LANCASTER UNIVERSITY

Computing Department

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

InfoLab21

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

D. Parnas. ***Software Aging***. Proc. ICSE, 1992.

R. Martin. ***Stability.*** The C++ *Report*, 2008.

Crosscutting Concerns *Hamper* Architecture Stability

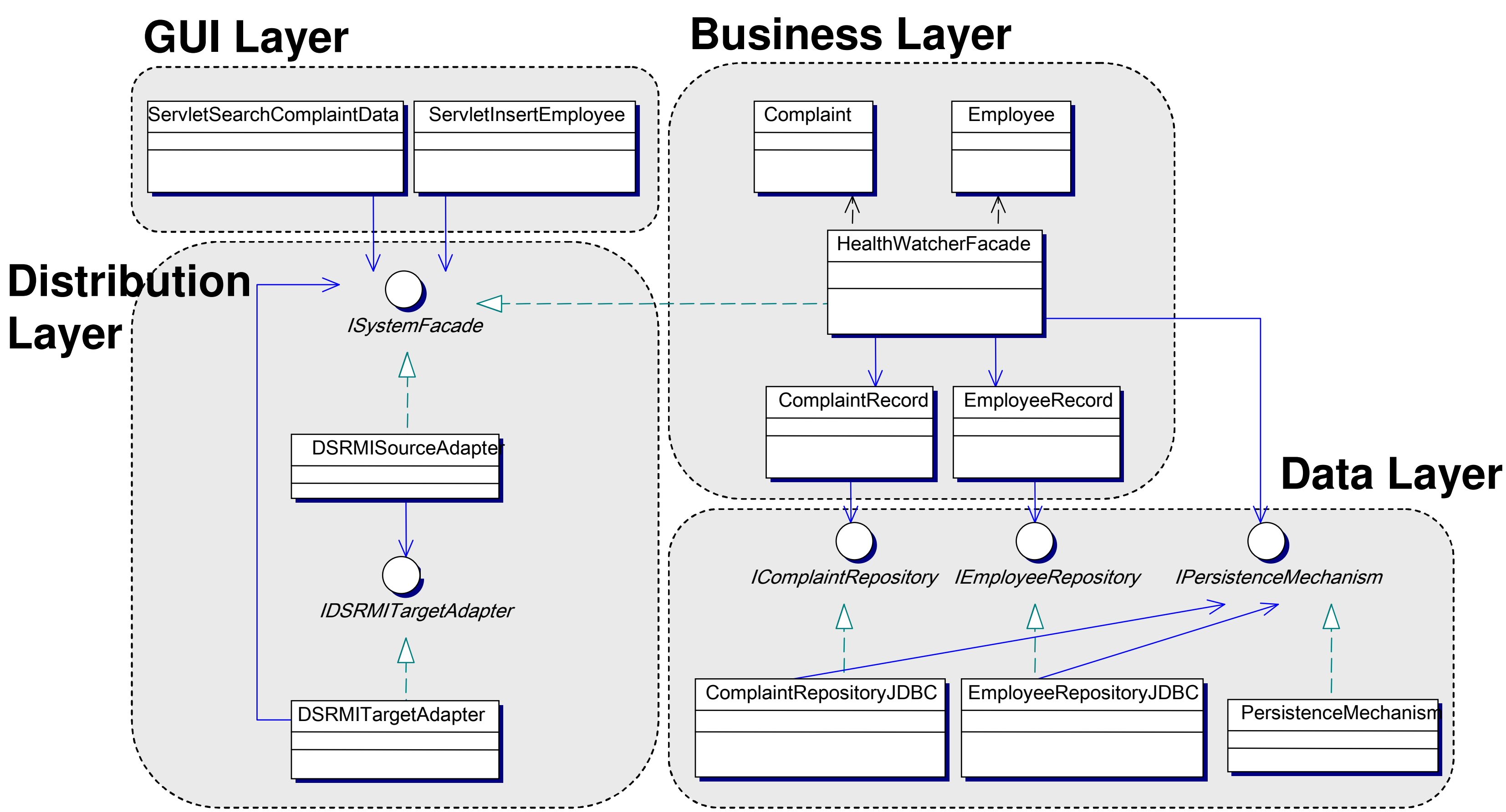
- The presence of the so-called *crosscutting concerns* (CCCs) is detrimental to software architecture stability
  - E.g. error handling, distribution, persistence, etc…

Garcia, A et al. ***On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study***. Proc. 21st ECOOP, July 2007, Berlin.

Murphy, G et al. ***Do Crosscutting Concerns Cause Defects?*** IEEE Transactions on Software Engineering, to appear, 2008.

