Refactoring: Code Smells

Admin Notes

- REGISTER FOR BLACKBOARD
- Watch blackboard site for updates on class as hurricane season approaches

“Refactoring may be the single most important technical factor in achieving agility”

(Jim Highsmith, *Agile Software Development Ecosystems*, page 155)

“Refactoring is like continuing repair of a living system. The goal is to stay within reasonable operating limits with limited continual damage. By staying within these limits you keep costs low, because costs relate nonlinearly to the amount of repair necessary. It is like maintaining your house. You are best off (financially) if you continuously maintain rather than do large lump repairs.”

(Dirk Riehle (quoted in Jim Highsmith’s *Agile Software Development Ecosystems*, page 155))

“Refactoring keeps you ready for change by keeping you comfortable with changing your code”

(Ken Auer and Roy Miller, *Extreme Programming Applied*, page 189)

What is refactoring?

One take…

"Duplication & needless complexity are removed from the code during each coding session, even when this requires modifying components that are already "complete." Automated unit tests are used to verify every change."

(Lisa Crispin & Tip House, *Testing Extreme Programming*, page 5)

Beck's definition

"A change to the system that leaves its behavior unchanged, but enhances some nonfunctional quality – simplicity, flexibility, understandability, performance"

(Kent Beck, *Extreme Programming Explained*, page 179)

Fowler's definition

"A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior"

(Martin Fowler, *Refactoring*, page 53)

Composite Definition

Changes made to the system that

- Do not change observable behavior (all the tests still pass)
- Remove duplication or needless complexity
- Enhance software quality
- Make the code simpler and easier to understand
- Make the code more flexible
- Make the code easier to change

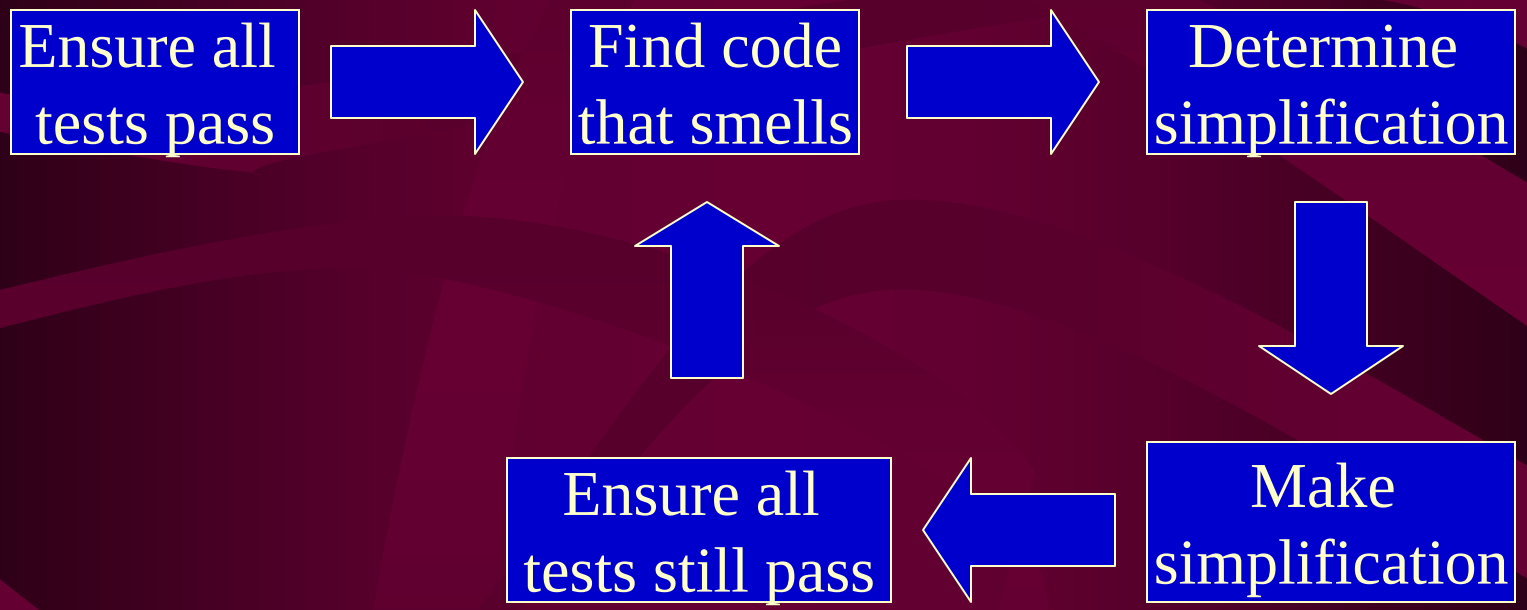
Why Refactor?

