



# Modularity Anomaly Types – Part II

Alessandro Garcia

LES | DI | PUC-Rio - Brazil



## Relation between architectural problems...

◆ ... and code anomalies

```

public class SearchData{
  //other methods
  void doPost(..){//complex}
  void doGet(..){//complex}
  void execute(..){
    try{ ... }
    catch(Transaction t){..}
    catch(Persistence p){..}
    //14 lines of code removed
    catch(Repository r){..}
  }
}
    
```

**Legend**

- g** GUI
- b** Business
- p** Persistence
- t** Transaction
- Expected flow
- - - Architectural Violation
- Package
- ▣ Architecture module

## Relevance of Code Anomalies

Architecturally Relevant

```

class HWFacade{
    public void updateComplaint(..){..}
    public Complaint searchComplaint(..){..}
    public void insertComplaint(..){..}

    public void insertEmployee(..){..}
    public Employee searchEmployee(..){..}
    public void updateEmployee(..){..}

    public void insertSymptom(..){..}
    public Symptom searchSymptom(..){..}
    public void updateSymptom(..){..}
    ...
}
        
```

**Architectural Interface Bloat**

3

## Relevance of Code Anomalies

Architecturally Irrelevant

```

class ComplaintRepo{
    public int insert(..){..}
    public void update(..){..}
    public int getIndex(..){..}
    public boolean exists(..){..}
    public Complaint search(..){..}
    public void reset(..){..}
    public Object next(..){..}
    public void remove(..){..}
    public List getList(..){..}
    public boolean hasNext(..){..}
    public void updateTimestamp(..){..}
    public int searchTimestamp(..){..}
    ...
}
        
```

4

## Three Questions

- 1 Are anomalous code elements and architecture problems related?
- 2 If so, which characteristics of the code anomaly are relevant for the architecture design?
- 3 To what extent the applied refactorings actually addressed architecturally-relevant code anomalies?

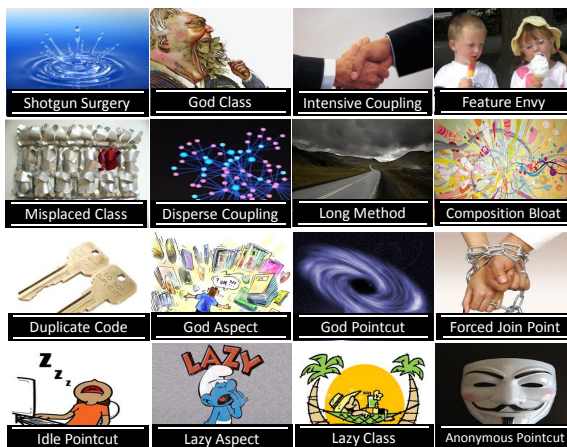
5

### Target systems

MIDAS  
PDP

Mobile Media  
Health Watcher  
Aspectual Watcher  
Aspectual Mobile

### Code anomalies



### Architectural anomalies



## Target Systems

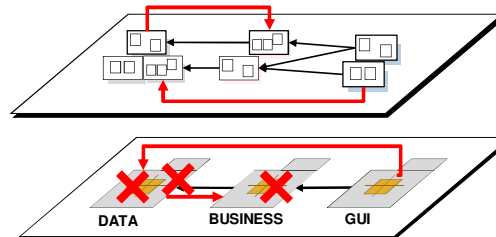
MIDAS	MM	HW	PDP
C++	Java	Java	C#
76 KLOC	54 KLOC	49 KLOC	22 KLOC
111 anomalies	170 anomalies	252 anomalies	175 anomalies

- ◆ 6 different systems
- ◆ 40 revisions
- ◆ Architecture information available

7

## Analysis

- Recovering **Actual Architecture**
- Identifying Architecture Anomalies
- Detecting Code Anomalies
- Analyzing the Impact of Code Anomalies



Q1 Are anomalous code elements and architecture problems related?

