

# Semantic Dependencies and Modularity of Aspect-Oriented Software

Alberto Costa Neto  
Márcio de Medeiros Ribeiro  
Marcos Dósea  
Rodrigo Bonifácio  
Paulo Borba  
**Federal University of Pernambuco**  
{acn, mmr3, mbd2, rba2, phmb}  
@cin.ufpe.br

Sérgio Soares  
**Pernambuco State University**  
sergio@dsc.upe.br



SOFTWARE • PRODUCTIVITY • GROUP



---

# Increasing complexity in Software Design

- Quality in Software Design
- Modularity as a desirable attribute
- Analyze designs and identify the best

---

# Aspect-Oriented Programming

- Crosscutting Concerns (CCC) Modularization
- No consensus about how to evaluate an AO design

---

# Techniques to evaluate AO Systems

- Evaluate: Coupling, Cohesion, Size, Complexity and Separation of Concerns
- Based on:
  - Metrics
  - Dependence Graphs
  - Design Structure Matrixes (DSM) + Net Option Values (NOV)

# Example of a DSM

	Display	Mouse	Keyboard
Display			
Mouse	X		X
Keyboard		X	

**Mouse and Keyboard cannot be independently developed**

---

# Overview

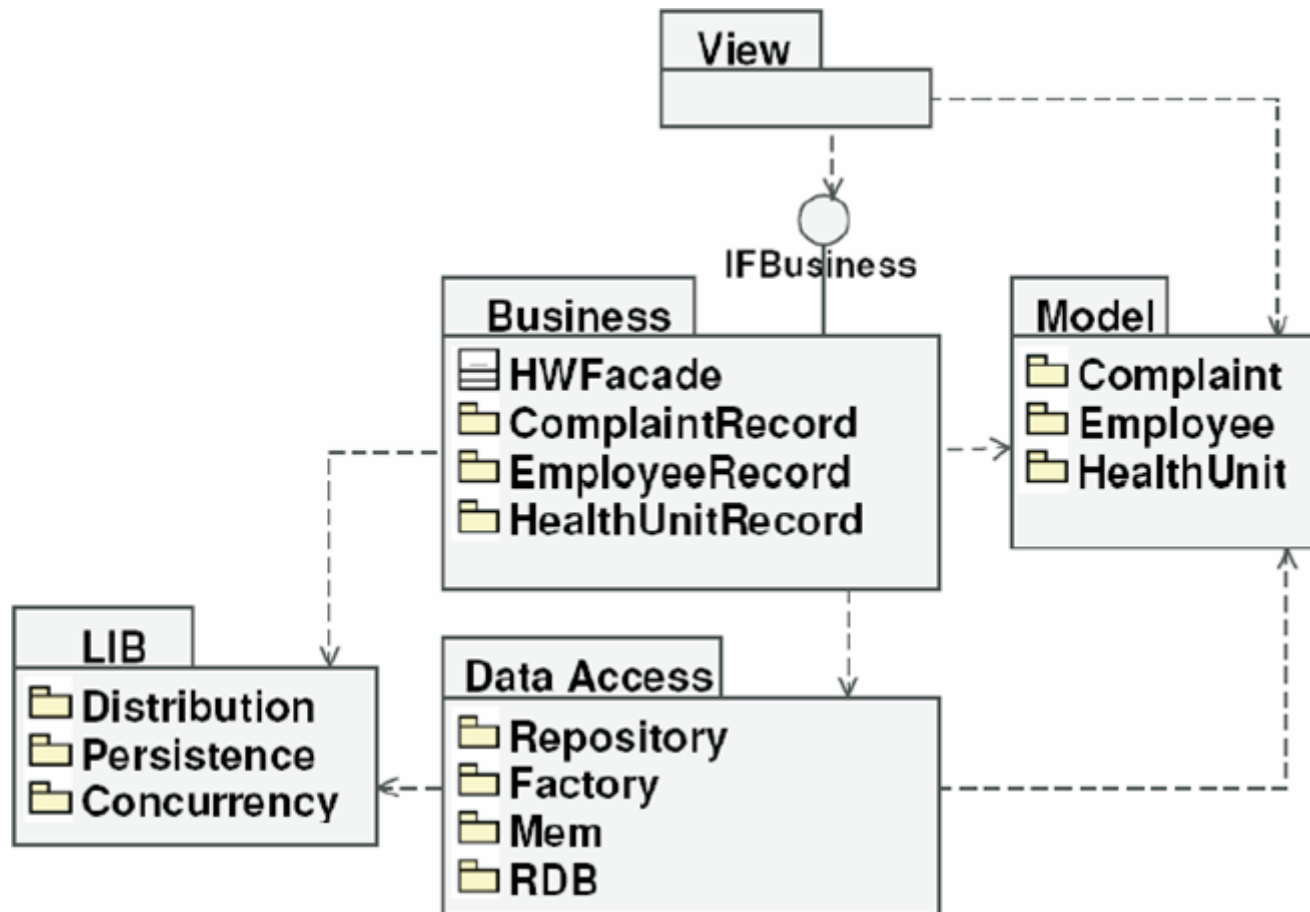
- We used DSMs to compare the OO and AO versions of a system
  - Some Syntactic Dependencies disappeared (AO)
  - Components were not really independent
- Identified Semantic Dependencies
- Expressed these dependencies as Design Rules and reanalyzed

