Stability Assessment of Aspect-Oriented Software Architectures
— New Findings and Challenges —

Alessandro Garcia
garciaa@comp.lancs.ac.uk

BCS Advanced Programming Group
8 January 2009
Stability: A Key Architecture Driver

• Stability is the ability of a software architecture to *sustain the modularity* of global design concerns and *not succumb to changes* [Parnas 94, Martin 97]
  – simpler definition: “a module is stable if it does not change”
  – instability indicators: modularity anomalies and *ripple effects*

• Empirical knowledge on stability as a key quality driver
  – reusable components are more stable and vice-versa [Gall 07]
  – stable component interfaces reduce defect-density [Conradi 04]

• Difficult to achieve stable architectures
  – increasing volatility of modern software requirements

Crosscutting Concerns

Hamper Architecture Stability

- The presence of the so-called *crosscutting concerns* (CCCs) is detrimental to software architecture stability
  - E.g. error handling, distribution, persistence, etc…


Architectural CCC: An Example

- On the use of a **Layered** software architecture

---

Architectural CCC: An Example

Crosscutting the interfaces of architecture layers

Legend:
- **Persistence Concern**
- **Distribution Concern**
- **Error Handling Concern**

- GUI Concern
- Distribution Concern
- Business Concern
- Persistence Concern
- Concurrency Concern

Mastering Architecture CCCs with Aspects

- Characteristics of Aspect-Oriented (AO) architectures
  - supported by AOP [Kiczales.97]
  - new modularity unit: architectural aspects
  - new “fine-grained” composition mechanisms:
    - join points and join point models
    - pointcuts (bindings + quantification mechanisms)
    - inter-type declarations: “enhancements” to type interfaces
    - etc…

Implementation-level Aspects

- *Exception handling* aspect

public class GenericOperations {
    public static boolean closeResultSet(ResultSet aResultSet) {
        ... // body of the original "try" block.
        return true; } ... // implementation of the class
    }

    public aspect GOHandler { // another source file

        pointcut crsHandler() : execution(public static boolean closeResultSet(..));

        boolean around() : crsHandler() { //advice
            try { return proceed();
            } catch (SQLException e) { ... return false; }
        }

    }

}
AOSD: Conventional Wisdom vs. Key Worries

• Recapping claims from early AOSD research stages
  – Architectural crosscutting concerns (CCC) must be “aspectised”
    • … thereby achieving architectures with superior stability
  – ‘Killer examples’: logging, tracing, error handling, distribution, persistence, etc…

• Key worries:
  – When upfront aspectisation translates into better software maintainability and evolvability?
    • superior stability of both CCCs and non-CCCs?
      – change effects in the presence of multiple aspects?
    • reuse: impact on well-known principles, such as the Open-Closed principle
Stability of AO Architectures is Unknown

• Only separation of concerns was considered!
  – influences of AOSD on fundamental principles?

• Empirical studies mostly focusing on single CCCs
  – does AOP scale in the presence of multiple evolving CCCs?

• No understanding of the *architectural* impact of aspects in evolving software systems
  – only aggressive incarnations of Aspect-Oriented (AO) ADLs
  – *multiple inheritance* were sources of major ripple effects

• Lack of *quantitative studies* to inform non-early adopters on the degree of AO architecture stability
Assessing AO Architecture Stability

- Comparison of AO vs. non-AO designs and implementations
  
  **focus:** AspectJ vs. Java as target Programming Languages

  - CASE 1: error handling aspects in 4 application architectures
  - CASE 2: aspectization of persistence, distribution, and exception handling in a N-Tier architecture

- **Goal:** observe evidences of modularity anomalies and ripple effects in the presence of changes
Case 1
First Empirical Study

- Exception handling with aspects
- Goal:
  - understand the modularity stability when extracting exception handling to aspects
    - optimal use of architectural decompositions and PL mechanisms
- Stability *estimation* based on variation in the measures: AO vs. non-AO architectures
  - measures for coupling, cohesion, size and separation of concerns
  - Negative values: AO solution likely to be more stable
Experimental Procedures

