Lecture 7

Aspect-orientation (AO*)

A new paradigm in Software Engineering

Copyright © Yijun Yu, 2005
Last lecture and tutorial …

Software Quality Measurements

• We have shown the use of quality measurements to monitor the progress of software development

• The development/restructuring (maintenance) activities (refactoring, tuning, adding features) can be guided by the metrics of softgoals
Today …

On Aspect Orientation

• Today we explain the paradigm of aspect-orientation

1. Concepts: What are aspects?
2. Practices: Aspect-orientation at large
   – AOP: Aspect-oriented programming
   – AOSD: Aspect-oriented software development
   – AORE: Aspect-oriented requirements engineering
   – AOSR: Aspect-oriented software reuse (probably next lecture)

3. A case study of AORE
4. Summary
1. What are aspects?

1. Some design principles
   • Divide and conquer: problem solving/design principle
   • Modularization: high cohesion/low coupling
     Separation of concerns
   • DRY: Don’t Repeat Yourself
     Increase the fan-in

2. Previous paradigms
   • 70s – 80s:
     Structured programming (Goto’s considered harmful) =>
     Structured Analysis, Structured Design
   • 80s – 90s:
     Object-oriented programming (OOP) =>
     OOA/OOD => UML

3. Why another paradigm?
   • Since late 90s …
     Separation of the crosscutting concerns

4. What are aspects?
   – Modularizing the crosscutting concerns
1.1 Some design principles

Structured programming

- What is structured program?
  - A program has no more GOTO's
  - Only three kinds of structure prevails
    - Sequential
    - If-then-else
    - Loops
    [Dijkstra: Goto considered harmful]
  - In other words, every statement block has single-entry, single-exit as Hammock Graph
    [Weiser: Program slicing]
- “Whenever possible, we wish to maximize fan-in during the design process. Fan-in is the raison d'être of modularity. Each instance of multiple fan-in means that some duplicate code has been avoided.”
  raison d'être: grounds for existence
  (http://www.french-linguistics.co.uk/dictionary/)
  [Yourdon & Constantine79] Structured Design (pg. 172, see also http://wwwpa.win.tue.nl/wstomv/quotes/structured-design.html)
  [parnas: Modularization, information hiding]
(1) A decomposition hierarchy from abstract to concrete: Divide and Conquer, Structured Design;
(2) Don’t Repeat Yourself, Factoring / Refactoring …
Example

- VALUE
- MOVE UNTIL
- FIND
- SCAN
- REMOVALL
- STRINGCOMP

Yourdon & Constantine, SD, pg.168
1.1 Some design principles

Object-oriented programming

- Everything is an object (Smalltalk)
- Information hiding / Encapsulation: object groups related data and the operations on the data into a module
- Object has structural relationships:
  - inheritance: generalization / specialization: isA/instanceOf
  - aggregation: hasA / isPartOf
  - associations: 1-to-many, 1-to-1, many-to-many
- In the end, the structurally-related objects are packaged into components
1.2 Aspect-orientation

• Component language
  (any structured or OO language, even corresponding design and requirements specification)

• What are crosscutting concerns?

• An aspect language
  – What are joinpoints?
  – What are pointcuts?
  – What are advices?

• A weaving mechanism
Aspect concepts

- Concepts: cross-cutting, component, aspect, join points, weaving

AOP hides the join points
AOP (THE MAGIC)

VALUE

FIND

SCAN

MOVE UNTIL

REMOVALL

STRINGCOMP
AOP (NOT REALLY MAGIC)

FIND: AT LINE 5
VALUE: AT LINE 7
SCAN: AT LINE 15
MOVE UNTIL: AT LINE 8
REMOVALL: AT LINE 2
AOP example
Stan Wagon’s bike

My square-wheel bike, on permanent display at Macalester College. This construction, believe it or not, earned me an entry in "Ripley’s Believe It or Not"; beats standing in a block of ice for three days or growing three-foot long fingernails.

--
http://www.stanwagon.com
Stan Wagon
(wagon@macalester.edu), Prof. of Mathematics and Computer Science, Macalester College, St. Paul, Minnesota
The Weaver

FIND: AT LINE 5
VALUE: AT LINE 7
SCAN: AT LINE 15
MOVE UNTIL: AT LINE 8
REMOVALL: AT LINE 2

STRINGCOMP

VALUE

MOVE UNTIL

REMOVALL
aspect Logging {
  pointcut NeedLogging():
    call(\text{void FIND()} ) ||
    call(\text{void MOVEUNTIL()} ) ||
    call(\text{void REMOVALL}() ) ||
    call(\text{void SCAN()} ) ||
    call(\text{void VALUE()});

  after() returning: NeedLogging() {
    \text{STRINGCOMP}();
  }
}

AspectJ

aspect

The DRIVER

The GROUND

pointcut

The WAGON

advice
2. Aspect-orientation at large

2.1 Aspect-oriented Programming

- It permeates into almost every popular high-level programming languages
  
  - Java
    - Hyper/J, AspectJ, AJDT, JBoss
  
  - C/C++/C#
    - AspectC/C++, C#
  
  - PHP
    - AOPHP, AspectPHP

... and many many more: see AOSD.NET
Every AOP mechanism has to support

• Definition and representation of aspects
  – Definition of Advices in the component language
  – Definition of Joinpoints in regular expressions
    • Optionally, they can introduce new data members, changing the structures of components
  – Representation: New keywords, New directives, XML, but never change the code of components directly

• Implementing a weaver
  – As preprocessor => generates woven components in the component language (AspectC, AOPHP)
  – As instrumenting compiler => generates woven components in the bytecode for the languages supporting reflection (AspectJ)
  – As interpreterator => interpreting the woven code on-the-fly (AspectPHP)
2. Aspect-orientation at large