---

# Health Watcher System

- Real Web-based information system
  - Originally developed in Java
  - Restructured to use AspectJ
- Selected because of significant number of CC and non-CC concerns.
- Requires several day-to-day design decisions (GUI, persistence, concurrency)

# Health Watcher Architecture



---

# Design Parameters Chosen

- **Software Components + Design Rules**
  - Compare design structures of different HW implementation
- **Initial Notion of Dependency**
  - Explicit references between components

---

# Design Rules

- Parameters used as interfaces between modules
  - Parameters that are less likely to be changed
- Promotes decoupling of design parameters

# OO HW

		Design Rules					Lib			Health Watcher					
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Conceptual Model	-													
2	IFDistribution		-												
3	IFPersistence			-											
4	IFView	4			-		1								
5	IFData	7				-									
6	IFPatterns						-								
7	Distribution							-							
8	Persistence								-						
9	Concurrency									-					
10	Model						2	4			-				
11	View	35			21						3	-			
12	Business	10		1	1	5	1		1	1			-		2
13	Data	19		7					12					-	
14	Patterns	12		1	1	16		8					2	12	-

---

# OO HW Analysis

- Design Rule **Conceptual Model** avoids many dependencies
- **Model** and **Data** can be developed in parallel
- After that, **View**, **Business** and **Pattern** development can start

---

# OO HW Analysis (continuation)

- Architecture prevents some but not all code tangling and scattering
  - HWFacade (Concurrency Management and Distribution)
  - Exception Handling

---

# Concurrency Management (OO)

```
public void insert(Employee employee) throws
    ObjectNotValidException , ObjectAlreadyInsertedException ,
    ObjectNotValidException , RepositoryException {
    manager.beginExecution(employee.getLogin());
    if (employeeRepository.exists(employee.getLogin())) {
        throw new ObjectAlreadyInsertedException(
            ExceptionMessages.EXC_JA-EXISTE);
    } else {
        employeeRepository.insert(employee);
    }
    manager.endExecution(employee.getLogin());
}
```

**Tangled code in EmployeeRecord**

---

# Concurrency Management (AO)

```
public aspect HWManagedSynchronization {  
  private ConcurrencyManager  
    manager = new ConcurrencyManager ();  
  
  public pointcut synchronization (Employee emp) :  
    execution(* EmployeeRecord.insert (Employee))  
    && args (emp);  
  
  before (Employee emp) : synchronization (emp) {  
    manager.beginExecution (emp.getLogin ());  
  }  
  
  after (Employee emp) : synchronizationPoints (emp) {  
    manager.endExecution (emp.getLogin ());  
  }  
}
```

---

# AO HW

Design Rules					Lib			Health Watcher					Aspects			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

1 Conceptual Model	-															
2 IFPersistence		-														
3 IFView	4		-													
4 IFData	7			-												
5 IFPatterns					-											
6 Distribution						-										
7 Persistence							-									
8 Concurrency								-								
9 Model									-							
10 View	35		17						3	-	1					
11 Business	10		1	5							-					
12 Data	19	6					12					-				
13 Patterns	22	1		15								10	-			
14 Aspects - Concurrency	2						3	1		1	1	3	1	-		1
15 Aspects - Distribution	4		1				7			1	1				-	
16 Aspects - Persistence		4									7	1				-
17 Aspects - Patterns	32		2							29	2	9				-