• Selection of 4 different software architectures
  – We partially or totally considered the implementation of:
    • Telestrada: traveller information system
      – 3350 LOC, > 200 modules
    • Pet Store: e-commerce demo for Java EE platform
      – 17500 LOC, > 330 modules
    • Eclipse CVS Core Plugin
      – 20000 LOC, > 170 modules
    • Health Watcher: web-based information systems for healthcare complaints
      – 6630 LOC, > 134 modules
Results – Concern Metrics

AO solutions seems to be more stable
difference is granted to the distinct EH aspectization strategies

<table>
<thead>
<tr>
<th>Application</th>
<th>Concern Diffusion over Components</th>
<th>Concern Diffusion over Operations</th>
<th>Concern Diffusion over LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Refactored</td>
<td>Original</td>
</tr>
<tr>
<td>Telestrada</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>22</td>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>Aspects</td>
<td>-</td>
<td>18</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
<td>18</td>
<td>42</td>
</tr>
<tr>
<td>Diff.</td>
<td>-18.18%</td>
<td>+4.76%</td>
<td>-100%</td>
</tr>
<tr>
<td>Java Pet Store</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>110</td>
<td>20</td>
<td>256</td>
</tr>
<tr>
<td>Aspects</td>
<td>-</td>
<td>37</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>110</td>
<td>57</td>
<td>256</td>
</tr>
<tr>
<td>Diff.</td>
<td>-48.18%</td>
<td>-21.88%</td>
<td>-92.81%</td>
</tr>
<tr>
<td>Eclipse CVS Core Plugin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>59</td>
<td>0</td>
<td>236</td>
</tr>
<tr>
<td>Aspects</td>
<td>-</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>59</td>
<td>4</td>
<td>236</td>
</tr>
<tr>
<td>Diff.</td>
<td>-93.22%</td>
<td>-23.73%</td>
<td>-100%</td>
</tr>
<tr>
<td>Health Watcher</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>35</td>
<td>0</td>
<td>115</td>
</tr>
<tr>
<td>EH Aspects</td>
<td>5</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Other Aspects</td>
<td>7</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>47</td>
<td>10</td>
<td>138</td>
</tr>
<tr>
<td>Diff.</td>
<td>-78.72%</td>
<td>-48.53%</td>
<td>-100%</td>
</tr>
</tbody>
</table>

certain instabilities observed
many operations with more than one try-catch block & no reuse

reuse was exceptionally high
Results – Size Metrics

Contradicting the general intuition that AOP makes programs smaller...

<table>
<thead>
<tr>
<th>Application</th>
<th>Lines of Code</th>
<th>Number of Attributes</th>
<th>Number of Operations</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telestrada</td>
<td>Classes</td>
<td>3352</td>
<td>127</td>
<td>423</td>
</tr>
<tr>
<td></td>
<td>Aspects</td>
<td>459</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>3352</td>
<td>127</td>
<td>423</td>
</tr>
<tr>
<td></td>
<td>Diff.</td>
<td>-0.54%</td>
<td>0%</td>
<td>+13.71%</td>
</tr>
<tr>
<td>Java Pet Store</td>
<td>Classes</td>
<td>17482</td>
<td>542</td>
<td>2075</td>
</tr>
<tr>
<td></td>
<td>Aspects</td>
<td>2045</td>
<td>6</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>17482</td>
<td>542</td>
<td>2075</td>
</tr>
<tr>
<td></td>
<td>Diff.</td>
<td>+0.89%</td>
<td>+1.11%</td>
<td>+11.57%</td>
</tr>
<tr>
<td>Eclipse CVS Core Plugin</td>
<td>Classes</td>
<td>18876</td>
<td>852</td>
<td>1832</td>
</tr>
<tr>
<td></td>
<td>Aspects</td>
<td>1620</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>18876</td>
<td>852</td>
<td>1832</td>
</tr>
<tr>
<td></td>
<td>Diff.</td>
<td>+2.82%</td>
<td>+0.23%</td>
<td>+9.66%</td>
</tr>
<tr>
<td>Health Watcher</td>
<td>Classes</td>
<td>5732</td>
<td>152</td>
<td>542</td>
</tr>
<tr>
<td></td>
<td>EH Aspects</td>
<td>86</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Other Aspects</td>
<td>812</td>
<td>12</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>6630</td>
<td>167</td>
<td>655</td>
</tr>
<tr>
<td></td>
<td>Diff.</td>
<td>-6.56%</td>
<td>+2.4%</td>
<td>+11.45%</td>
</tr>
</tbody>
</table>

