



REFACTORING

A practical guide

John E. Boal
TestDrivenDeveloper.com

What is Refactoring?

- Refactor (verb)
- 1. Improve the code's internal structure without changing it's behavior
- The process:
 - Make one small change to the code that improves it. Recompile and re-test.
 - Iterate on the above over and over until there are no more improvements to make.

Why should we refactor?

- When we need to make a change, clean it up
- Clean, clear code is easier to:
 - Understand
 - Modify
 - Test
 - Maintain
- Less likely to have bugs
- Lowers risk

Why shouldn't we refactor?

- *If it works DON'T FIX IT*
- If poorly factored code works, has been tested, and does what the customer needs... *we shouldn't change it.*
- Existing code that has already been tested and is working ... if we don't need to change some functionality, we should not touch it.
- Refactoring code can introduce risk (in untested [legacy] code)
- Don't refactor it if you don't own it...

How do we know what to refactor?

- Look for *Smells* in the code.
- Something is not quite right...
- Code or Design smells are what we are looking for. These areas we need to refactor.
- Examples of some smells on next slide

Code Smells – some examples

- Method is too long (more than one screen)
- Comments
- Many parameters (more than 3)
- Duplicated code
- Conditional complexity
- Names that do not communicate intent
- Dead code (including commented code)

Gimme an example..

- `public void foo(int x)`
- `{`
- `return (x+1)*(x-1);`
- `}`

▪ Could be:

- `public void foo(int x)`
- `{`
- `return x^2-1;`
- `}`

More simple examples

- `for(int i=0; i < length; i++)`
- `{ ... }`

- `for(int arrayIndexer=0; arrayIndexer < length; arrayIndexer++)`
- `{...}`


■ And

- `public string get_setting(...)`
- `{...}`

- `public string LoadSettingFromConfigurationFile(...)`
- `{...}`



So how do we refactor?

- What does it mean to refactor code?
 - There are many techniques we can use
 - Martin Fowler has a site Refactoring.com that lists out a catalog of known techniques and simple examples
 - Here are some of the most commonly used examples:
- 

Techniques - Extract Method

- If there is functionality that is needed in more than one place, pull it out into its own method.
- If we require writing the same code more than once, we can extract that code we wrote the first time and call the new method from both places.

Techniques – Rename


- Renaming a variable, method, or even a class can make code clearer.
- Long, descriptive names are good
- If we can't name something explicitly because we can't aptly describe its function, then it probably is doing too much. Make sure to stick to the single-focus principle.

Decompose Conditional

- If we have a complicated condition in an if() statement, we can extract the calculation into a method.
- This is another case of Extract Method.



Create Parameter Object


- In a method if we have a group of parameters that naturally group together, we can create an object for these and pass that object instead.
 - Look for the “Too Many Parameters” code smell... this probably should occur if we have more than about 3 parameters on a method
- 

Replace Exception With Test

- Don't throw/catch exceptions when a simple if() statement could do the work.
- Exceptions are expensive to set up and slow down performance.
- Use a return code or value instead.
- Exceptions should only be used in exceptional conditions...



Change Constructor to Factory

- If a constructor takes parameters, or does a lot of work in creating the object, using a Factory pattern would be better.
 - A Factory class encapsulates the logic needed to build the object, so the object class can remain focused on its sole purpose.
- 

Replace Algorithm

- Replace the algorithm in the body of a method with a simpler, faster, or clearer algorithm
- This is a common refactoring.
- Make sure that there are unit tests wrapped around the method before undertaking this refactor... we don't want to change the behavior...



Many more techniques

- There are more techniques and common patterns for refactoring.
- Only the most commonly used were covered here
- Learn more about refactoring and code smells at the sites and in the books listed in the last slide



Tools

- Visual Studio, Eclipse have built-in basic refactoring tools such as:
 - Extract Method
 - Rename
- Third party tools such as ReSharper etc. offer even more refactoring tools

References

- An alphabetical list of types of refactoring:
 - <http://www.refactoring.com/catalog/>
- Code Smells
 - <http://c2.com/xp/CodeSmell.html>
 - <http://www.codinghorror.com/blog/2006/05/code-smells.html>
- Books:
 - Refactoring to Patterns by Joshua Kerievsky