Architectural CCC: An Example

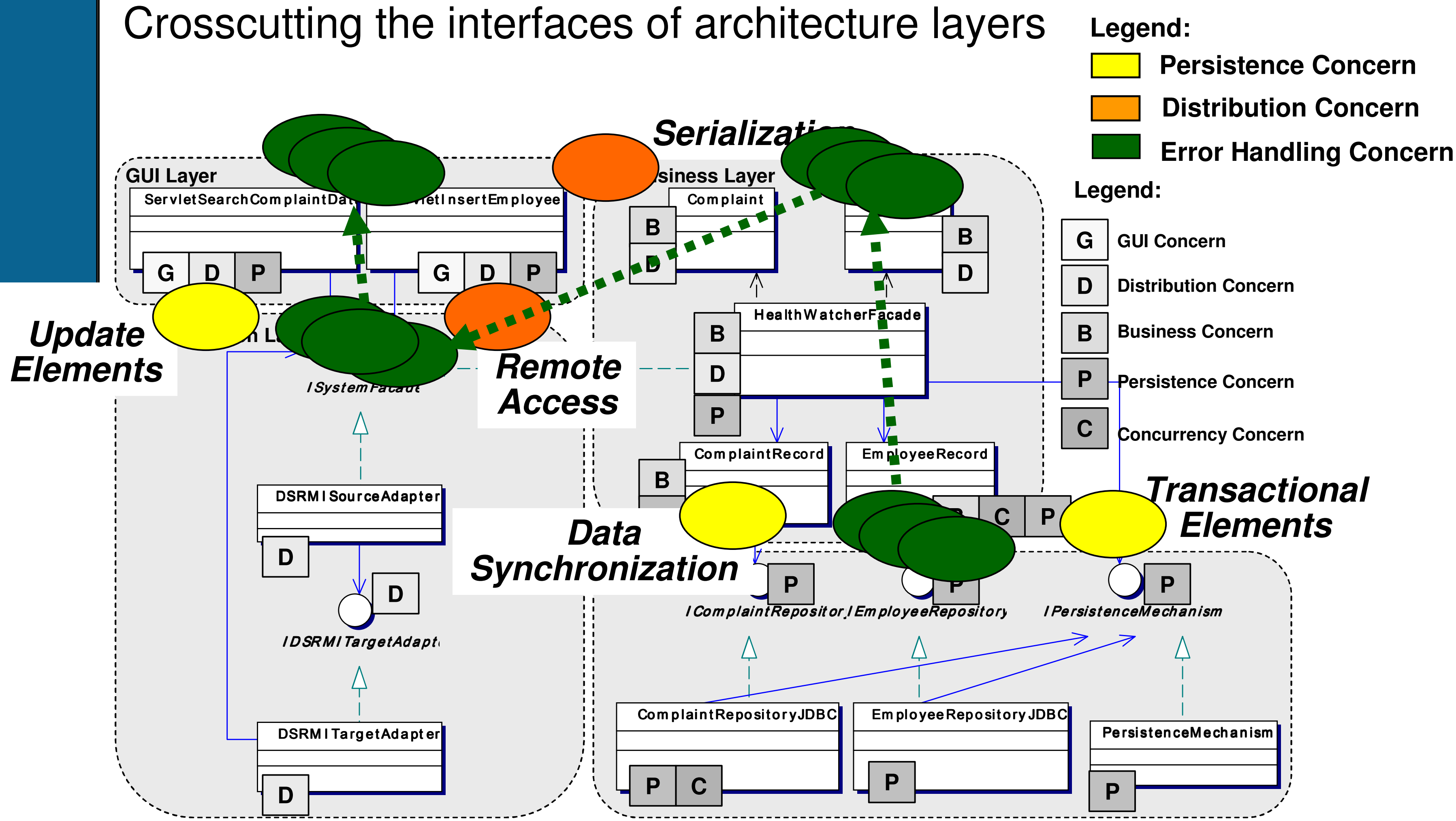
- On the use of a Layered software architecture



S. Soares, et al. **"Implementing Distribution and Persistence Aspects with AspectJ"**. Proceedings of the OOPSLA'02, pp. 174-190.

