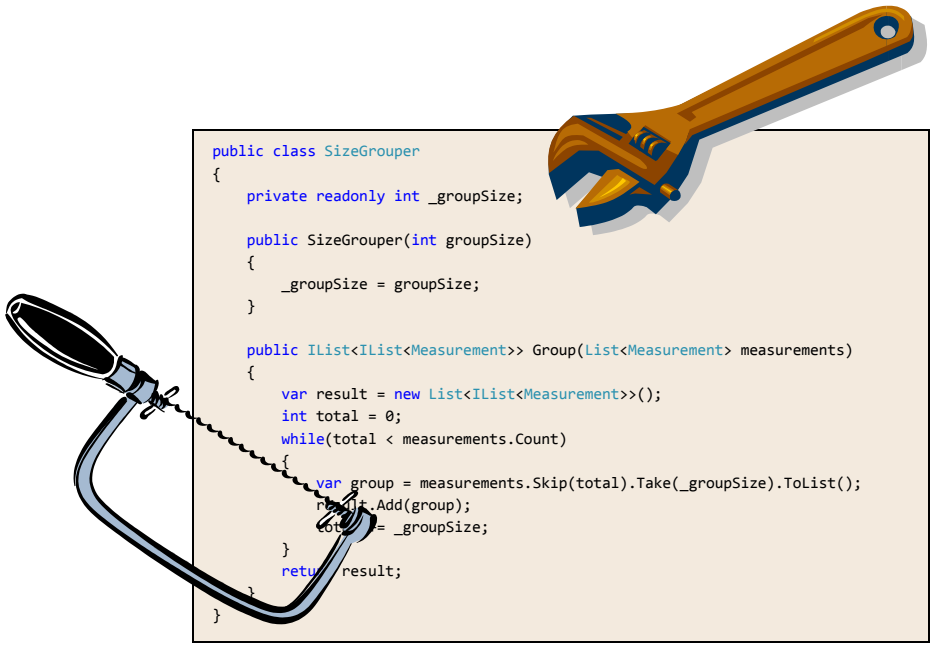
TFD: Refactoring

Scott Allen
http://www.pluralsight.com/

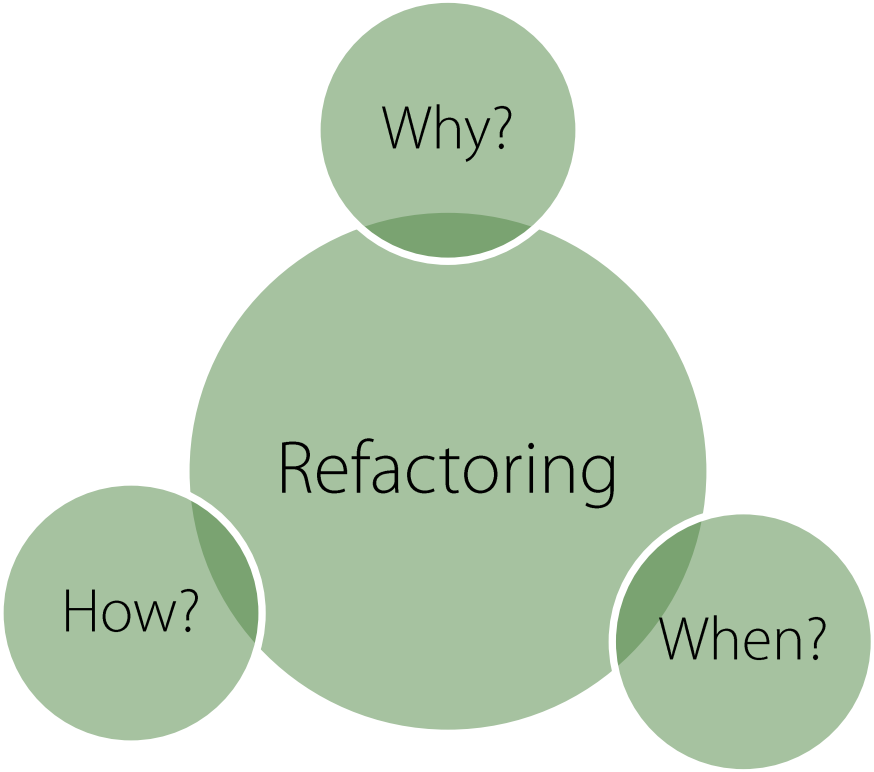
Refactoring Software

Change the internal implementation
without changing the external functionality

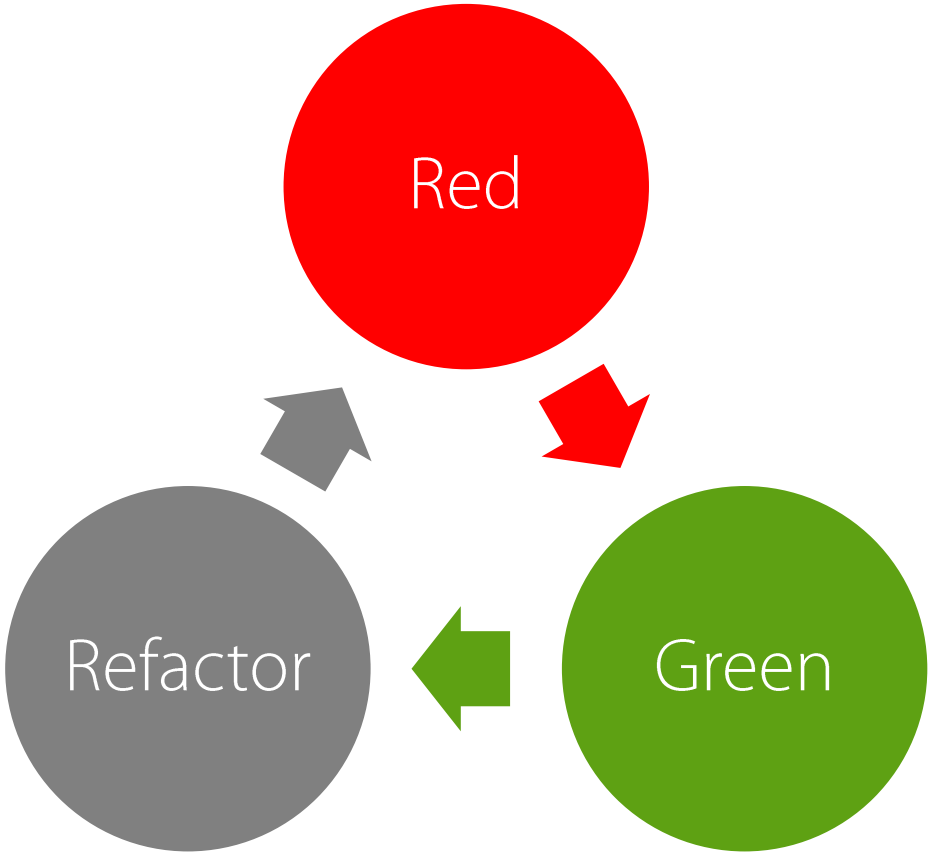


```
public class SizeGrouper
{
    private readonly int _groupSize;

    public SizeGrouper(int groupSize)
    {
        _groupSize = groupSize;
    }

    public IList<IList<Measurement>> Group(List<Measurement> measurements)
    {
        var result = new List<IList<Measurement>>();
        int total = 0;
        while(total < measurements.Count)
        {
            var group = measurements.Skip(total).Take(_groupSize).ToList();
            result.Add(group);
            total += _groupSize;
        }
        return result;
    }
}
```

