Dependency Cycles

Master Project
Bledar Aga
University of Bern

Software Composition Group
Spring 2013
Content

- Introduction
- Problem
- Dependency types
- Approaches
- Concrete example
- Existing approaches
- Existing solutions
- Our solution
- Refactoring
- Best refactoring
- Profit
- Best profit example
- Next steps
- Summary
Introduction

Test and release with minimal amount of work
Problem

- Compromises the modularity of the system
- Increase the development time
- Prevents proper reuse
Dependency types

- **Inheritance**: Extends or implement

- **Reference**: `new Object();`

- **Invocation**: `object.equals();`

- **Extension**: Method defined in a package, class is defined in different package
Approaches

✦ Move class
✦ Move method
✦ The use of design principles and patterns
  ◦ Dependency Inversion Principle, DIP
  ◦ Dependency Injection Pattern, DI
  ◦ Combination of both DIP + DI
Concrete example
Concrete example

Inheritance

Method invocation
Move class example
Move method example

Move method example
Design pattern example

Dependency Injection

Dependency Inversion
# Existing approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Type</th>
<th>Inheritance</th>
<th>Reference</th>
<th>Invocation</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move class [1]</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Move method [1]</td>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DIP [2]</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DI [3]</td>
<td></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DIP + DI</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

[1] Janik Laval - Package Dependencies Analysis and Remediation in Object-Oriented Systems  
[2] Robert C. Martin - Design Principles and Design Patterns  
[3] Martin Fowler - Inversion of Control Containers and the Dependency Injection pattern
## Existing solutions

<table>
<thead>
<tr>
<th>Tool</th>
<th>Type</th>
<th>Analysis</th>
<th>Refactoring</th>
<th>Multi platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>JooJ [1]</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ByCycle [2]</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Lattix LDM[3]</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Pasta [4]</td>
<td></td>
<td>Yes</td>
<td>MC</td>
<td>No</td>
</tr>
<tr>
<td>Stan [5]</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Structure101 [6]</td>
<td></td>
<td>Yes</td>
<td>MC, MM</td>
<td>Yes</td>
</tr>
<tr>
<td>EDSM [7]</td>
<td></td>
<td>Yes</td>
<td>MC, MM</td>
<td>Yes</td>
</tr>
</tbody>
</table>

[1] JooJ: Real-time Support for Avoiding Cyclic Dependencies
[3] Lattix LDM: Using dependency models to manage complex software architecture
[4] Pasta: Improving Java software through package structure analysis
Our solution
Refactoring

refactor(model){
    sccs = findSccs(model) // use Tarjan's algorithm
    orderScc(sccs) // min # of cycle
    WHILE sccs NOT EMPTY {
        scc = sccs.getFirstElement()
        WHILE scc NOT EMPTY {
            orderCycle(scc) // min # of arcs
            cycle = scc.getFirstElement()
            WHILE cycle NOT EMPTY {
                orderDependency(cycle) // min # of dependency
                dependency = cycle.getFirstElement()
                propose(dependency) // using approaches
            }
        }
    }
}
Best refactoring

If the approach is not adopted the subtree is not considered

Root = starting point
p = profit
d = # of dependences
t = # of types
All possibilities = t^d

MC = Move class
MM = Move method
DIP = Dependency Inversion Principle
DI = Dependency Injection
Profit

 PROFIT is equal to the number of deleted dependencies

 We are evaluating other metrics

 ▶ Stable Dependencies Principle, SDP [1]
 ▶ Stable Abstraction Principle SAP [1]

[1] Robert C. Martin - Design Principles and Design Patterns
Best profit example

Root = starting point
MC = Move class
MM = Move method
DIP = Dependency Inversion Principle
DI = Dependency Injection

Best profit = MC + (DIP+DI)
= 3 + 4
= 7
Next steps

- Understanding Moose, Softwarenaut
- Experimenting hypothesis
- Finding new alternatives and evaluate
- Implementation of solution
Summary

- Problem caused by dependency cycles
- Different type of dependencies
- Several approaches
- Semiautomatic solution
- Refactoring with best profit