Architectural CCC: An Example

Crosscutting the interfaces of architecture layers



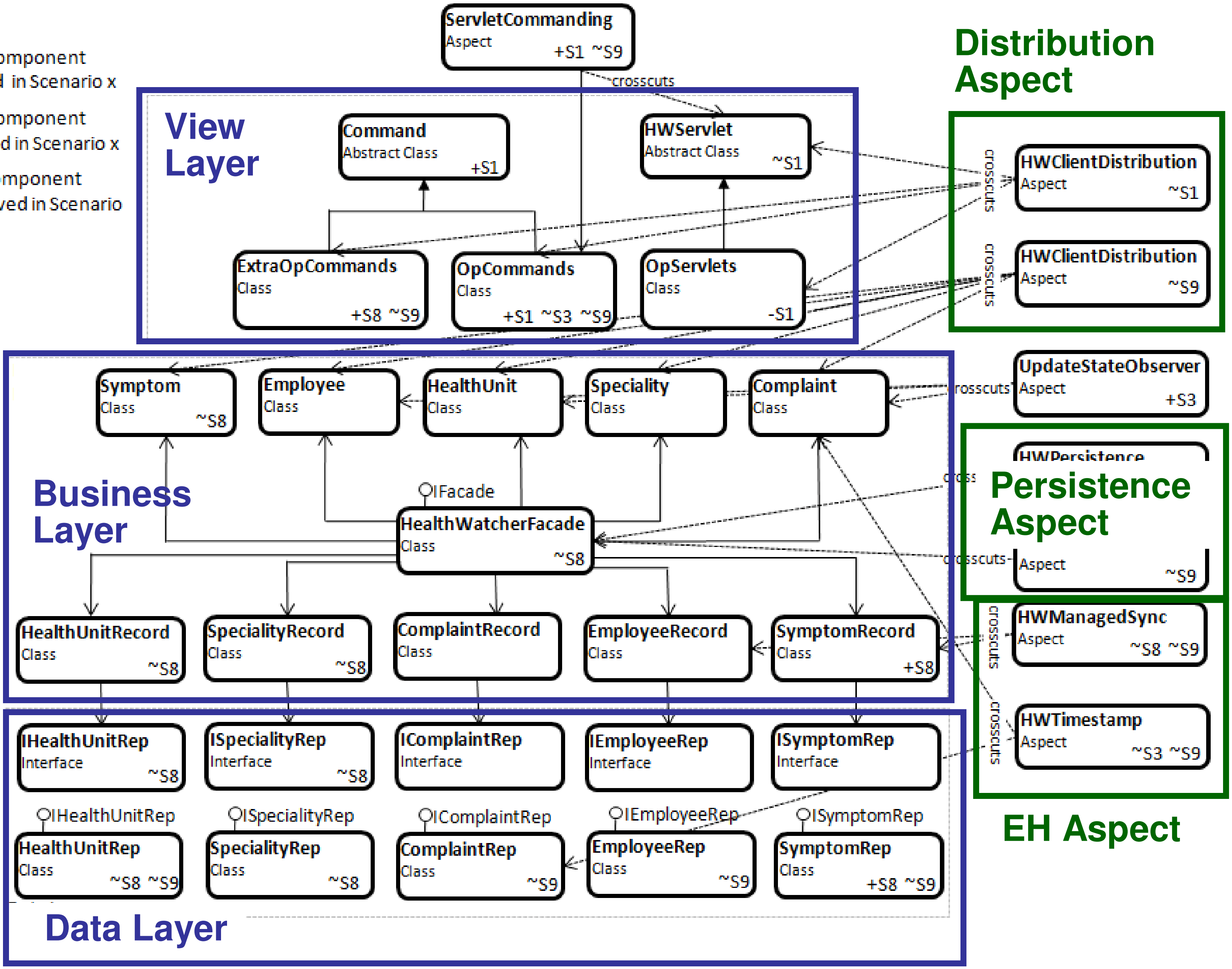
Kulesza, Sant'Anna, Garcia, Coelho, Lucena. **"Quantifying the Effects of AOP: A Maintenance Study."** Proceedings of Intl. Conf. on Software Maintenance (ICSM'06), September 2006.

AO Architecture

+Sx Component added in Scenario x

~Sx Component altered in Scenario x

-Sx Component removed in Scenario x



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…

G. Kiczales et al. ***Aspect-Oriented Programming***. Proc. ECOOP 1997, pp. 220-242

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

AO Software Architectures

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:

JAVA

- 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

ASPECTJ

- 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

Application		*Concern Diffusion over Components*		*Concern Diffusion over Operations*		*Concern Diffusion over LOC*	
		Original	Refactored	Original	Refactored	Original	Refactored
Telestrada	*Classes*	22	0	42	0	208	0
	Aspects	-	18	-	44	-	0
	Total	**22**	**18**	**42**	**44**	**208**	**0**
	Diff.	-18.18%		+4.76%		-100%	
Java Pet Store	*Classes*	110	20	256	21	1168	84
	Aspects	-	37	-	179	-	0
	Total	**110**	**57**	**256**	**200**	**1168**	**84**
	Diff.	-48.18%		-21.88%		-92.81%	
Eclipse CVS Core Plugin	*Classes*	59	0	236	0	1118	0
	Aspects	-	4	-	180	-	0
	Total	**59**	**4**	**236**	**180**	**1118**	**0**
	Diff.	-93.22%		-23.73%		-100%	
Health Watcher	*Classes*	35	0	115	0	488	0
	EH Aspects	5	10	9	70	0	0
	Other Aspects	7	0	12	0	48	0
	Total	**47**	**10**	**136**	**70**	**536**	**0**
	Diff.	-78.72%		-48.53%		-100%	