Topics

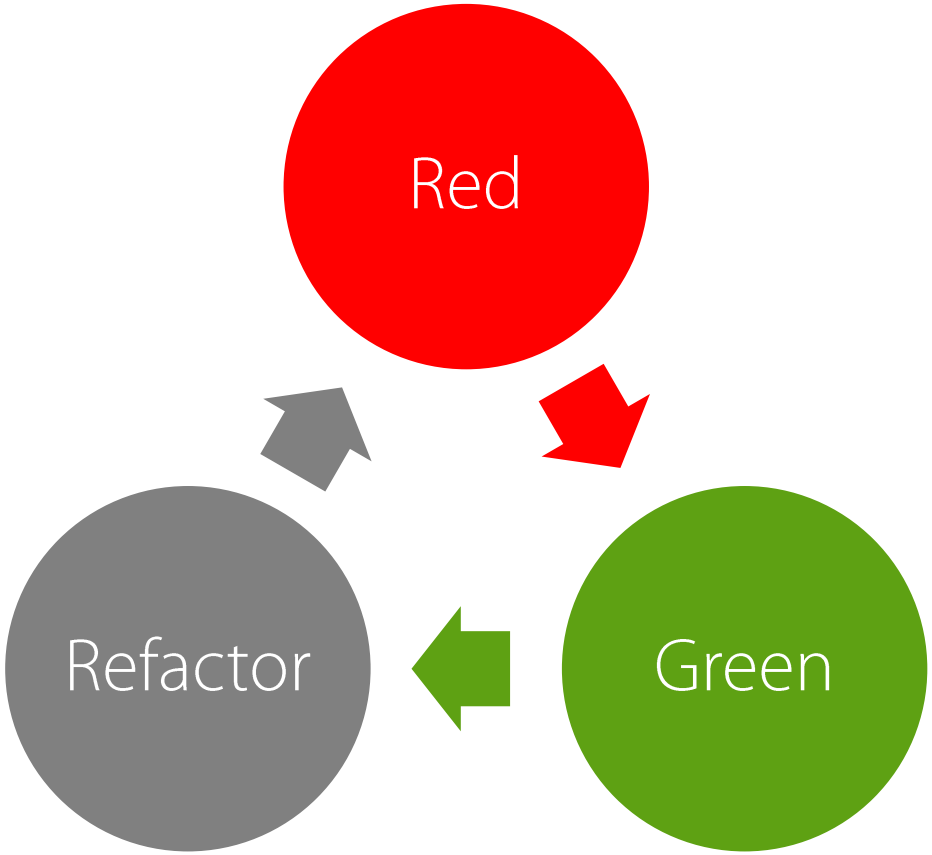


The Cycle