8

## Analyzing the Impact of Code Anomalies

- ◆ Downstream Analysis
  - ◆ Which architecture problems are related to code anomalies

9

## Analyzing the Impact of Code Anomalies

- ◆ Downstream Analysis

Category	Caused by Code Anomalies (%)	Not Caused by Code Anomalies (%)
HW	85	15
MM	80	20
PDP	70	30
MIDAS	70	30

10

## Analyzing the Impact of Code Anomalies

- ◆ Upstream Analysis
  - ◆ Which code anomalies are related to architecture problems

11

## Analyzing the Impact of Code Anomalies

- ◆ Upstream Analysis

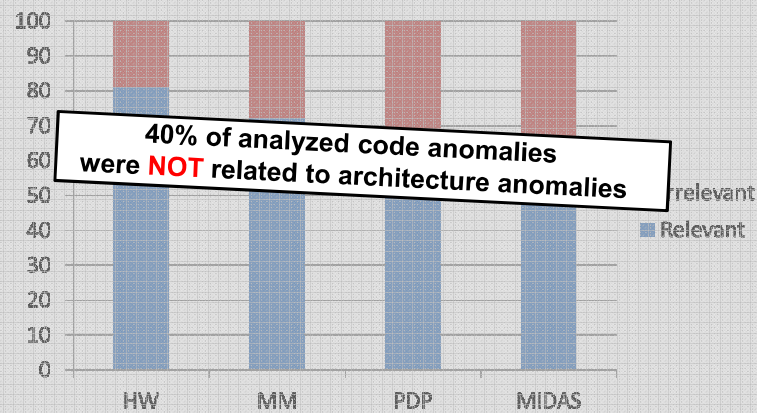
Category	Relevant (%)	Irrelevant (%)
HW	80	20
MM	72	28
PDP	65	35
MIDAS	50	50

12



## Analyzing the Impact of Code Anomalies

### ◆ Upstream Analysis



13



## Three Questions

- 1 Are anomalous code elements and architecture problems related?
- 2 If so, which characteristics of the code anomaly are relevant for the architecture design?
- 3 To what extent the applied refactorings actually addressed architecturally-relevant code anomalies?

14

## Identifying Relevant Code Anomalies

- ◆ The following characteristics were analyzed:
  - ◆ **Code anomaly type**
  - ◆ Change density
  - ◆ Error density
  - ◆ Anomaly density
  - ◆ And others...

Q2 which characteristics of code anomalies are architecturally-relevant?

## Type of Code Anomaly

Kinds of Code Anomalies	AW	HW	AM	MM
Divergent Change	5	7	5	5
Feature Creep	5	7	5	2
God Class	5	7	5	5
Inappropriate Intimacy	6	7	7	8
Long Method	7	8	4	6
Long Parameter List	7	2	5	3
Misplaced Class	1	1	2	3
Shotgun Surgery	3	2	4	5
Small Class	4	7	3	6

None of the code anomalies was the best indicator across all analyzed systems





## Identifying Relevant Code Anomalies

- ◆ The following characteristics were analyzed:
  - ◆ Change density
  - ◆ Error density
  - ◆ Anomaly density



which characteristics of code anomalies are architecturally-relevant?

17



## Studying prioritization models

- ◆ Which other characteristics could be explored for detecting architecturally-relevant code anomalies ?
  - ◆ Change density
  - ◆ Error density
  - ◆ Anomaly density



which characteristics of code anomalies are architecturally-relevant?




**R. Arcoverde** et al – **RSSE/ICSE 2012**: Automatically Detecting Architecturally-Relevant Code Anomalies




**R. Arcoverde** et al – **SBES 2013**: *Prioritization of Code Anomalies Based on Architecture Sensitiveness*. SBES'13) Brasilia, Brazil, September 2013.