- Prevent “design decay”
- Clean up messes in the code
- Simplify the code
- Increase readability and understandability
- Find bugs
- Reduce debugging time
- Build in learning we do about the application
- Redoing things is fundamental to every creative process

How to refactor

- Make sure your tests pass
- Find some code that “smells”
- Determine how to simplify this code
- Make the simplifications
- Run tests to ensure things still work correctly
- Repeat the simplify/test cycle until the smell is gone

Refactoring Flow



Requirements for Refactoring

- *Collective code ownership*
- *Coding standards*
- Pair programming
- Simple design
- **Tests**
- *Continuous integration*
- Rested programmers

(Beck, page 66)

Where to refactor

Anywhere that needs it, provided:

- Tests exist and currently pass for the code to be refactored
- Someone else is not concurrently working in the same code
- The customer agrees that the area is worth the time and money to refactor

When to refactor

- “All the time”
- Rule of Three
- When you add functionality
- When you learn something about the code
- When you fix a bug
- When the code smells

When not to refactor

- **When the tests aren’t passing**
- When you should just rewrite the code
- When you have impending deadlines (Cunningham’s idea of unfinished refactoring as debt)

Problems with refactoring

- Taken too far, refactoring can lead to incessant tinkering with the code, trying to make it perfect
- Refactoring code when the tests don’t work or tests when the application doesn’t work leads to potentially dangerous situations
- Databases can be difficult to refactor
- Refactoring published interfaces can cause problems for the code that uses those interfaces

Why developers are reluctant to refactor

- Lack of understanding
- Short-term focus
- Not paid for overhead tasks like refactoring
- Fear of breaking current program

Code Smells

- Indicators that something may be wrong in the code
- Can occur both in production code and test code

In the following slides, the code smells and refactorings are taken from Fowler's *Refactoring*, "Refactoring Test Code" by Arie van Deursen, Leon Moonen, Alex van den Bergh, and Gerard Kok, published in *Extreme Programming* Perspectives, or David Astel's *Test-Driven Development: A Practical Guide*, as indicated on the list slides

Code smells (Fowler)

- Alternative Classes with Different Interfaces
- Comments
- Data Class
- Data Clumps
- Divergent Change
- Duplicated Code
- Feature Envy
- Inappropriate Intimacy
- Incomplete Library Class
- Large Class
- Lazy Class
- Long Method
- Long Parameter List
- Message Chains
- Middle Man
- Parallel Inheritance Hierarchies
- Primitive Obsession
- Refused Bequest
- Shotgun Surgery
- Speculative Generality
- Switch Statements
- Temporary Field

Code smells (van Deursen, et al.)

- Mystery Guest
- Resource Optimism
- Test Run War
- General Fixture
- Eager Test
- Lazy Test
- Assertion Roulette
- Indirect Testing
- For Testers Only
- Sensitive Equality
- Test Code Duplication

Code Smells (Astels)

- Inappropriate assertions
- Duplication between test method and TestCase names
- Dependent test methods

Comments

- Often used as deodorant for other smells
- Not necessarily bad in and of themselves but may indicate areas where the code is not as clear as it could be
- Refactorings
  - Extract Method
  - Introduce Assertion

Duplicated Code

- Code repeated in multiple places
- Refactorings
  - Extract Method
  - Extract Class
  - Pull Up Method
  - Form Template Method

Data Class

- A class whose only purpose is holding data
- Class has instance variables, getters, and setters
- Refactorings
  - Move Method
  - Encapsulate Field
  - Encapsulate Collection

Data Clumps

- Sets of variables usually passed together in multiple places
- Refactorings
  - Extract Class
  - Introduce Parameter Object
  - Preserve Whole Object

Inappropriate Intimacy

- Classes using too many things that should be private in other classes
- Refactorings
  - Move Method
  - Move Field
  - Change Bidirectional Association to Unidirectional
  - Replace Inheritance with Delegation
  - Hide Delegate

Large Class

- A class with too many instance variables or too much code
- Refactorings
  - Extract Class
  - Extract Subclass
  - Extract Interface
  - Replace Data Value with Object

Lazy Class

- A class that isn’t doing enough work to justify its maintenance
- Refactorings
  - Inline Class
  - Collapse Hierarchy

Long Method

- Methods with many statements, loops, or variables
- Astels defines long as > 10, Fowler doesn’t say
- Refactorings
  - Extract Method
  - Replace Temp with Query
  - Replace Method with Method Object
  - Decompose Conditional