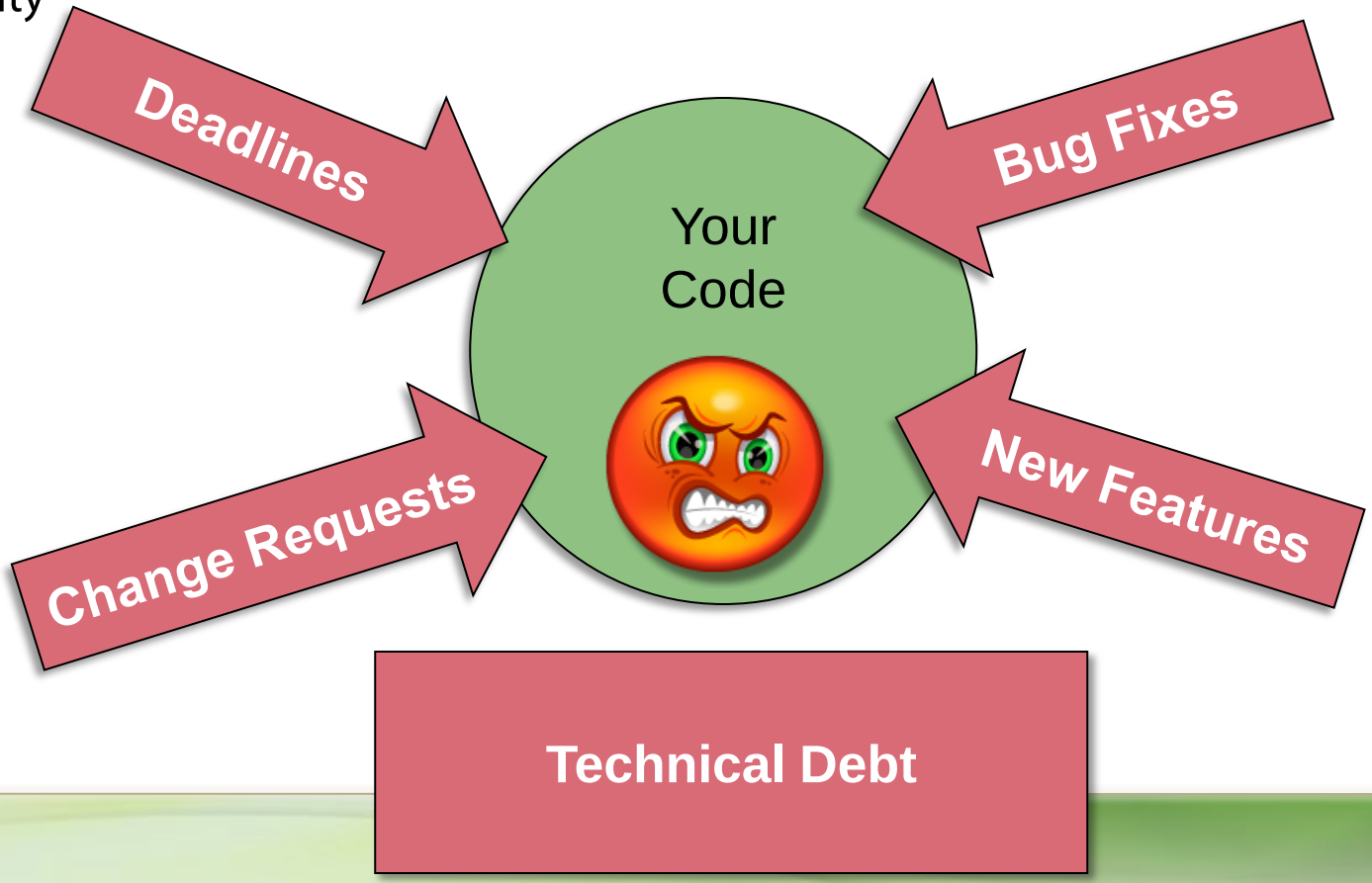
When is the Cook “Done”?

