Using Code Metrics for Targeted Code Refactoring
Today’s Discussion

- Why do we refactor?
- How do we know when to refactor?
- Refactoring techniques
- Refactoring in the real world
Why do we refactor code?

What is the goal of refactoring?

» Reduce Complexity
Why do we refactor code?

- Why Reduce complexity?
  - Increase testability
  - Decrease maintenance
How do we traditionally know *when* to refactor?

- “Code Smell”
  - feeling = subjective
How do we traditionally know *when* to refactor?

**Example Code Smells**

- Comments
- Long Methods
- Long Classes
How can we approach refactoring *objectively*?

- Code Metrics
  - What are they?
  - Why aren’t they used often?
Cyclomatic Complexity

- Distinct paths
  - conditionals, loops
How can one determine CC?

- PMD
- JavaNCSS
- Eclipse Metrics plug-in
Rules of Thumb:

- methods \( > 10 \) = complex
- excellent coverage: 1:1 ratio of test cases to cc
False Positives:

- "update" logic
- ignore aggregates
Code Metrics

- Depth of Inheritance
  » Hierarchy tree
  ● extends clause only
How can one determine DIT?

» Eclipse Metrics plug-in
» Adana Maven plug-in
Code Metrics: Depth of Inheritance

- Rule of Thumb
  » excessive depths:
    - increase difficulty in testing
    - decrease comprehensibility
False Positives

» Exception Hierarchies usually have high DIT
Code Metrics

- Non Commenting LOC
  - Method Length
  - Class Length
Code Metrics: Method Length

- Method Length
  - 100+ NCLOC
    - too much work
    - increased complexity
Code Metrics: Class Length

- too much responsibility
- too much responsibility
- complex

Class Length

1000+ NCLOC
Code Metrics: Class Length

- Public Method Count
  » Many = lots of responsibility
  • difficult to test
Code Metrics: Class Length

- Unique Attributes
  - 45+ unique types
- Members, parameters, variables

brittle
Code Metrics: Class Length

- Excessive Imports
  - 45+ Imports
  - lots of responsibility
How to determine high LOC?

» PMD

» JavaNCSS

» Eclipse Metrics plug-in
Code Metrics: Lines of Code

- Rules of Thumb
  - High LOCs
    - complexity
    - difficulty in testing
    - brittleness
False Positives

» JavaBeans, Transfer Objects, Entity Beans, POJOs, etc

- many public methods
  - LOC is high
Code Metrics

- Rules of Thumb
  - Code metrics are objective
  - Application subjective

» High LOC != Productivity
Code Metrics

- Common Correlations
  - Long methods usually have higher CC
  - Many imports usually correlate to long class, unique members, etc
Refactoring Techniques

- Books
  - Refactoring: Improving the Design of Existing Code
  - Refactoring to Patterns
  - Working Effectively with Legacy Code

- Website
Refactoring Techniques

- Long Methods
  - Extract Method
  - Substitute Algorithm
  - Add Parameter
Refactoring Techniques

- Add Parameter

```plaintext
Before:

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>getContact()</td>
</tr>
</tbody>
</table>

After:

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>getContact(Date)</td>
</tr>
</tbody>
</table>
```
Refactoring Techniques

- Long Classes
  - Extract Class
  - Extract Subclass
    - Push Down Method
    - Form Template Pattern
Refactoring Techniques

- Extract Class

<table>
<thead>
<tr>
<th>Person</th>
<th>1</th>
<th>Telephone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td>areaCode number</td>
</tr>
<tr>
<td>officeAreaCode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>officeNumber</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getOfficeNumber</td>
<td></td>
<td>getTelephoneNumber</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>officeTelephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>getTelephoneNumber</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Telephone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>areaCode number</td>
</tr>
<tr>
<td>getTelephoneNumber</td>
</tr>
</tbody>
</table>
Refactoring Techniques

- Depth of Inheritance
  - Replace Inheritance with Delegation
  - Pull up and Push down Method
Refactoring Techniques

- Replace Inheritance with Delegation
Refactoring Techniques

- Pull up Method
Refactoring Techniques

- Push Down Method

Before:
- Employee
  - getQuote
  - Salesman
  - Engineer

After:
- Employees
  - Salesman
    - getQuote
  - Engineer
Refactoring Techniques

- Too Many Imports
  - Extract Class
Refactoring Techniques

- Cyclomatic Complexity
  - Replace Conditional with Polymorphism
  - Extract Method
Refactoring for Real

- Find Complexity
  - Run PMD
    - visual confirms added maintenance issue
Conditionals

» Keep growing

» Affect testability
Refactoring for Real

Solution

» Replace Conditionals with Polymorphism

» Extract Class

» Liberal interpretation of GOF patterns
Refactoring for Real

- GOF Patterns
  - Abstract Factory
  - Chain of Responsibility
  - Strategy
Refactoring for Real

Benefits?

» Plug-ability

» Testability
  ◆ pushed complexity into manageable pieces
Review of Today’s Discussion

- Why do we refactor?
- How do we know when to refactor?
- Refactoring techniques
- Refactoring in the real world
Thank you!

Questions? Comments?
Please Contact:
Andrew Glover
Telephone  (866) 682-9259
or
aglover@vanwardtechnologies.com