18

 Prioritization heuristics				
<b>Change density</b>	System	# of Ranked CE	Arch. Relevant	%
	HW	14	10	71%
	MM	10	7	70%
	<b>PDP</b>	<b>10</b>	<b>10</b>	<b>100%</b>
<b>Error density</b>	System	# of Ranked CE	Arch. Relevant	%
	<b>HW</b>	<b>14</b>	<b>12</b>	<b>85%</b>
	MM	10	8	70%
	PDP	10	8	70%
<b>Anomaly density</b>	System	# of Ranked CE	Arch. Relevant	%
	HW	10	7	60%
	MM	10	9	70%
	PDP	10	8	70%
	<b>MIDAS</b>	<b>10</b>	<b>6</b>	<b>90%</b>

19

 Prioritization Factors	
..	<ol style="list-style-type: none"> <li><b>1. There is no 'universal' prioritization model</b></li> <li><b>2. Prioritization models: satisfactory results too late</b></li> </ol>

May 16

20

Prioritization heuristics				
<b>Change density</b>	System	# of Ranked CE	Arch. Relevant	%
	HW	14	10	<b>Version 12</b>
	MM	10	7	
	<b>PDP</b>	<b>10</b>	<b>10</b>	<b>100%</b>
<b>Error density</b>	System	# of Ranked CE	Arch. Relevant	%
	<b>HW</b>	<b>14</b>	<b>12</b>	<b>85%</b>
	MM	10	8	<b>Version 9</b>
	PDP	10	8	
<b>Anomaly density</b>	System	# of Ranked CE	Arch. Relevant	%
	HW	10	7	
	MM	10	9	<b>Version 10</b>
	PDP	10	8	
	<b>MIDAS</b>	<b>10</b>	<b>6</b>	<b>90%</b>

21

## Earliness of Anomaly

- Early anomaly: appears in the 1<sup>st</sup> version of each system

**18%**

Of all architecturally-relevant code anomalies were identified as **early anomalies**

22

## Earliness of Anomaly

- ◆ Early anomaly: appears in the 1<sup>st</sup> version of each system

**18%**

Of all architecturally-relevant code anomalies were identified as **early anomalies**

and were related to more than

**37%**

of all architecture problems

23

## However, we observed that...

- ◆ ... architecturally-relevant code anomalies tend to “flock together” even in the 1st version of each system:
  - ◆ anomaly agglomerations within a class or syntactically-related classes can be used as good indicator
- ◆ Continuous detection of emerging anomaly agglomerations?

May 16

Alessandro Garcia @ OPUS Group

24


## Three Questions

- 1 Are anomalous code elements and architecture problems related?
- 2 If so, which characteristics of the code anomaly are relevant for the architecture design?
- 3 To what extent the applied refactorings actually addressed architecturally-relevant code anomalies?


25

## Refactoring of Relevant Anomalies

- ◆ 658 refactorings
  - ◆ 33% high-level
    - ◆ Move member (16%)
    - ◆ Extract class or superclass (12%)
  - ◆ 67% low-level
    - ◆ Rename (32%)
    - ◆ Extract local variable (16%)
- ◆ **37%** of all architecture-relevant anomalies were refactored
- ◆ Isolated versions concentrated most of the refactoring efforts

 architecturally-relevant refactorings?


26



## Concluding Remarks

Most architecture problems are related to anomalous code elements

27




## Concluding Remarks

Most architecture problems are related to anomalous code elements

Anomaly agglomerations are more frequently related to architecture problems

28




## Concluding Remarks

Most architecture problems are related to anomalous code elements

Anomaly agglomerations are more frequently related to architecture problems

Architecturally-relevant anomalies are not frequently refactored

29



## Concluding Remarks

Most architecture problems are related to anomalous code elements

Anomaly agglomerations are more frequently related to architecture problems

Architecturally-relevant anomalies are not frequently refactored

Detecting early code anomalies is a needed asset for assisting developers when prioritizing refactoring efforts

30