The Cycle



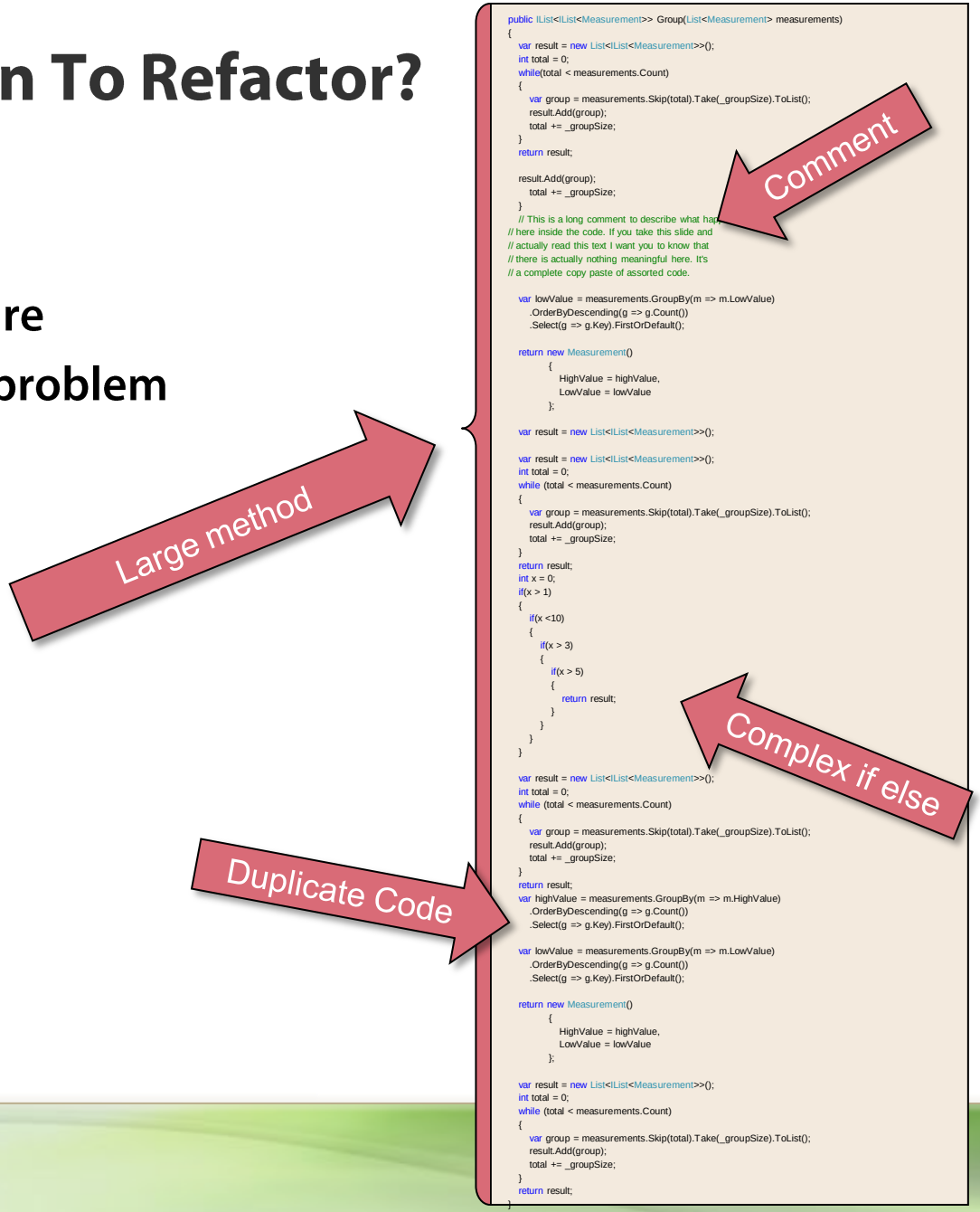
Why Refactor?

