Aspect-Oriented Refactoring

Ran Ettinger
PRG Student Conference
24 June 2003
Contents

★ 1 claim
★ 4 definitions (D1-D4)
★ 4 questions (Q1-Q4)
★ 3 promises (P1-P3)
★ 1 example
Claim: the quality of existing software can be improved by the introduction of aspect-oriented constructs
Q1: Software Quality ?
D1: software configurability

The ability to build, release and evolve a software system in a number of different configurations.
Q2: How can high configurability improve software quality?
Example – Summary of Integer Binary Trees
Standard Edition

Summary of the tree:

- sum is 21
- size is 4
Standard <Debug> Edition

8
--- 7
--- 5
-----
----- 1

Summary of the tree:

sum is 21
size is 4
Summary of the tree:
sum is 21
size is 4
max is 8
height is 2
```java
void computeSummary(Node node) {
    int leftHeight = 0;
    height = 0;
    if (node.left != null) {
        computeSummary(node.left);
        leftHeight = ++height;
    }
    if (node.right != null) {
        computeSummary(node.right);
        if (leftHeight > ++height)
            height = leftHeight;
    }
    size++;
    if (node.val > max)
        max = val;
    sum += val;
}
```
Q3: What are the OO configurability mechanisms and techniques?
Pre-compiler directives

```c
void computeSummary(Summary summary) {
    #ifdef HEIGHT
        int leftHeight=0;
        height = 0;
    #endif
    if (node.left != null) {
        computeSummary(node.left);
        #ifdef HEIGHT
            leftHeight = ++height;
        #endif
    }
    if (node.right != null) {
        computeSummary(node.right);
        #ifdef HEIGHT
            if (leftHeight > ++height) height = leftHeight;
        #endif
    }
} ...
```

Aspect-Oriented Refactoring
void computeSummary(Summary summary) {
#ifdef HEIGHT
    int leftHeight = 0;
    height = 0;
#endif
    if (node.left != null) {
        computeSummary(node.left);
#ifdef HEIGHT
        leftHeight = ++height;
#endif
    }
    if (node.right != null) {
        computeSummary(node.right);
#ifdef HEIGHT
        if (leftHeight > ++height) height = leftHeight;
#endif
    }
} …
#define DEFINE HEIGHT

Aspect-Oriented Refactoring
Subclassing

```java
void computeSummary(Node node) {
    if (node.left != null) {
        computeSummary(node.left);
    }
    if (node.right != null) {
        computeSummary(node.right);
    }
}

class Max extends Summary {
    void computeSummary(Node node) {
        super.computeSummary(summary);
        if (node.val > max)
            max = val;
    }
}
```
Subclassing

Sum

Size

Max

Height

Sum Size

Size Max

Max Height

Sum Size Max

Size Max Height

Sum Size Max Height

Aspect-Oriented Refactoring
D2: Aspect-Oriented Programming (AOP)

A novel programming paradigm that allows clean modularisation of crosscutting concerns, such as debugging and profiling, in software design.
Contents (Revisited)

- 1 claim {2}
- 4 definitions {5, 17, 24, 25}
- 4 questions {4, 6, 12, 19}
- 3 promises {23, 29, 30}
- 1 example {...}
Q4: In what way does AOP promote configurability?
void computeSummary(Node node) {
    if (node.left != null) {
        computeSummary(node.left);
    }
    if (node.right != null) {
        computeSummary(node.right);
    }
}

package MaxBinaryTree;
class Summary {
    int max;
    void computeSummary(Node node) {
        // after
        if (node.val > max)
            max = val;
    }
}
AO Solution – Feature.Max

-hyperspace
  hyperspace BinaryTreeSummaryHyperspace
  composable class objectModel.*;
  composable class treeSummary.*;
  composable class treeSummary.max.*;

-concerns
  package objectModel : Feature.Kernel
  package treeSummary : Feature.TreeSummary
  package treeSummary.max : Feature.Max

-hypermodules
  hypermodule Max
    hyperslices:
      Feature.Kernel,
      Feature.TreeSummary,
      Feature.Max,
    relationships:
      mergeByName;
  end hypermodule;
## AO Solution Vs. OO