reuse was exceptionally high

<u>certain instabilities observed</u>

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

Results – Size Metrics

Contradicting the general intuition that AOP makes programs smaller…

some reuse was compensated by the overhead of using AspectJ

Application		*Lines of Code*		*Number of Attributes*		*Number of Operations*		*Vocabulary Size*	
		Original	Refac.	Original	Refac.	Original	Refac.	Original	Refac.
Telestrada	*Classes*	3352	2885	127	127	423	437	224	224
	Aspects	-	459	-	0	-	44	-	18
	Total	**3352**	**3334**	**127**	**127**	**423**	**481**	**224**	**242**
	Diff.	**-0.54%**		**0%**		**+13.71%**		**+8.04%**	
Java Pet Store	*Classes*	17482	15593	542	542	2075	2135	339	339
	Aspects	-	2045	-	6	-	180	-	37
	Total	**17482**	**17638**	**542**	**548**	**2075**	**2315**	**339**	**376**
	Diff.	**+0.89%**		**+1.11%**		**+11.57%**		**+10.91%**	
Eclipse CVS Core Plugin	*Classes*	18876	17803	852	854	1832	1848	257	257
	Aspects	-	1620	-	0	-	180	-	4
	Total	**18876**	**19423**	**852**	**854**	**1832**	**2028**	**257**	**261**
	Diff.	**+2.82%%**		**+0.23%%**		**+9.66%**		**+1.43%**	
Health Watcher	*Classes*	5732	4641	152	152	542	553	98	98
	EH Aspects	86	853	3	7	9	73	5	10
	Other Aspects	812	701	12	12	104	104	31	31
	Total	**6630**	**6195**	**167**	**171**	**655**	**730**	**134**	**139**
	Diff.	**-6.56%**		**+2.4%**		**+11.45%**		**+3.73%**	

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

Scenario	Tangled try-catch blocks		Nested try-catch blocks		Placement of exception-throwing code		Handler depends on local variables			Flow of control after handler execution				Score	Should extract?
	yes	no	yes	no	non-t.	term.	read	write	no	mask.	prop.	ret.	loop		
Untangled Handler		X	X	X	X	X			X	X	X	X		0-1	Yes. ✓
Tangled, Non-Mask. Handler	X			X	X	X			X		X	X		0	Yes. ✓
Nested, Non-Mask. Handler	X		X		X	X			X		X	X		1	Yes. ✓
Tangled Handler, Term. ETC	X			X		X			X	X				0	Yes. ✓
Nested Handler, Term. ETC	X		X			X			X	X				1	Yes. ✓
Block Handler	X			X	X				X	X				2	Depends. ?
Nested Block Handler	X		X		X				X	X				3	Depends. ?
Context-Dependent Handler	X	X		X	X	X	X			X	X	X		2-4	Depends. ?
Nested, Context-Dependent Handler	X	X	X		X	X	X			X	X	X		3-5	No. ✗
Context-Affecting Handler	X					X		X		X	X	X		3-6	No. ✗
Loop Iter. Handler		X	X	X	X	X	X	X	X				X	5-9	No. ✗

≈70%

≈ 5..15%

When to Aspectize EH?

Scenario	Tangled try-catch blocks		Nested try-catch blocks		Placement of exception-throwing code		Handler depends on local variables			Flow of control after handler execution				Score	Should extract?
	yes	no	yes	no	non-t.	term.	read	write	no	mask.	prop.	ret.	loop		
Untangled Handler		X	X	X	X	X			X	X	X	X		0-1	Yes.
Tangled, Non-Mask. Handler	X			X	X	X			X		X	X		0	Yes.
Nested, Non-Mask. Handler	X		X		X	X			X		X	X		1	Yes.
Tangled Handler, Term. ETC	X			X		X			X	X				0	Yes.
Nested Handler, Term. ETC	X		X			X			X	X				1	Yes.
Block Handler	X			X	X				X	X				2	Depends.?
Nested Block Handler	X		X		X				X	X				3	Depends.?
Context-Dependent Handler	X	X		X	X	X	X			X	X	X		2-4	Depends.?
Nested, Context-Dependent Handler	X	X	X		X	X	X			X	X	X		3-5	No.
Context-Affecting Handler	X	X	X	X	X	X		X		X	X	X		3-6	No.
Loop Iter. Handler		X	X	X	X	X	X	X	X				X	5-9	No.