Long Parameter List

- Many parameters passed into a method
- Refactorings
  - Replace Parameter with Method
  - Introduce Parameter Object
  - Preserve Whole Object

Middle Man

- One class simply delegates many of its requests to another class
- Refactorings
  - Hide Middle Man
  - Inline Method
  - Replace Delegation with Inheritance

Shotgun Surgery

- Making one change requires changing code in multiple places
- Refactorings
  - Move Method
  - Move Field
  - Inline Class

Switch Statements

- Using a switch statement where polymorphism would work better
- Refactorings
  - Replace Conditional with Polymorphism
  - Replace Type Code with Subclasses
  - Replace Type Code with State/Strategy
  - Replace Parameter with Explicit Methods
  - Introduce Null Object

Dealing with a Code Smell

- Pragmatic view: Analyze each one & determine whether there really is a problem
- Purist view: Get rid of it

Next Time

- Explanations of all those refactorings
- Continue with reading as in syllabus

Alternative Classes with Different Interfaces

- Methods that do the same thing but have different signatures
- Refactorings
  - Rename Method
  - Move Method

Assertion Roulette

- Multiple assertions in a test method with no explanations
- Refactorings
  - Add Assertion Explanation

Dependent test methods

- One test method depends on another to be able to run
- Refactorings
  - Make Methods Independent

Divergent Change

- One class changed in different ways for different reasons (e.g. “three methods change for a new database, while four other methods change for a new financial instrument”)
- Refactorings
  - Extract Class

Sensitive Equality

- Equality checks that are affected by non-essential things (such as using toString which may be affected by minor punctuation changes)
- Refactorings
  - Introduce Equality Method

Duplicated Naming Information

- Test methods repeat information given in the TestCase class name
- Example: public class TestCustomer extends TestCase
  - public void testCustomerName()
- Refactorings
  - Rename Method

Eager Test

- A test checks several methods of the object it’s testing
- Refactorings
  - Extract Method

Feature Envy

- A method making more use of another class than the one it is in
- Refactorings
  - Move Method
  - Move Field
  - Extract Method

For Testers Only

- Methods that exist in production code solely for the use of the tests
- Refactorings
  - Extract Subclass

General Fixture

- A test class's setUp routine contains elements only used by some of the test cases
- Refactorings
  - Extract Method
  - Inline Method
  - Extract Class

Inappropriate Assertions

- Test methods using an xUnit assert method that is not the best suited for the test
- Refactorings
  - Replace Assert

Incomplete Library Class

- Libraries that don’t provide all needed methods
- Refactorings
  - Introduce Foreign Method
  - Introduce Local Extension

Indirect Testing

- A test class testing objects other than the main one it is testing
- Refactorings
  - Extract Method
  - Move Method

Lazy Test

- Several test methods check the same method using the same fixture
- Refactorings
  - Extract Method

Message Chains

- One object asks another object for something, which causes the asked object to ask another object, and so on
- Refactorings
  - Hide Delegate

Mystery Guest

- Tests using external resources
- Refactorings
  - Inline Resource
  - Setup External Resource

Parallel Inheritance Hierarchies

- Every time you make a subclass of one class, you have to make a subclass of another
- Refactorings
  - Move Method
  - Move Field

Primitive Obsession

- Using primitive data types where classes or record types would work better
- Use small classes to better handle small tasks
- Refactorings
  - Replace Data Value with Object
  - Extract Class
  - Introduce Parameter Object
  - Replace Array with Object
  - Replace Type Code with Class
  - Replace Type Code with Subclasses
  - Replace Type Code with State/Strategy

Refused Bequest

- A class doesn’t use things it inherits from its superclass
- Refactorings
  - Replace Inheritance with Delegation

Resource Optimism

- Tests that make optimistic assumptions about existence and state of external resources
- Refactorings
  - Setup External Resource

Speculative Generality

- Making code more general in case it’s needed later
- Unused hooks and special cases make code more difficult to understand
- Refactorings
  - Collapse Hierarchy
  - Inline Class
  - Remove Parameter
  - Rename Method

Temporary Field

- Instance variables set only in certain circumstances or fields used to hold intermediate results
- Refactorings
  - Extract Class
  - Introduce Null Object

Test Run War

- Only one person at a time can run tests because they compete for access to resources
- Refactorings
  - Make Resource Unique

Test Code Duplication

- Code repeated in multiple tests
- Quite similar to Code Duplication
- Refactorings
  - Extract Method