Tool-Supported Detection of Code Smells in Software Aspectization

Péricles Alves and Diogo Santana

Agenda

• Recurring Mistakes
• Code Smells
• ConcernReCS Tool
• ConcernReCS Extension

Motivation

• AOP is about 15 years around and it is not widely adopted by industry
• Refactoring OO code to aspects is an error-prone task
• Lack of tools to help developers when using AOP

Classification of Recurring Mistakes

• Experimentally found in our previous work [1]: 80 groups of participants were asked to refactor to aspects crosscutting concerns in three small Java applications
• Mistakes made by novice programmers in AOP
• Focuses in the two basic features of AO languages: Advices and pointcuts

Mistakes in Software Aspectization

Advice-related mistakes

• Incorrect Implementation Logic: Part of the base code is refactored out to an advice. However, it produces an unexpected behavior
• Incorrect Advice Type: The programmer uses an incorrect type of advice to implement part of the concern

Mistakes Related to Compilation

- **Compilation Error or Warning:**
  A syntactic or potential fault signaled by the AspectJ compiler through an error or warning message

Mistakes Related to the Base Code

- **Duplicated Crosscutting Code:**
  Part of the code that implements a crosscutting concern appears replicated in both aspects and the base code
- **Incomplete Refactoring:**
  The developer do not refactor part of the crosscutting concern
- **Excessive refactoring:**
  Part of the base code that does not belong to a concern is refactored to aspects

Code Smells in Software Aspectization

Catalog of Code Smells

- Scenarios in OO source code that seem to lead to mistakes during software aspectization
- **Observed by:**
  Inspecting the code of 80 AO implementations and interviewing some participants of our previous study

**Primitive Constant**

- **Definition:**
  A concern has one or more constant attributes of a primitive type among its implementation elements
- **Mistake:** Duplicated crosscutting code
- **Example:**
  ```java
  public class ATM {
  // constants corresponding to menu options
  private static final int MSG_O_ = 0;
  ...
  }
  ```

**Attribute of a Non-Dedicated Type**

- **Definition:**
  A concern has one or more class attributes of non-dedicated types
- **Mistake:** Incomplete refactoring
- **Example:**
  ```java
  public abstract class Connection {
  Timer timer; new Timer();
  protected long totalConnectionTime = 0;
  ...
  }
  ```
Element out of Inheritance Tree

- **Definition:**
  There is concern code of the same kind in all leaves of an inheritance tree, in some other tree nodes (not leaves) and in other classes of the system
- **Mistake:** Incomplete refactoring
- **Example:**

The ConcernReCS Tool

ConcernReCS

- Eclipse plug-in
- Extension of ConcernMapper
- **Goal:**
  Automatically find and report error-prone scenarios (code smells) in OO source code before the concerns have been refactored out to aspects

Mapping Concerns

- The first step in order to use ConcernReCS is to create a concern map using the ConcernMapper
- Should be added to the map only those elements, such as classes, methods and attributes, that are completely dedicated to the concern

Concern Map Expansion

- Code elements partly dedicated to a concern are automatically inferred based on:
  - calls to mapped methods and
  - read/write access to mapped attributes
- Concern code of any other kind should be refactored using the Extract Method refactoring and the resulting method added to the concern map

Example of a Mapped Concern

Map of the `ErrorMessage` concern in the Chess application (used in our previous study)
User Preferences

• In the preference page is possible to select:
  – Which code smells are presented
  – Which concerns are enabled

Finding Code Smells

• Based on the concern map, ConcernReCS runs a series of detection strategies

• Examples of detection strategies
  – Find constants implementing a concern
  – Infer elements in an inheritance tree that partially implement a concern

• The result is a set of code smells which can potentially lead to mistakes during the concern aspectization

Error-Proneness

• For each code smell, ConcernReCS gives a number ranging from 0 to 1 which indicates how prone developers are to make a mistake when it appears

• This number is based on the percentage of participants in our previous study who made mistakes related to each code smell

• The nearest to 1 the error-proneness is, the more prone to make mistakes developers are

Finding Code Smells

• Simplified flow chart

Code Smells Visualization

• For each code smell, ConcernReCS presents:
  – The smell name
  – The refactoring mistake it may leads to
  – The concern in which it appears
  – The source file
  – The specific code element in which it is located
Code Smells Visualization

- ConcernReCS main view

Related Work

- Nunes et al. [1] have presented mistakes in concern mapping tasks, which can lead to aspectization mistakes, such as *Incomplete* and *Excessive Refactoring*.

- Macia et al. [2] have pointed out that systematic removal of code smells can be used to effectively combat symptoms of architecture degradation.

Ongoing Work

The ConcernReCS Tool Extension

- **Goal:** automatically find some Fowler's Bad Smells by the implementation of all metrics used in Juliana's work.

- **Bad Smells:** Divergent Change, Shotgun Surgery and Long Class.

About the extension

- It will share some Concern-ReCS's modules, such as its interface.

Motivation

- Improve software quality and maintainability.

- Apply the most useful and recommended metrics to detect Fowler's bad smells.

The tool extension diagram
Metrics component

Most useful metrics

• Divergent Change: NCC.

• God Class: CBO, NOM, LCOM, NOA, WMC, LOC, CDLOC and CDO.

• Shotgun Surgery: inconclusive.

Related Work

• There can be found a wide literature related to software metrics tools.

• Almost all metrics were found in others tools, except NCC and CDLOC.

<table>
<thead>
<tr>
<th>Tool</th>
<th>ConcernMorph</th>
<th>ConcernTagger</th>
<th>Metrics 1.3.6</th>
<th>QM/M</th>
<th>AVAIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Useful metrics</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Status

• Review all available literature to find the missing metrics and learn new ways to implement the already found ones.

• Adapt the found code to ConcerReCS or create interfaces with others tools.

Future Work

• Tool evaluation:
  Evaluate the tool in large-scale software projects in order to investigate its scalability and eventually improve it

Results


Acknowledgments

• FAPEMIG (Péricles) and CNPq/PIBITI (Diogo)