AO Software Architectures

Case 2

Taking History of Architecture Changes into Consideration…

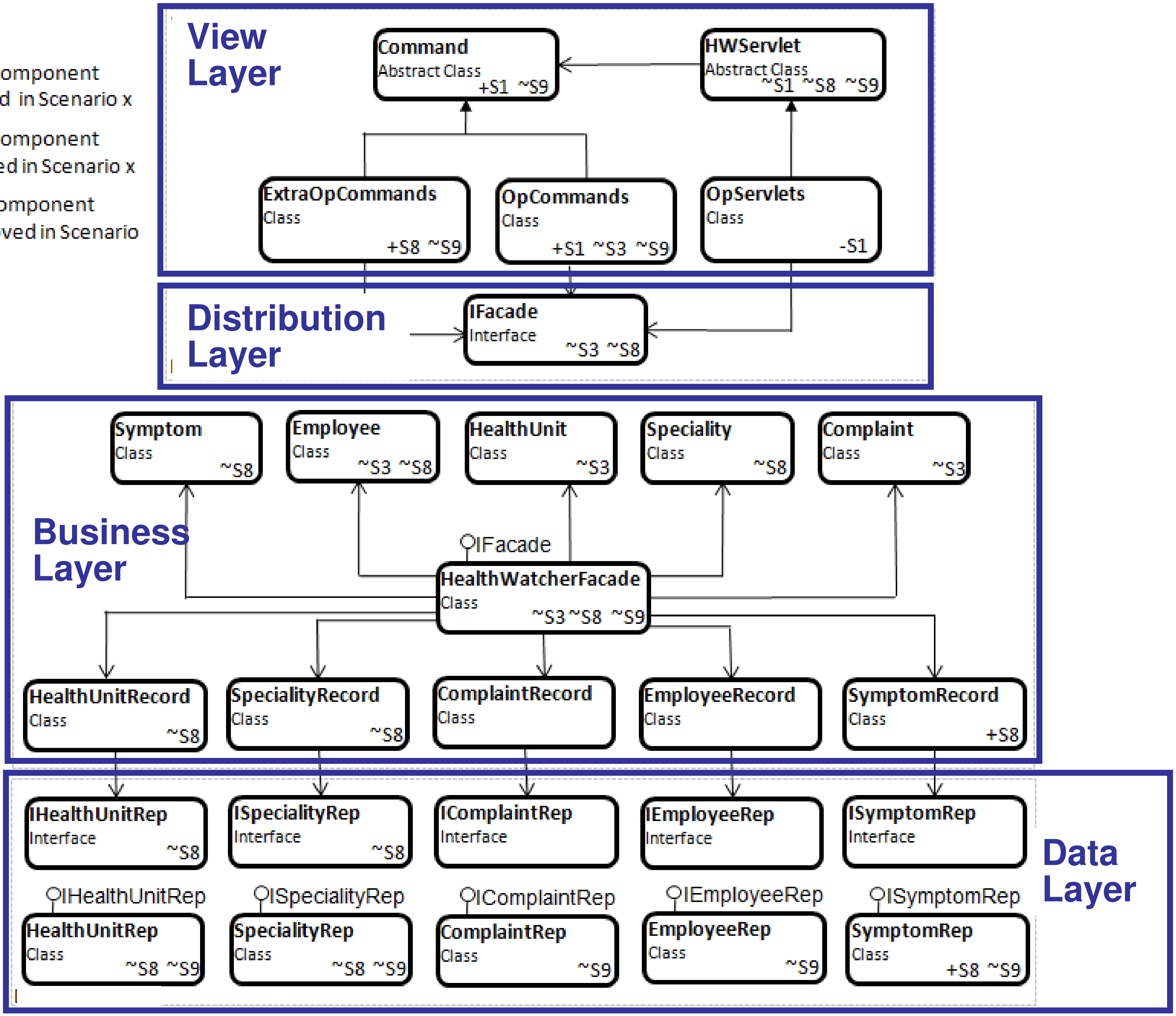
- 2nd Exploratory Empirical Study
  - evolution of the HW architecture
- Aim: assessing various facets of AO vs. OO architecture stability
  - focus on typical software maintenance tasks
  - analysed 9 change scenarios (i.e. 10 releases)
- Multi-dimensional analysis
  - modularity sustenance
  - ripple effects
  - satisfaction of basic design principles