---

# AO HW Analysis

- Dependencies between health watcher and lib components were transferred to aspects
- health watcher components do not depend on aspects
- Better Separation of Concerns
- Possibility to enhance the support to Parallel Development

---

# Syntactic Coupling

- Direct reference between components
  - Inheritance, composition, method signature, pointcut declarations...
  - When a referred component changes, compilation errors may appear
- But we found dependencies that are not visible in the source code
- It is not enough to a complete study of coupling in AO systems

---

## Use of wildcards (quantification)

- Transaction Management (begin/commit...)

```
pointcut transactionalMethods():  
execution(* HealthWatcherFacade.*(..)) &&  
!execution(static *.*(..));
```

- Suppose that a developer creates a method in HealthWatcherFacade class
  - If it is **not static**, it **will have transaction management** (probably unnecessary)

---

# More Examples

- The class depends on aspect to work
  - ❑ A method requires transaction management
  - ❑ Class changes and pointcut does not match anymore
  
- cflow(below)
  - ❑ Impossible to know the join points in advance
  - ❑ Parallel development is impossible (requires knowledge about the control flow)

---

# Semantic Coupling

- Not syntactically defined in the code
  - No explicit reference between system components
- No compilation errors when removing or modifying components

---

# Semantic Coupling (continuation)

- We did not find such dependencies in OO without a corresponding syntactic representation
- Unsuitable when considering parallel development

# DSM with Semantic Coupling

Design Rules					Lib			Health Watcher					Aspects			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

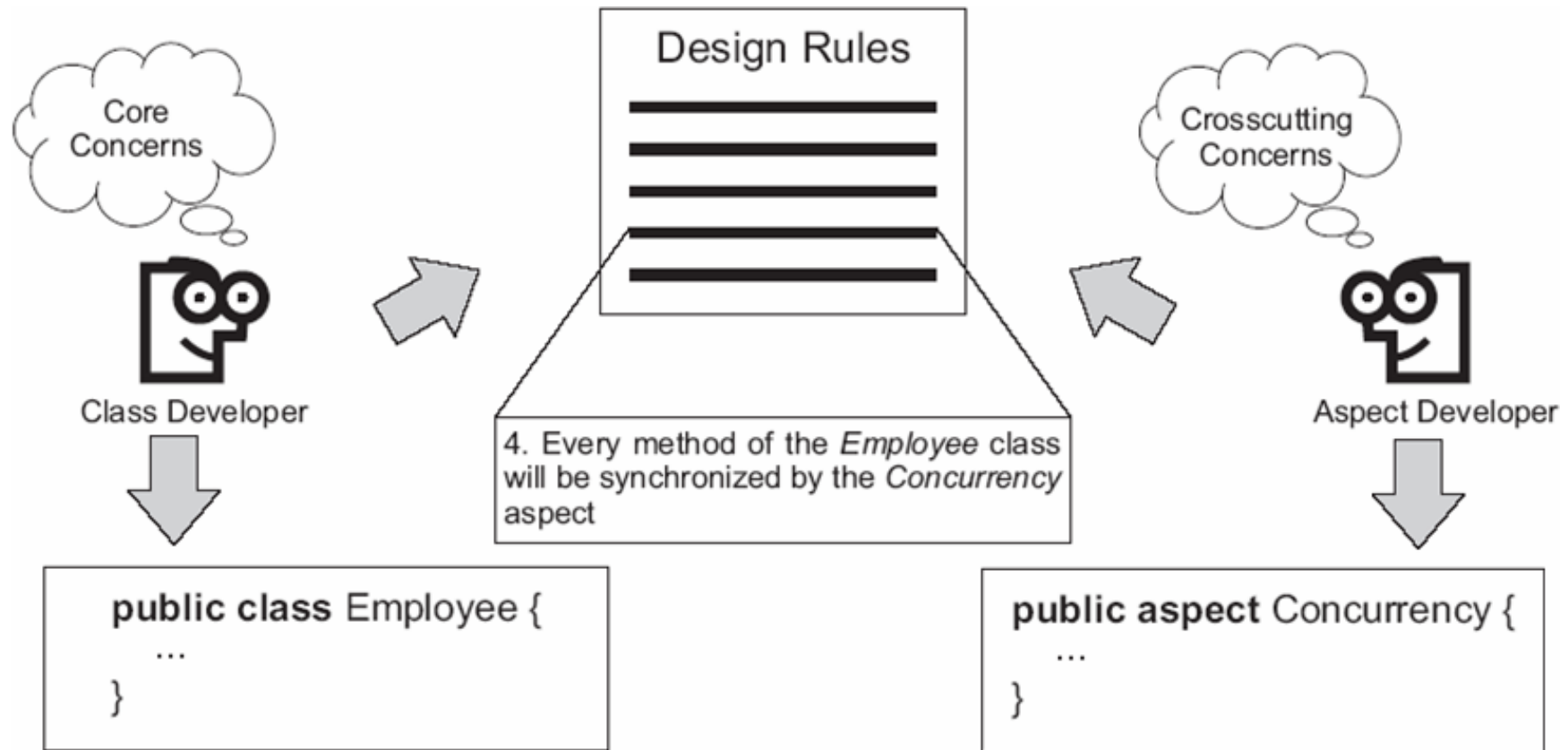
1 Conceptual Model	-															
2 IFPersistence		-														
3 IFView	4		-													
4 IFData	7			-												
5 IFPatterns					-											
6 Distribution						-										
7 Persistence							-									
8 Concurrency								-								
9 Model									-					1	10	7
10 View	35		17						3	-	1				7	4
11 Business	10		1	5							-			1	1	8
12 Data	19	6					12					-		3		
13 Patterns	22	1		15								10	-			2
14 Aspects - Concurrency	2						3	1		1	1	3	1			1
15 Aspects - Distribution	16		1			7				11	1				-	
16 Aspects - Persistence		4									7		1			-
17 Aspects - Patterns	32		2							3	31		9			-