some reuse was compensated by the overhead of using AspectJ

some reuse of handlers
Lessons Learned: Estimating Architecture Stability

- **Size measures** – *evidences of potential instabilities later*
  - increased number of operations (join point exposition)
    - new operations: 3.3% Telestrada, 2.9% in the Java Pet Store, 0.79% in the CVS Plugin, 1.7% in the Health Watcher
    - negative result: many cases did not state the developer intent

- **Coupling**
  - no major influence of aspectization
  - however, a closer examination in the code...
    - subtle kind of coupling
      - use of exception softening creates an implicit, compile-time dependency of the base code on the EH aspect
AO architectures seem to be more stable, but…

• **Which** elements of an exception handling strategy makes AO architectures likely to be more stable?
  – when AO mechanisms are beneficial or harmful to design stability?

• We have determined the top-5 factors that contribute to positive/negative stability of exception handling
  – E.g. based on the modularity measures used in the study
## Modular Aspectization of EH

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Tangled try-catch blocks</th>
<th>Nested try-catch blocks</th>
<th>Placement of exception-throwing code</th>
<th>Handler depends on local variables</th>
<th>Flow of control after handler execution</th>
<th>Score</th>
<th>Should extract?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untangled Handler</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>non-t.</td>
<td>term.</td>
<td>read</td>
<td>write</td>
</tr>
<tr>
<td>Tangled, Non-Mask. Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested, Non-Mask. Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tangled Handler, Term. ETC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested Handler, Term. ETC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested Block Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context-Dependent Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested, Context-Dependent Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context-Affecting Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop Iter. Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \approx 70\% \]

\[ \approx 5.15\% \]
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Tangled try-catch blocks</th>
<th>Nested try-catch blocks</th>
<th>Placement of exception-throwing code</th>
<th>Handler depends on local variables</th>
<th>Flow of control after handler execution</th>
<th>Score</th>
<th>Should extract?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untangled Handler</td>
<td>X</td>
<td>X</td>
<td>yes</td>
<td>X</td>
<td>X</td>
<td>0-1</td>
<td>Yes.</td>
</tr>
<tr>
<td>Tangled, Non-Mask. Handler</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>0</td>
<td>Yes.</td>
</tr>
<tr>
<td>Nested, Non-Mask. Handler</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>1</td>
<td>Yes.</td>
</tr>
<tr>
<td>Tangled Handler, Term. ETC</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>0</td>
<td>Yes.</td>
</tr>
<tr>
<td>Nested Handler, Term. ETC</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>1</td>
<td>Yes.</td>
</tr>
<tr>
<td>Block Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>2</td>
<td>Depends</td>
</tr>
<tr>
<td>Nested Block Handler</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>3</td>
<td>Depends</td>
</tr>
<tr>
<td>Context-Dependent Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>2-4</td>
<td>Depends</td>
</tr>
<tr>
<td>Nested, Context-Dependent Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>3-5</td>
<td>No.</td>
</tr>
<tr>
<td>Context-Affecting Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>3-6</td>
<td>No.</td>
</tr>
<tr>
<td>Loop Iter. Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>5-9</td>
<td>No.</td>
</tr>
</tbody>
</table>
Case 2

AO Software Architectures
Taking History of Architecture Changes into Consideration...

