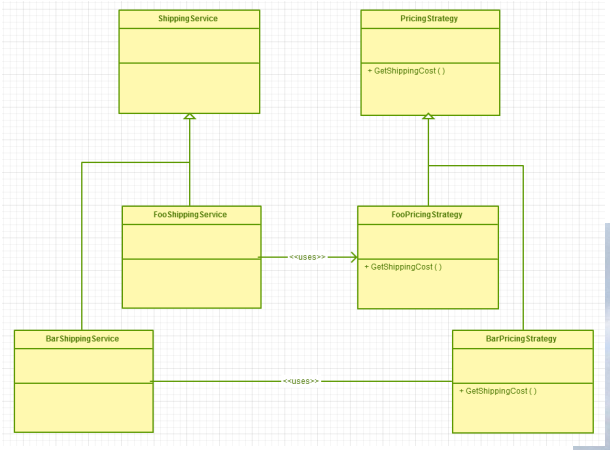
- Extract Method
- Replace Conditional Logic with Strategy
- Move Embellishment to Decorator
- Replace State-Altering Conditionals with State
- Introduce Null Object

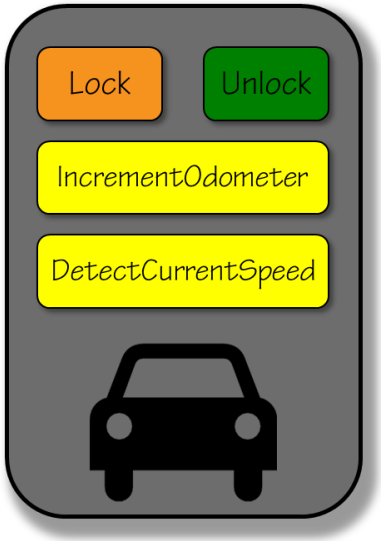
```
public void UpdateQuality()
{
    for (var i = 0; i < Items.Count; i++)
    {
        if (Items[i].Name != "Aged Brie" && Items[i].Name != "Backstage passes to a
        {
            if (Items[i].Quality > 0)
            {
                if (Items[i].Name != "Sulfuras, Hand of Ragnaros")
                {
                    Items[i].Quality = Items[i].Quality - 1;
                }
            }
        }
        else
        {
            if (Items[i].Quality < 50)
            {
                Items[i].Quality = Items[i].Quality + 1;
                if (Items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
                {
                    if (Items[i].SellIn < 11)
                    {
                        if (Items[i].Quality < 50)
                        {
                            Items[i].Quality = Items[i].Quality + 1;
                        }
                    }
                    if (Items[i].SellIn < 6)
                    {
                        if (Items[i].Quality < 50)
                        {
                            Items[i].Quality = Items[i].Quality + 1;
                        }
                    }
                }
            }
        }
    }
```

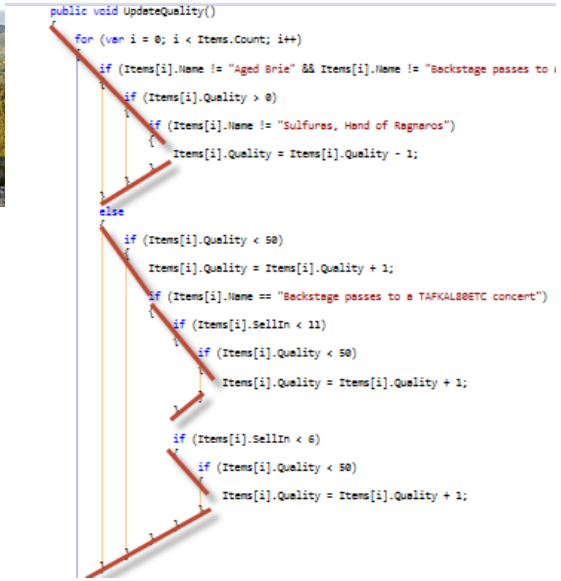
Change Preventers: Poorly Written Tests

- Tests are meant to help
- Badly written tests can prevent change
- Tight coupling
- Brittleness
- Poor performance
- Environment Affinity

Summary







```
public void UpdateQuality()
{
    for (var i = 0; i < Items.Count; i++)
    {
        if (Items[i].Name != "Aged Brie" && Items[i].Name != "Backstage passes to
        {
            if (Items[i].Quality > 0)
            {
                if (Items[i].Name != "Sulfuras, Hand of Ragnaros")
                {
                    Items[i].Quality = Items[i].Quality - 1;
                }
            }
        }
        else
        {
            if (Items[i].Quality < 50)
            {
                Items[i].Quality = Items[i].Quality + 1;
                if (Items[i].Name == "Backstage passes to a TAFKAL80ETC concert")
                {
                    if (Items[i].SellIn < 11)
                    {
                        if (Items[i].Quality < 50)
                        {
                            Items[i].Quality = Items[i].Quality + 1;
                        }
                    }
                    if (Items[i].SellIn < 6)
                    {
                        if (Items[i].Quality < 50)
                        {
                            Items[i].Quality = Items[i].Quality + 1;
                        }
                    }
                }
            }
        }
    }
```

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

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice

pluralsight
hardcore developer training

Guest Opinion: Scott Hanselman