Non-AO Architecture

+Sx Component added in Scenario x

~Sx Component altered in Scenario x

-Sx Component removed in Scenario x

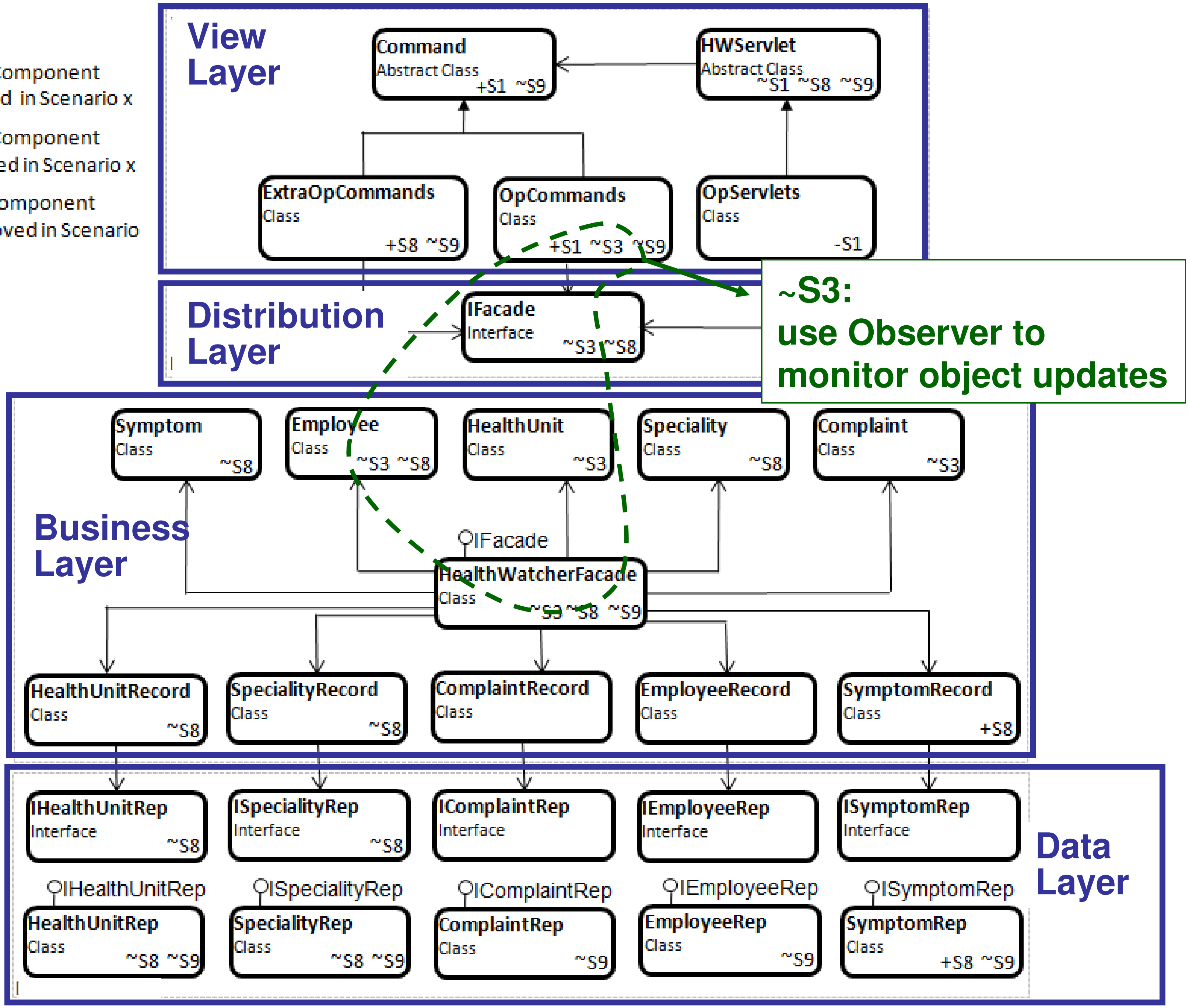


Non-AO Architecture

+Sx Component added in Scenario x

~Sx Component altered in Scenario x

-Sx Component removed in Scenario x

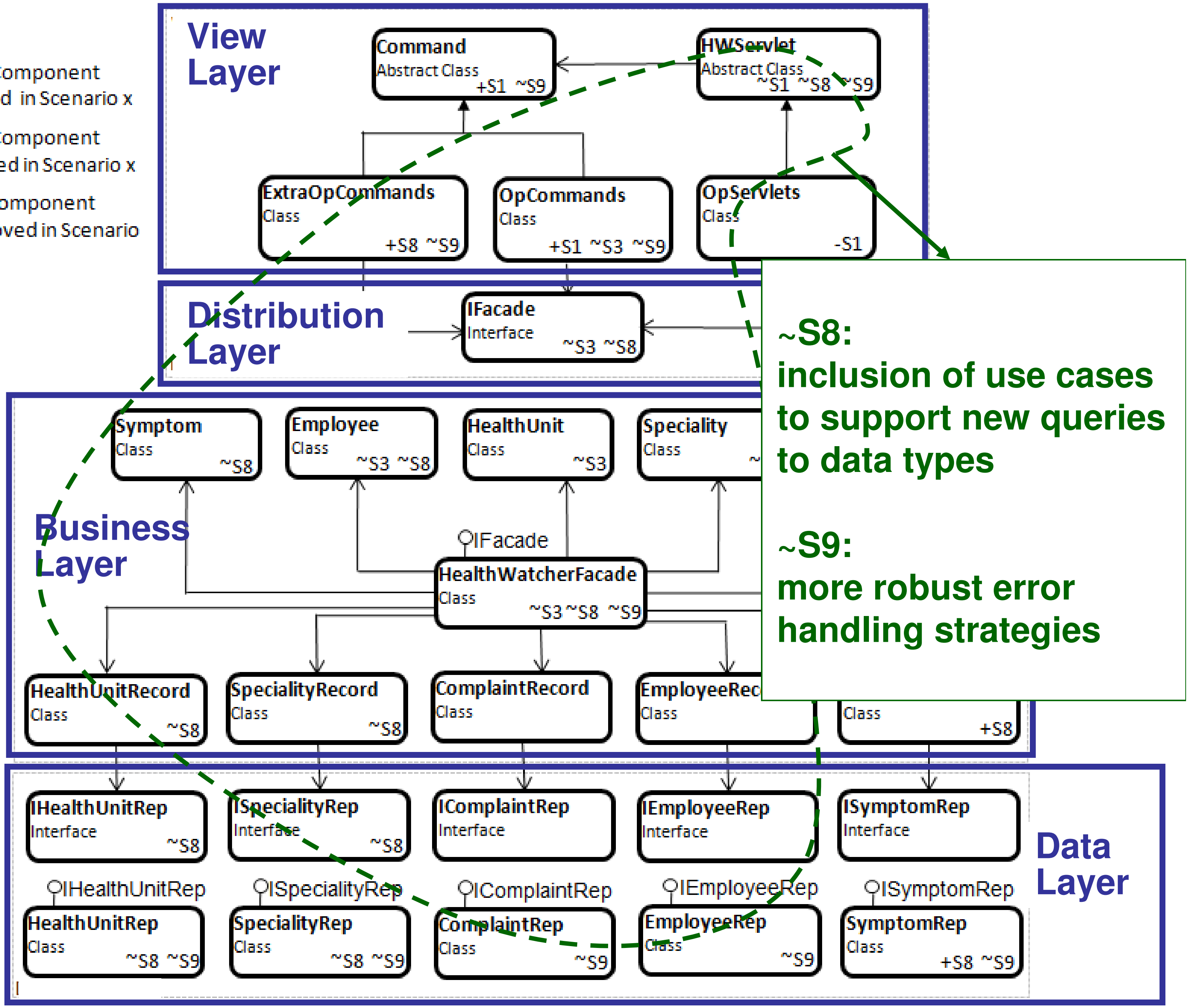


Non-AO Architecture