- **To improve the quality of the code**
  - Readability
  - Maintainability
  - Scalability
  - Extensibility



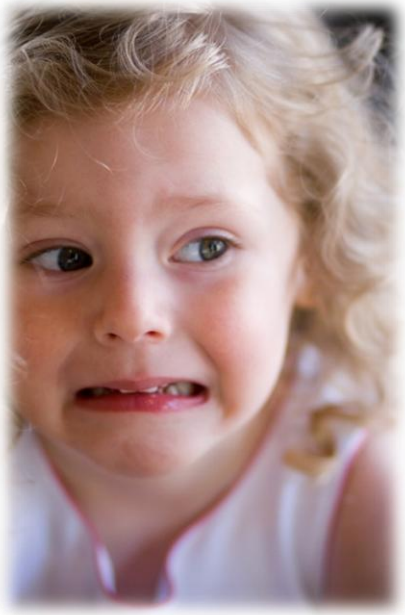
When To Refactor?

- After fixing a failing test
- Before adding a new feature
- After identifying a quality problem



When Not To Refactor

- You have a fear of breaking the software

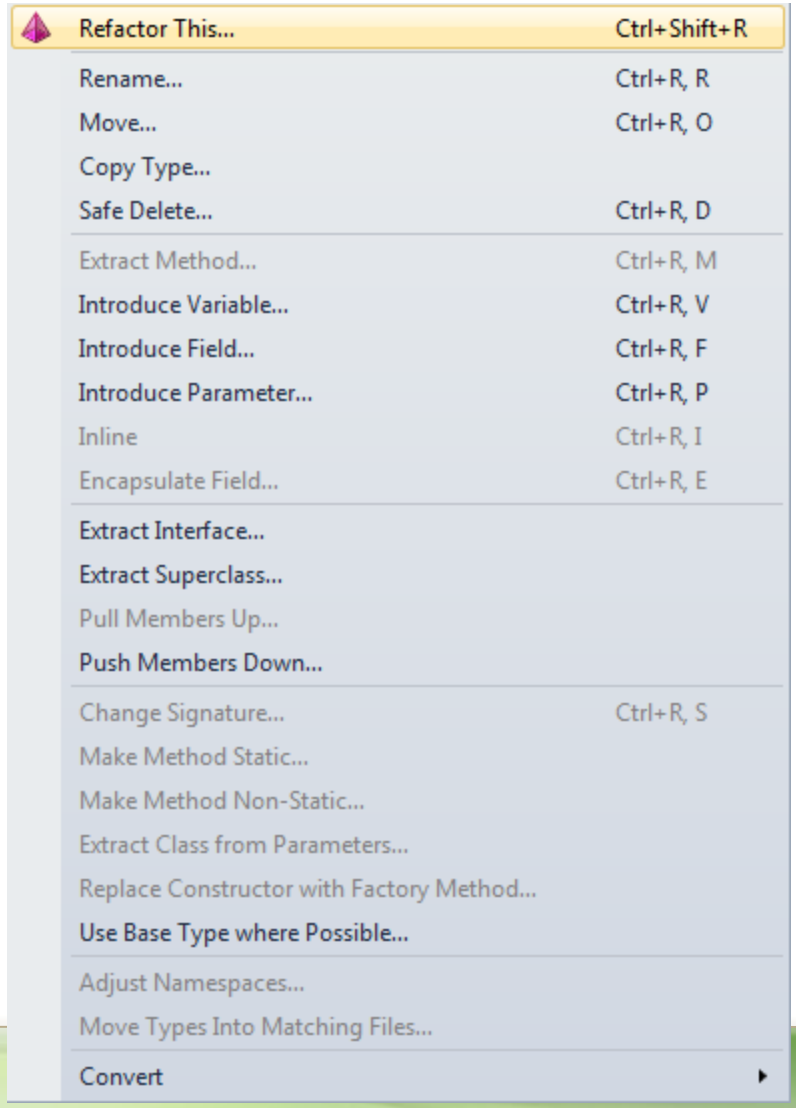


There are no unit tests for this code!

```
public IList<IList<Measurement>> Group(List<Measurement> measurements)
{
    var result = new List<IList<Measurement>>();
    int total = 0;
    while(total < measurements.Count)
    {
        var group = measurements.Skip(total).Take(_groupSize).ToList();
        result.Add(group);
        total += _groupSize;
    }
    return result;

    result.Add(group);
        total += _groupSize;
    }
    // This is a long comment to describe what happens
  // here inside the code. If you take this slide and
  // actually read this text I want you to know that
  // there is actually nothing meaningful here. It's
  // a complete copy paste of assorted code.

    var lowValue = measurements.GroupBy(m => m.LowValue)
        .OrderByDescending(g => g.Count())
        .Select(g => g.Key).FirstOrDefault();

    return new Measurement()
            {
                HighValue = highValue,
                LowValue = lowValue
            };

    var result = new List<IList<Measurement>>();

    var result = new List<IList<Measurement>>();
    int total = 0;
    while (total < measurements.Count)
    {
        .group = measurements.Skip(total).Take(_groupSize).ToList();
        t.Add(group);
        += _groupSize;

        result;
        0;
        )

        10)

        x > 3)

        if(x > 5)
        {
            return result;
        }
    }
    }

    var result = new List<IList<Measurement>>();
    int total = 0;
    while (total < measurements.Count)
    {
        var group = measurements.Skip(total).Take(_groupSize).ToList();
        result.Add(group);
        total += _groupSize;
    }
    return result;
    var highValue = measurements.GroupBy(m => m.HighValue)
        .OrderByDescending(g => g.Count())
        .Select(g => g.Key).FirstOrDefault();

    var lowValue = measurements.GroupBy(m => m.LowValue)
        .OrderByDescending(g => g.Count())
        .Select(g => g.Key).FirstOrDefault();

    return new Measurement()
            {
                HighValue = highValue,
                LowValue = lowValue
            };

    var result = new List<IList<Measurement>>();
    int total = 0;
    while (total < measurements.Count)
    {
        var group = measurements.Skip(total).Take(_groupSize).ToList();
        result.Add(group);
        total += _groupSize;
    }
    return result;
}
```