<table>
<thead>
<tr>
<th></th>
<th>Scattered</th>
<th>Tangled</th>
<th>Run-time overhead</th>
<th>Code size overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ifdef</td>
<td>+++</td>
<td>+++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subclassing</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+++</td>
</tr>
<tr>
<td>Decorator/Strategy</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>AO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Aspect-Oriented Refactoring
P1: In this talk, I will demonstrate how to perform refactoring of OO software systems into AO, while arguing for the gained quality.
D3: Refactoring

The process of improving the design of existing software by performing small and well defined source code transformations, without modifying the original functionality.
D4: Program Slicing

The process of determining which statements are contributing to a computation.
void computeSummary(Node node) {
    int leftHeight = 0;
    height = 0;
    if (node.left != null) {
        computeSummary(node.left);
        leftHeight = ++height;
    }
    if (node.right != null) {
        computeSummary(node.right);
        if (leftHeight > ++height)
            height = leftHeight;
    }
    size++;
    if (node.val > max)
        max = node.val;
    sum += val;
}
void computeMax(Node node) {
    computeSummary(root);
    max = computeMax(root);
}

void computeMax(Node node) {
    if (node.left != null) {
        computeSummary(node.left);
    }
    if (node.right != null) {
        computeSummary(node.right);
    }
    if (node.val > max) {
        max = node.val;
    }
}
void computeSummary(Node node) {
    if (node.left != null) {
        computeSummary(node.left);
    }
    if (node.right != null) {
        computeSummary(node.right);
    }
}

package MaxBinaryTree;

class Summary {
    int max;
    void computeSummary(Node node) { // after
        if (node.val > max)
            max = node.val;
    }
}
P2: The *Extract Slice* approach (both the OO and AO versions) suffers from several deficiencies that will be reported.
P3: A variation on this approach will be presented, one that extracts each code fragment of the slice separately.
void computeSummary(Node node) {
    int leftHeight=0;
    height = 0;
    if (node.left != null) {
        computeSummary(node.left);
        leftHeight = ++height;
    }
    if (node.right != null) {
        computeSummary(node.right);
        if (leftHeight > ++height)
            height = leftHeight;
    }
    size++;
    if (node.val > max)
        max = node.val;
    sum += val;
}
void computeSummary(Node node) {
    HeightComputation heightComputation =
    new HeightComputation(this);
    if (node.left != null) {
        computeSummary(node.left);
        heightComputation.computeHeightLeft();
    }
    if (node.right != null) {
        computeSummary(node.right);
        heightComputation.computeHeightRight();
    }
    size++;
    if (node.val > max)
        max = node.val;
    sum += node.val;
}
class HeightComputation
    
    private int leftHeight=0;
    public HeightComputation(Summary summary) {
        summary.height = 0;
    }
    void computeHeightLeft() {
        leftHeight = ++summary.height;
    }
    void computeHeightRight() {
        if (leftHeight > ++summary.height)
            summary.height = leftHeight;
    }

void computeSummary(Node node) {
    if (node.left != null) {
        computeSummaryLeft(node);
    }
    if (node.right != null) {
        computeSummaryRight(node);
    }
}

void computeSummaryLeft(Node node) {
    computeSummary(node.left);
}

void computeSummaryRight(Node node) {
    computeSummary(node.right);
}
package treeSummary.height;
public class Summary {
    int height;

    void computeSummary(Node node) {
        node.leftHeight = 0;
        height = 0;
    }

    void computeSummaryLeft(Node node) {
        node.leftHeight = ++height;
    }

    void computeSummaryRight(Node node) {
        if (node.leftHeight > ++height)
            height = node.leftHeight;
    }
}

Fragmented AO Extract Slice - 2
Feature.Height

-concerns
    package objectModel : Feature.Kernel
    package treeSummary : Feature.TreeSummary
    package treeSummary.height : Feature.Height

-hypermodules
    hypermodule SumSizeMaxHeight
        hyperslices:
            Feature.Kernel,
            Feature.TreeSummary,
            Feature.Height;
        relationships:
            mergeByName;

    order action Feature.Height.Summary.computeSummary before
    action Feature.TreeSummary.Summary.computeSummary;

end hypermodule;
Conclusions

⋆ Existing OO software can be improved by (slicing-based) refactoring into AO

⋆ Aspect-oriented refactoring tools will facilitate AOP adoption