+Sx Component added in Scenario x

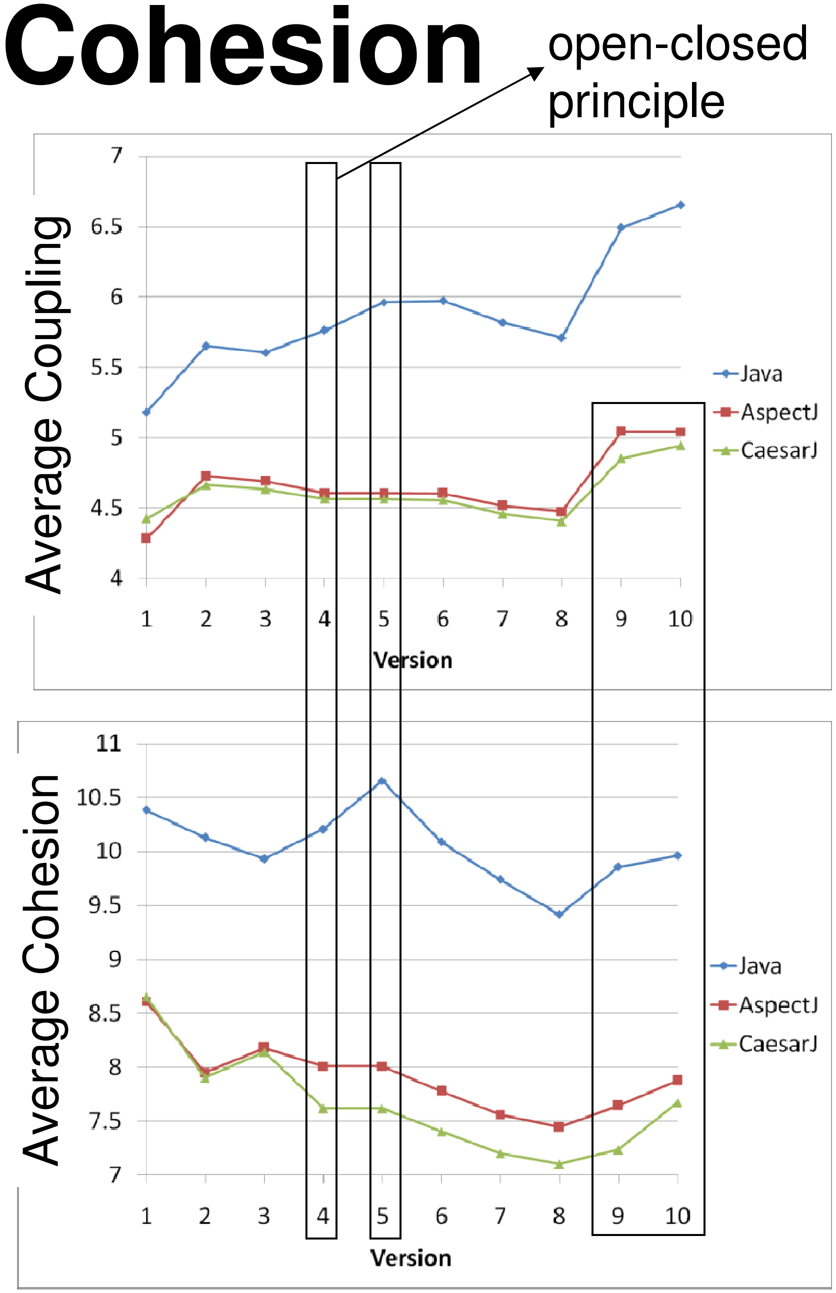
~Sx Component altered in Scenario x

-Sx Component removed in Scenario x



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



Ripple Effects

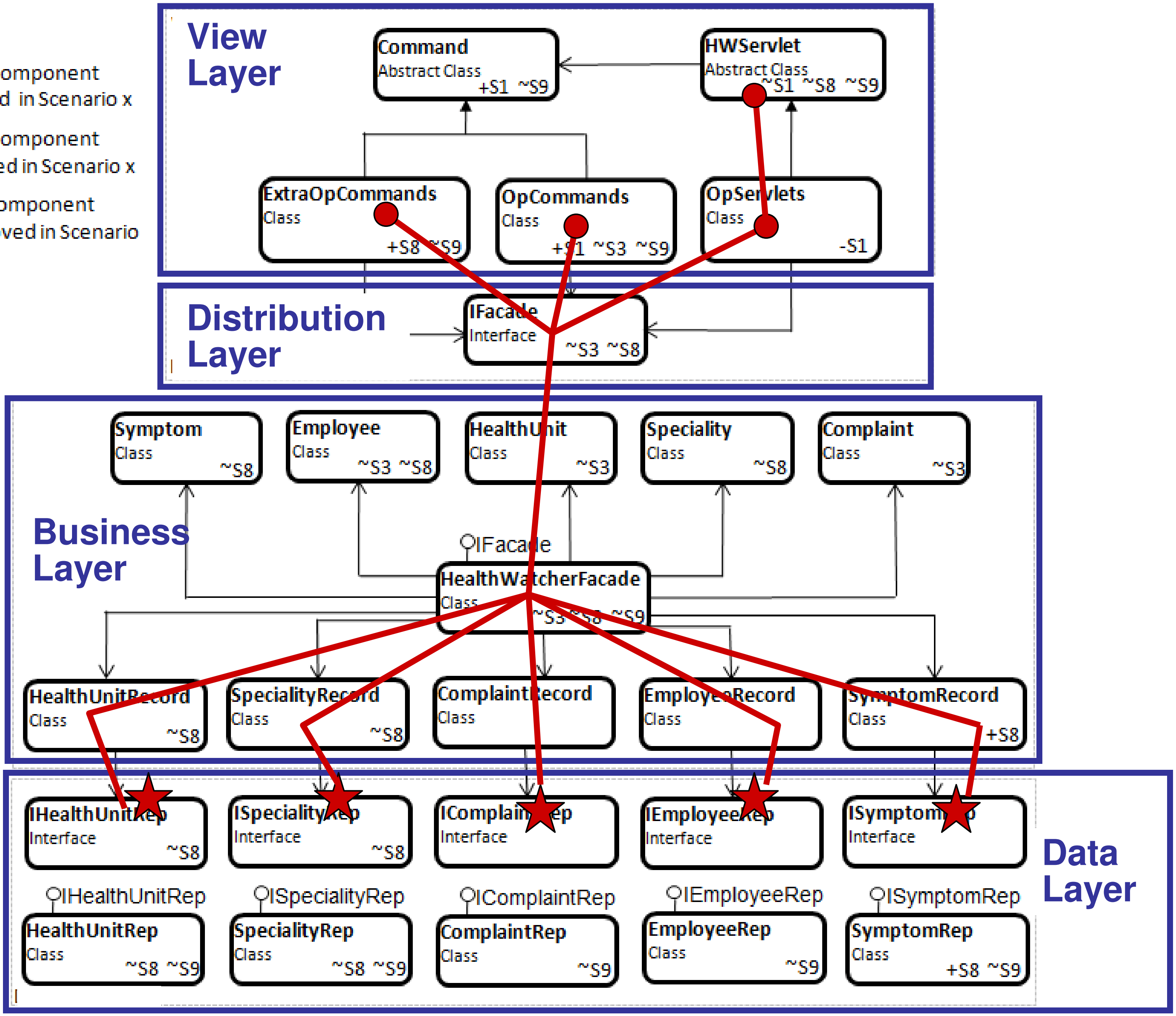
- Localization of changes to CCCs
  - These have to span multiple layers/concerns in the OO architecture

Architecture Design Stability

+Sx Component added in Scenario x

~Sx Component altered in Scenario x

-Sx Component removed in Scenario x



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!

AO Software Architectures

Research Challenges

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

LANCASTER UNIVERSITY
Computing Department

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

InfoLab21