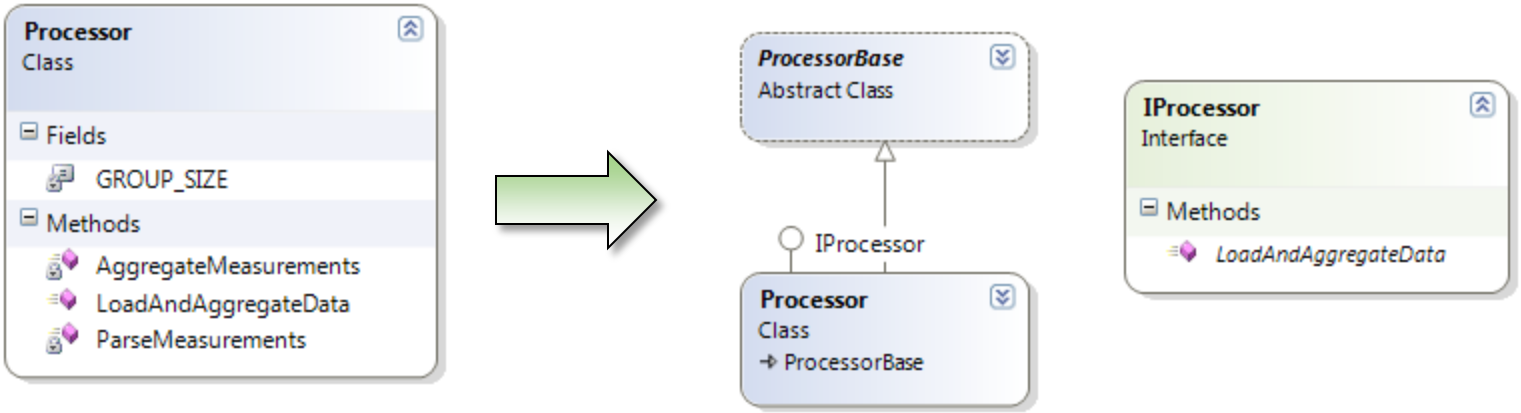
Simple Refactorings

- Rename
- Introduce Parameter
- Extract method

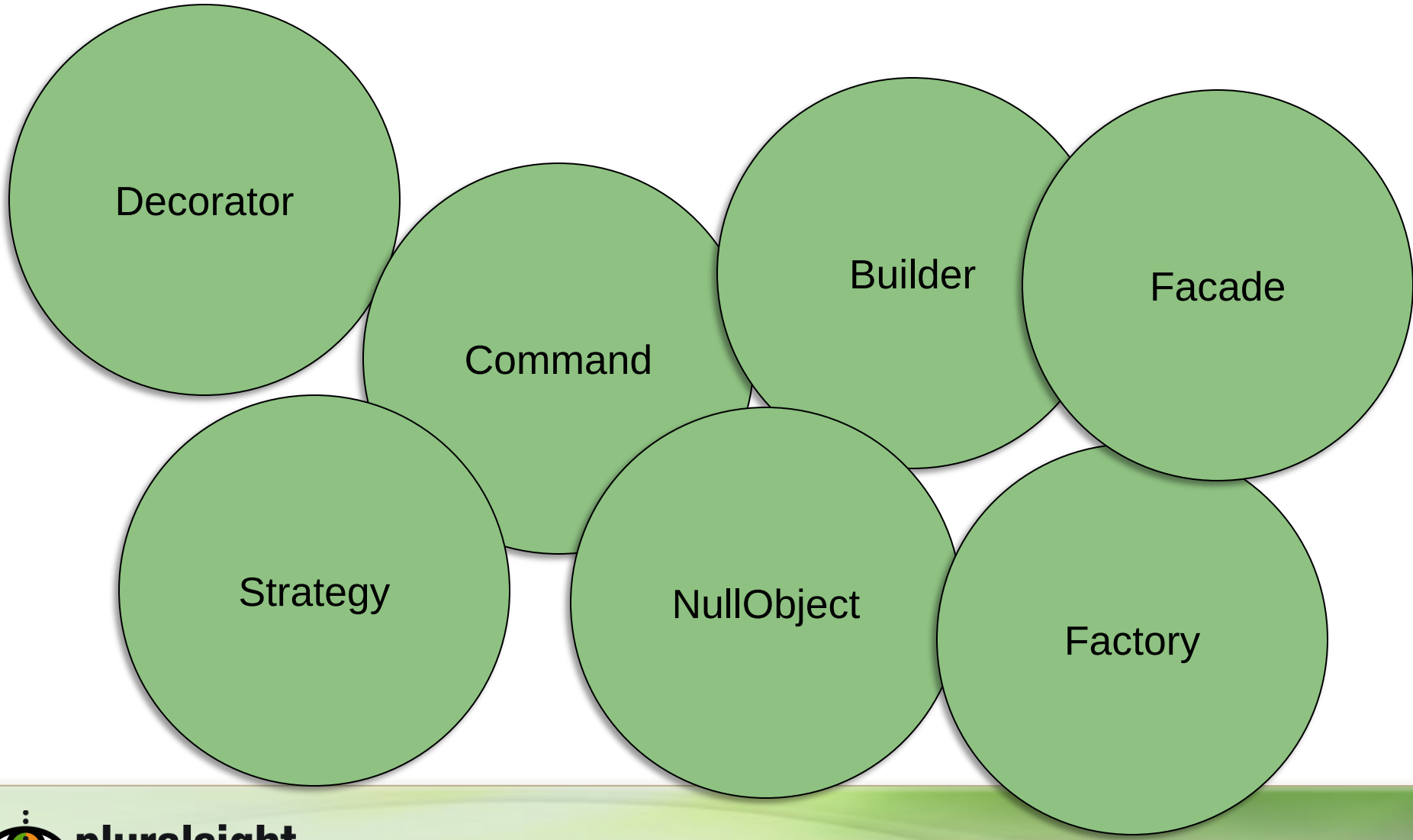


Refactoring To Abstractions

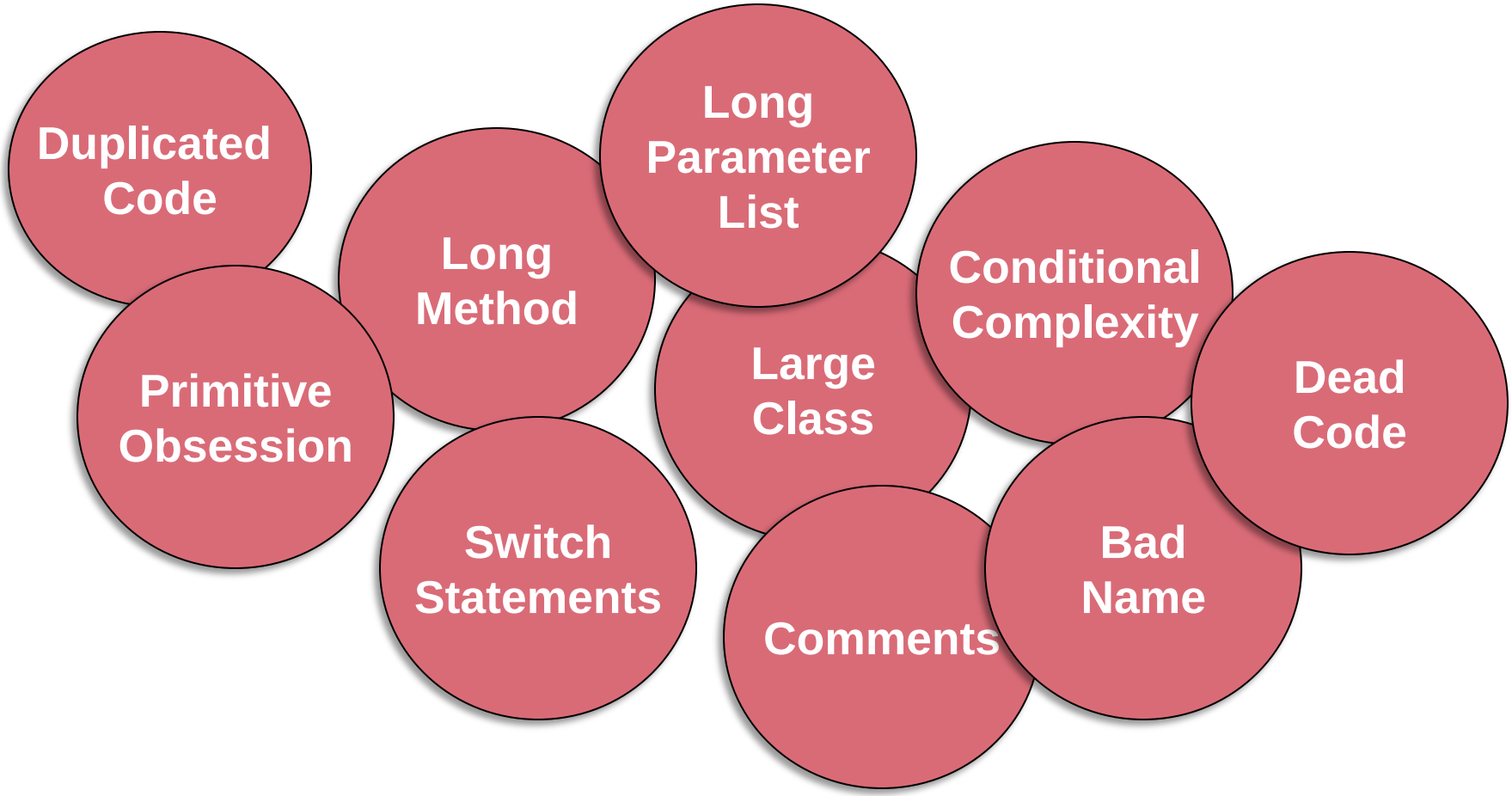
- Extract interface
- Extract superclass



Refactoring to Design Patterns



Code Smells


Duplicated Code
Long Parameter List
Long Method
Conditional Complexity
Primitive Obsession
Large Class
Dead Code
Switch Statements
Comments
Bad Name

Summary