- **2nd Exploratory Empirical Study**
  - evolution of the HW architecture

- **Aim:** assessing various facets of AO vs. OO architecture stability
  - focus on typical software maintenance tasks
  - analysed 9 change scenarios (i.e. 10 releases)

- **Multi-dimensional analysis**
  - modularity sustenance
  - ripple effects
  - satisfaction of basic design principles
Non-AO Architecture

**View Layer**
- Command Abstract Class
- ExtraOpCommands Class
- OpCommands Class
- OpServlets Class

**Distribution Layer**
- IFacade Interface

**Business Layer**
- Symptom Class
- Employee Class
- HealthUnit Class
- Speciality Class
- Complaint Class
- HealthWatcherFacade Class

**Data Layer**
- HealthUnitRecord Class
- SpecialityRecord Class
- ComplaintRecord Class
- EmployeeRecord Class
- SymptomRecord Class

Additional Notes:
- +Sx Component added in Scenario x
- ~Sx Component altered in Scenario x
- -Sx Component removed in Scenario x
Non-AO Architecture

- Sx Component added in Scenario x
- ~Sx Component altered in Scenario x
- -Sx Component removed in Scenario x

~S3: use Observer to monitor object updates
Non-AO Architecture

View Layer
- Command: Abstract Class
- ExtraOpCommands: Class
- OpCommands: Class
- HWServlet: Abstract Class

Distribution Layer
- Ifacade Interface

Business Layer
- Symptom: Class
- Employee: Class
- HealthUnit: Class
- Speciality: Class
- HealthWatcherFacade: Class

Data Layer
- HealthUnitRep Interface
- SpecialityRep Interface
- ComplaintRep Interface
- EmployeeRep Interface
- SymptomRep Interface

Changes:
- +Sx Component added in Scenario x
- ~Sx Component altered in Scenario x
- -Sx Component removed in Scenario x

Notes:
- ~S8: inclusion of use cases to support new queries to data types
- ~S9: more robust error handling strategies
Coupling and Cohesion

- More stable coupling and cohesion in AO
- Version 4 introduces the Observer pattern
  - Pointcuts and declare parents reduce coupling
- CCCs aspectised upfront were stable
  - Exception: again EH
    - refactoring methods to expose context

![Graphs showing changes in coupling and cohesion over versions](Image)
Ripple Effects

• Localization of changes to CCCs
  – These have to span multiple layers/concerns in the OO architecture
Ripple Effects

• Localization of changes to CCCs
  – These have to span multiple layers/concerns in the OO implementation

• Localization of changes to non-CCCs
  – OO performs better or comparable to AO
  – interesting as the AO versions have the same core layers

• Removal of elements in the base seems to cause more instabilities in AO architecture implementations
  – Observed in several studies: product lines (ICSE), framework composition (ICCBSS)
  – controlled experiment: were major causes of severe faults [Pascal Durr, PhD thesis, Univ. Twente]

• However modularity properties were not affected in the AO version!
Learning from the 2 Cases and Beyond…

• With AO software architectures:
  – modularity anomalies of otherwise CCCs are in general minimized
  – amount of changes or observation of ripple effects are reduced

• However… New Challenges
  – AO mechanisms do not scale with CCC overlaps
    • suggests a hybrid AOP language: AspectJ + HyperJ?
  – Limitations of join point models for specific crosscutting concerns, such as exception handling
  – Empirical validation of the concern metrics
  – Execution of rigorous (semi-)controlled experiments, case studies, etc…
Challenges...

- Aspectisation of architectural exception handling was a consistent problem in terms of error proneness [Coelho et al, ECOOP 2008]
  - "Softening" checked exceptions is not a safe mechanism
    - Exception and Throwable should never be softened
    - AspectJ renders Java’s static checks useless
  - More faults in exception being caught by subsumption in AO programs
  - EJFlow – a design model for explicit definition of end-to-end architectural exception flows [Cacho et al, AOSD 2008]
    - Extended AspectJ (using abc)

- The next challenge for designers of AO programming languages
  - how to promote more reliable AOP?
- AOP fault models for specific CCCs
Questions?

Alessandro Garcia
garciaa@comp.lancs.ac.uk

moving soon (Jan 21) to PUC @ Rio, Brazil
Email to: afgarcia@inf.puc-rio.br

BCS Advanced Programming Group