---

# DSM with Semantic Coupling Analysis

- If we do not consider Semantic Dependencies we can **derive inconsistent analysis**
- We have to **improve** the existing AOP **coupling metrics**

# New Design Rules



**DR's can reduce the syntactic and semantic dependencies between aspects and HW components**

---

# New Design Rules

- All classes in **model components** must be **serializable**
  - Support the distribution concern
- The **method** used to **insert an employee** must deal with **synchronization**
- All **business facade** methods must have a **transactional management support**

# DSM with new Design Rules

		Design Rules									Lib			Health Watcher				Aspects				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	Conceptual Model	-																				
2	IFPersistence		-																			
3	IFView	4		-																		
4	IFData	7			-																	
5	IFPatterns					-																
6	DRConcurrency						-															
7	DRDistribution							-														
8	DRPersistence								-													
9	DRPatterns									-												
10	Distribution										-											
11	Persistence											-										
12	Concurrency												-									
13	Model													-								
14	View	35		17										3	-		1					
15	Business	10		1	5																	
16	Data	19	6									12										
17	Patterns	22	1		15												10	-				
18	Aspects - Concurrency	2										3	1							-		1
19	Aspects - Distribution	16		1																	-	
20	Aspects - Persistence		4																			-
21	Aspects - Patterns	32		2																		-

---

# DSM with new Design Rules Analysis

- Dependencies were redirected to the DRs
- Requires class and aspect developers agreement about the DRs

---

# Experience

- Analyzed different HW versions
- The Notion of Semantic Dependency
  - Must be considered for measuring coupling
  - Impossibility to develop in parallel
- DSM are useful to reason about dependencies between a great number of components
- Version with DR's provides better modularization

# Semantic Dependencies and Modularity of Aspect-Oriented Software

Alberto Costa Neto  
Márcio de Medeiros Ribeiro  
Marcos Dósea  
Rodrigo Bonifácio  
Paulo Borba  
**Federal University of Pernambuco**  
{acn, mmr3, mbd2, rba2, phmb}  
@cin.ufpe.br

Sérgio Soares  
**Pernambuco State University**  
sergio@dsc.upe.br



SOFTWARE • PRODUCTIVITY • GROUP

