

Refactoring 2

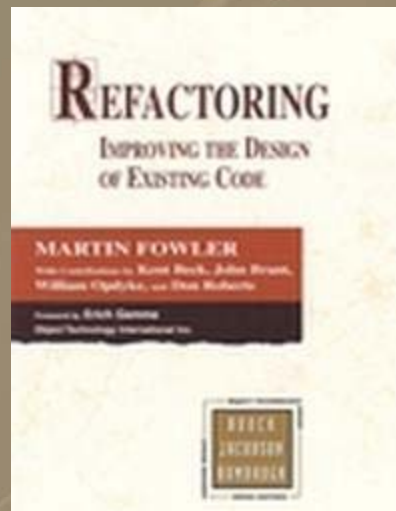
The background of the slide is an abstract composition of wavy, liquid-like patterns. The top half is a solid, medium-orange color. The bottom half features a dark brown background with intricate, shimmering patterns of lighter brown and gold, resembling oil ripples or molten metal. The overall aesthetic is modern and technical.

Admin

- Blackboard
- Quiz

Acknowledgements

- Material in this presentation was drawn from



Martin Fowler, *Refactoring: Improving the Design of Existing Code*

Refactorings (Fowler)

- Add Parameter
- Change Bidirectional Association to Unidirectional
- Change Reference to Value
- Change Unidirectional Association to Bidirectional
- Change Value to Reference
- Collapse Hierarchy
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Convert Procedural Design to Objects
- Decompose Conditional
- Duplicate Observed Data
- Encapsulate Collection
- Encapsulate Dncast
- Encapsulate Field
- Extract Class
- Extract Hierarchy
- Extract Interface
- Extract Method
- Extract Subclass
- Extract Superclass
- Form Template Method
- Hide Delegate
- Hide Method
- Inline Class
- Inline Method

Refactorings (Fowler)

- Inline Temp
- Introduce Assertion
- Introduce Explaining Variable
- Introduce Foreign Method
- Introduce Local Extension
- Introduce Null Object
- Introduce Parameter Object
- Move Field
- Move Method
- Parameterize Method
- Preserve Whole Object
- Pull Up Constructor Body
- Pull Up Field
- Pull Up Method
- Push Down Field
- Push Down Method
- Remove Assignments to Parameters
- Remove Control Flag
- Remove Middle Man
- Remove Parameter
- Remove Setting Method
- Rename Method
- Replace Array with Object
- Replace Conditional with Polymorphism
- Replace Constructor with Factory Method
- Replace Data Value with Object
- Replace Delegation with Inheritance

Refactorings (Fowler)

- Replace Error Code with Exception
- Replace Exception with Test
- Replace Inheritance with Delegation
- Replace Magic Number with Symbolic Constant
- Replace Method with Method Object
- Replace Nested Conditional with Guard Clauses
- Replace Parameter with Explicit Methods
- Replace Parameter with Method
- Replace Record with Data Class
- Replace Subclass with Fields
- Replace Temp with Query
- Replace Type Code with Class
- Replace Type Code with State/Strategy
- Replace Type Code with Subclasses
- Self Encapsulate Field
- Separate Domain from Presentation
- Separate Query from Modifier
- Split Temporary Variable
- Substitute Algorithm
- Tease Apart Inheritance

Refactorings (Testing)

- Inline Resource
- Setup External Resource
- Make Resource Unique
- Reduce Data
- Add Assertion Explanation
- Introduce Equality Method

Refactorings (Astels)

- Make Methods Independent
- Replace Assert

Consolidate Conditional Expression

- Multiple conditionals can be extracted into method
- Don't do if conditions are really independent
- Example

BEFORE

```
double diasabilityAmount() {  
    if (_seniority < 2) return 0;  
    if (_monthsDisabled > 12) return 0;  
    if (!_isPartTime) return 0;  
    if (_isVeteran) return 50;  
    // Calculate disability amount
```

AFTER

```
double diasabilityAmount() {  
    if (isNotEligibleForDisability) return 0;  
    if (_isVeteran) return 50;  
    // Calculate disability amount
```

Duplicate Observed Data

- Problem: You have data stored in a GUI component and domain methods need access
- Solution: Copy the data into a domain object (so the methods can access it) and use an Observer to keep the two locations synchronized

Extract Class

- Remove a piece of a class and make it a separate class
- Done when class is too big to understand easily or behavior is not narrowly defined enough
- Indicated by having subsets of data & methods that go together, are changed together, or are dependent on each other
- Ask "What if I removed this? What would become useless?"

Extract Interface

- Define an interface to move away from a concrete implementation
- Allows easier use of differing databases or MockObjects

Extract Method

- Pull code out into a separate method when the original method is long or complex
- Name the new method so as to make the original method clearer
- Each method should have just one task
- <http://www.refactoring.com/catalog/extractMethod.html>

Extract Subclass

- Used when a class has some behavior used for some instances and not for others
- Make a subclass that inherits the common functionality

Introduce Assertion

- Make implicit assumptions in the code explicit
- <http://www.refactoring.com/catalog/introduceAssertion.html>

Introduce Explaining Variable

- Break up complex expressions into chunks with clarifying names
- <http://www.refactoring.com/catalog/introduceExplainingVariable.html>

Introduce Parameter Object

- Replace a group of parameters that go together with an object
- Makes long parameter lists shorter & easier to understand
- Can indicate functionality to move to new class
- <http://www.refactoring.com/catalog/introduceParameterObject.html>

Preserve Whole Object

- Send the whole object rather than long lists of values obtained from the object
- May increase dependency
- A method that uses multiple values from another class should be examined to determine if the method should be moved instead
- <http://www.refactoring.com/catalog/preserveWholeObject.html>

Rename Method

- Method names should clearly communicate the one task that the method performs
- If you are having trouble naming the method, it might be doing too much. Try extracting other methods first

Replace Conditional with Polymorphism

- Replace switch statements with polymorphic subclasses (or push case behavior down to existing subclasses)
- <http://www.refactoring.com/catalog/replaceConditionalWithPolymorphism.html>

Replace Magic Number with Symbolic Constant

- Replace hard-coded literal values with constants
- Avoids duplication and shotgun surgery
- <http://www.refactoring.com/catalog/replaceMagicNumberWithSymbolicConstant.html>

Replace Nested Conditional with Guard Clauses

- In some conditionals, both paths are normal behavior & the code just needs to pick one
- Other conditionals represent uncommon behavior
- Use if/else with the normal behavior paths & guard clauses with uncommon behavior
- <http://www.refactoring.com/catalog/replaceNestedConditionalWithGuardClauses.html>

Replace Parameter with Method

- A routine invokes a method just to pass the result of that method on to another method
- Let the 2nd method call the first method directly (if it can)
- <http://www.refactoring.com/catalog/replaceParameterWithMethod.html>

DeMorgan's Law

- Used for simplifying boolean expressions
- $!(a \ \&\& \ b) \Rightarrow (!a) \ || \ (!b)$
- $!(a \ || \ b) \Rightarrow (!a) \ \&\& \ (!b)$

Further resources

- <http://www.refactoring.com>
- <http://c2.com/cgi/wiki?CodeSmell>