2.2 Aspect-Oriented SD

- AO includes the whole lifecycle of SE
  - http://www.aosd.net
- There is a conference AOSD
- There are workshops on Early Aspects at AOSD, OOPSLA, ICSE
- Hot topics related to all other SD technologies
  - Aspect-oriented Refactoring
  - Aspect Mining
  - Aspect-oriented Debugging
  - Aspect-oriented Testing
  - Aspect-oriented Slicing
  - Aspect-oriented Model Checking

...
2. Aspect-orientation at large

2.3 Aspect-Oriented RE

- Lessons learnt from success stories
  - SP => SA
  - OOP => OOA
  - Why not AOP => AOA?
    - Separation of crosscutting concerns earlier
    - Avoid duplication as early as possible
    - Identify aspects before mining them from code

- Discover aspects in the early requirements
  - From structured requirement documents
  - From unstructured (textual) documents

- Verify discovered (candidate) aspects in AOP
3. A Case Study on AORE

1. Quickly go through goal-oriented requirements engineering basics
2. A requirements engineering process to elicit early aspects (goal aspects)
3. A reverse engineering exercise to identify candidate aspects (code aspects)
4. Linking goal aspects with code aspects
3.1 Requirements Goal Models

• A goal model is an intentional model
• A goal can be decomposed into AND or OR subgoals
• A goal model has both hard and soft goals
  – A hard goal can be either satisfied or denied
  – A soft goal is partially satisfied => satisficed
• Soft goal uses HELP (+), HURT (-), MAKE (++) or BREAK (--) correlations to show partial satisfaction (satisfice) from a set of subgoals
3.1.1 Hard goal model

- AND decomposition
- OR decomposition

- Schedule meeting
- Generate schedule
  - Interactively
  - Manually Automatically

- Collect constraints
- Collect other constraints

- Collect timetables
- Share timetables

- Person collects
- From all
  - From Initiator only
  - By all means
  - By email

- System collects

© 2004 John Mylopoulos

Intentions and Agents – 6
3.1.2 Soft goal model
3.1.3 Goal-Oriented Requirements Analysis

Goal: Contact a Friend
Goal: Email a Friend
Goal: Mail a Friend
Goal: Call a Friend

FS: Fully satisfied
PS: Partially satisfied
UN: Unknown
PD: Partially denied
FD: Fully denied
CF: Conflict

++: MAKE
+: HELP
--: BREAK

OR

AND

Text Editor
SMTP

Software Engineering II
FS: Fully satisfied
PS: Partially satisfied
UN: Unknown
PD: Partially denied
FD: Fully denied
CF: Conflict

T: satisfied
F: denied
U: unknown

Goal: Contact a Friend
Goal: Email a Friend
Goal: Mail a Friend
Goal: Call a Friend

Text Editor
SMTP

++: MAKE
+: HELP
--: BREAK

OR
AND

Goal: Get Reliable Reply
3.1.4 V-graph

In order to reason about interplay of functional and non-functional requirements, we create a particular type of goal model, called *V-graph*
3.2 The Process

- Start from root-level goals and soft goals, correlate and decompose them into a V-graph
- A goal analysis based on the label propagation algorithm is used to check for:
  - Conflicts
  - Inconsistencies
  - Denial of any goal or soft goals
- After resolving the problems, a proper V-graph is obtained
- Then we list the candidate aspects from the V-graph
3.3 A Case Study

- Medi@Shop adapted from literature: 
  *Can we find aspects from early requirements?*

- osCommerce studied from an LAMP (Linux, Apache, MySQL, PHP) Open-Source project: 
  ([http://www.oscommerce.com](http://www.oscommerce.com)) 
  *Do they manifest in the developed software?*
osCommerce (version 2.2m2)
Duplications in code
## Candidate code aspects in the code

Clone detection (by Semantic Design, Inc)

<table>
<thead>
<tr>
<th>LOC</th>
<th>#clones</th>
<th>Code description</th>
<th>Need refactoring?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>319</td>
<td>require($path . $file);</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>260</td>
<td>echo $expression;</td>
<td>No</td>
</tr>
<tr>
<td>559</td>
<td>2</td>
<td>class email;</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>292</td>
<td>define ($variable, $value);</td>
<td>No</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
<td>class mime;</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>67</td>
<td>messageStack-&gt;add ($error);</td>
<td>Yes (NFR)</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>Postal code zone check</td>
<td>Yes (FR)</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>require(application_top.php); SSL check</td>
<td>Yes (FR/NFR)</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>Set HTML head CHARSET</td>
<td>Yes (NFR)</td>
</tr>
</tbody>
</table>
3.4 Identifying goal aspects
Correlate initial goals and softgoals
Inconsistent decomposition
Resolving inconsistency
Further decomposition
Resolving Conflicts
Result candidate aspects
Goal Aspects

goal aspect Responsiveness[transaction] {
    pointcut transaction():
        Preparing[cart, product]) ||
        CheckingOut[cart, product, account, stock]);
    required () by: transaction() {
        SessionCookie[transaction]();
    }
};

• AspectJ-like syntax
• Allow weaving the operationalized tasks with goals specified in the pointcut
Your exercise

• Reverse Engineering
  Identify some aspects in the OpenOME
  – Clone-detection or Callgraph extraction
  – Goal analysis

• Forward Engineering
  – Implement some new NFR through AspectJ
4. Summary

- The concepts of aspect-orientation
- The practise of AOP, AOSD, AORE, AOSR
- A Case study of AORE
Further readings


What’s next …

• A tutorial on aspect-oriented programming tools
  – AspectJ
  – Eclipse/AJDT
  – Visualizing Aspects
  – Aspect mining tool

• A lecture on (aspect-oriented) Software Reuse
  – Q7 in the OpenOME