# Refactoring Your Code – A Key Step to Agility

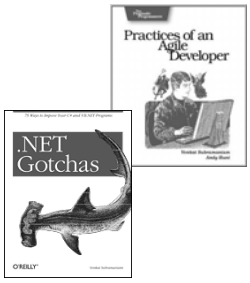
Venkat Subramaniam

```
spkr.name = 'Dr. Venkat Subramaniam'
spkr.founder = 'Agile Developer, Inc.'
spkr.affiliated = 'University of Houston'
spkr.associated = 'Rice Univ. Continuing Studies'

spkr.cred = %w{Programmer Speaker Trainer Author}

spkr.nfjs = 'NFJS Speaker since 2002'

spkr.website = 'http://www.agiledeveloper.com'
spkr.email = 'venkats@agiledeveloper.com'
```






# Abstract

Refactoring is one of the core practices in Agile Software Development. Refactoring is based on some core principles that apply to more than writing good code. But, what's refactoring? Why should you do it? How do you go about doing that? What tools are available to successfully refactor your App?

In this presentation we will address each of these questions. We will take an examples based approach to look at code that can benefit from refactoring. We will discuss how to identify a case for refactoring. Then we will use tools to help us refactor.

# Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion



# Good design vs. Over design

- Here's something from production code (changed to protect privacy!)

```
String className = System.getProperty("DataStoreFactory");
Class theClass = Class.forName(className);

IDataStoreFactory dsf
        = (IDataStoreFactory) theClass.newInstance();

IDataStore ds = dsf.create();
```

```
public IDataStore create() throws Exception
{
    DataStore1 ds1 = CreateDataStore1();

    return (IDataStore) new DataStore(ds1);
}
```

```
private DataStore1 CreateDataStore1() throws Exception
{
    String className = System.getProperty("DataStore1");
    String dbConnectionString = System.getProperty("DataBaseConnectionString");
    Object[] arguments = new Object[1];
    arguments[0] = dbConnectionString;

    Class theClass = Class.forName(className);
    Constructor constructor = theClass.getConstructor(new Class[] {String.class});
    return (DataStore1) constructor.newInstance(arguments);
}
```

## Do you see the smell in that code?

- Not quite obvious at first sight
- May make sense if extensibility is needed
- But, there were exactly one implementation of each interface (one factory, one data source, etc).
- How about the following:

```
DataStore dataStore
        = new DataStore(
              new DataStore1(
                 System.getProperty("DataBaseConnectionString")));
```



## My Code that Smells

- Let's start with an example
- Here is code that works
- What do you think about it?

## From Writing to Coding…

- William Zinsser Wrote “On Writing Well” 25 years ago!
- He gives good principles for writing well
- These principles apply to programming as much as writing non-fiction
  - Simplicity
  - Clarity
  - Brevity
  - Humanity



## Perfection

```
Perfection is achieved,
not when there is nothing left to add,
but when there is nothing left to remove.
                    - Antoine de Saint-Exupery
```

# Code Quality

```
Programs must be written for people to read,
and only incidentally for machines to execute.
                                -Abelson and Sussman
```



# Why?

- "Design, rather than occurring all up-front, occurs continuously during development."
- If the code is hard to understand, it is hard to
  - Maintain
  - improve
  - Work with for evolutionary design

## Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion



## What's Refactoring?

- "Art of improving the design of existing code"
- "A process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure"

## But, again...?

- Why fix what's not broken?
  - A software module
    - Should function its expected functionality
      - It exists for this
    - It must be affordable to change
      - It will have to change over time, so it better be cost effective
    - Must be easier to understand
      - Developers unfamiliar with it must be able to read and understand it



## You're not Refactoring if...

- You are adding new functionality
- Fixing bugs
- Making new design enhancements
- Throwing away the ?#$*! code and rewriting
- Making too many changes all at once

## Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion



## What's needed before refactoring?

- Anytime we touch code, we may break things (inadvertently)
  - You don't want one step forward and ten steps backward
- Before you refactor, make sure you have solid automated self-checking unit tests for your code
- Approach refactoring in small steps so it is easy to find bugs or mistakes you introduce

## Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion



## Points to Ponder

- Cohesion
- Encapsulation
- Don't Repeat Yourself (DRY)
- Tell Don't Ask (TDA)

## A word of Caution

- Some of the techniques, you will find, are quite opposing to other techniques
- Sometimes the wisdom tells you to go right, sometimes it tells you to go left
- You need to decide which is the right approach when



## Smells to take note of

- "Smell check" your code!
  - Duplication
  - Unnecessary complexity
  - Useless or misleading comments
  - Long classes
  - Long methods
  - Poor names for variables, methods, classes
  - Code that's not used
  - Improper use of inheritance
  - …

## Exercise on Refactor

- Let's deodorize my code!



## Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion

## Overview: Refactoring Techniques

- Several refactoring techniques exist
- You modify the code any time you think it will lead to
  - Clarity
  - Simplicity
  - Better understanding



## Composing Methods

- Long methods are problem
  - Lack cohesion
  - Too complex
- Refactoring techniques
  - Extract Method
  - Inline Method
  - Replace Temp with a Query
  - Introduce Explaining Variable
  - Replace Method with Method object
  - Substitute Algorithm

## Move Features Between Objects

- Where does this method go?
  - Often it is the (victim) class that's visible in the IDE?
  - Hard to get it right the first time
- Refactoring techniques
  - Move Method
  - Move Field
  - Extract Class
  - Inline Class
  - Hide Delegate
  - Remove Middle Man
  - Introduce Foreign Method
  - Introduce Local Extension



## Many more…

- Many more refactoring techniques than we can cover here
- Refer to Martin Fowler's celebrated book in references

# To refactor or not to refactor?

- To
  - Anytime you can cleanup the code
  - To make it readable, understandable, simpler
  - You are convinced about the change
  - Before adding a feature or fixing a bug
  - After adding a feature or fixing a bug
- Not to
  - Not for the sake of refactoring
  - When the change will affect too many things
  - When change may render application unusable
  - In the middle of adding a feature or fixing a bug
  - When you don't have unit tests to support your change



# Quiz Time

## Refactoring Your Code

- Why Refactor?
- What's Refactoring
- Before Refactoring
- Let's Refactor
- Refactoring Techniques
- Conclusion



## Conclusion

- Refactoring is a way of designing your system
- Eliminates the need for rigorous (often error prone) up-front design
- You can write some simple code and refactor
- Red-Green-Refactor is the mantra of TDD
- Leads to more pragmatic design

- Learn when to refactor and when not to

# References...


1. William Zinsser, “On Writing Well,” Collins.
2. Martin Fowler, “Refactoring: Improving the design of existing code,” Addison-Wesley.
3. Joshua Kerievsky, Refactoring To Patterns,” Addison-Wesley.
4. William C. Wake, “Refactoring Workbook,” Addison-Wesley.
5. Andrew Hunt and David Thomas, “Pragmatic Programmer,” Addison-Wesley.
6. Venkat Subramaniam and Andy Hunt, “Practices of an Agile Developer,” Pragmatic Bookshelf.


Download examples/slides from

**http://www.agiledeveloper.com/download.aspx**