Chasing Critical Code Smells with *JSpIRIT*

J. Andres Díaz-Pace, Santiago Vidal, Claudia Marcos, & Alessandro García

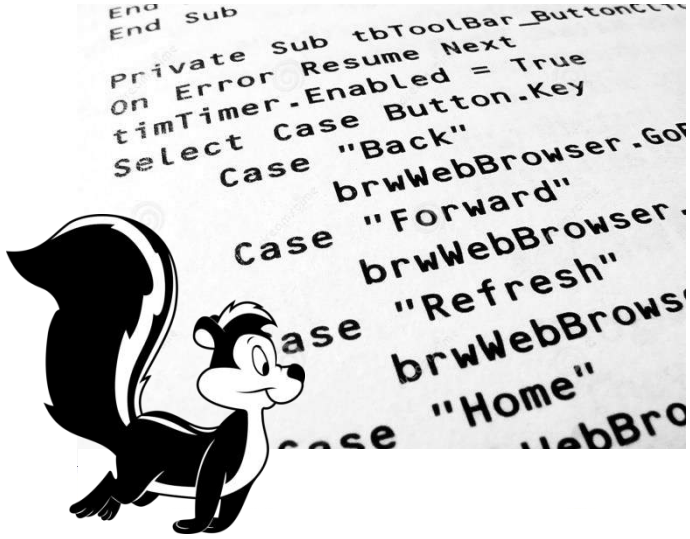
Email: adiaz@exa.unicen.edu.ar

Agenda

- Code smells as symptoms of design problems
- Detection + **Prioritization criteria**
  - JSpIRIT tool & framework
  - 3 criteria based on architectural information
- Demo: Mobile Media
- Lessons learned
- Next steps

About Code/Design smelling bad -1

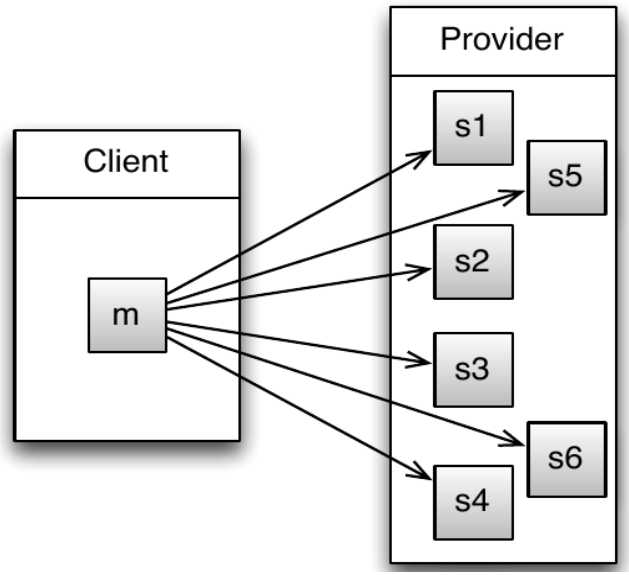
- Any software system evolves
  → **architecture decay**
  - Drift
  - Erosion
- **Technical debt** not properly managed
- … or simply **bad design decisions** that slipped off since early stages



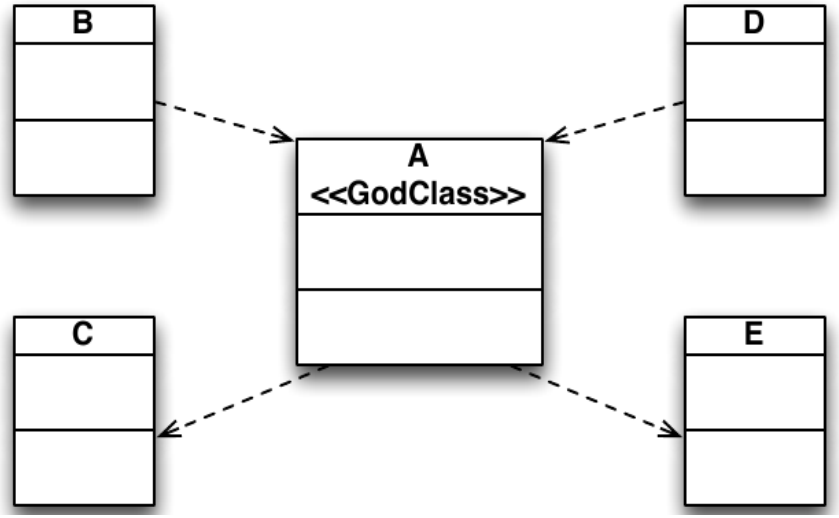
Code Smells [Fowler1999]
(or code anomalies)

Symptoms in the source code
that can indicate a (structural) problem

About Code/Design smelling bad -2



Intensive Coupling



God Class

Symptoms in the source code
that can indicate a (structural) problem

… should they (all) be refactored?

Code Smells Catalog & Tools

- God class
- Brain class
- Brain method (BM)
- Data class
- Dispersed coupling (DE)
- Feature envy (FE)
- Intensive coupling
- Refuse parent bequest
- Shotgun surgery
- …

$$GodClass = (WMC > VERYHIGH) and$$
$$ATFD > FEW) and (TCC < LOW)$$

Automated detection strategies

→ **metric-based**

[Lanza&Marinescu2016]

- Some available **tools**
  - JDeodorant
  - iPlasma & inFusion
  - DECOR
  - PMD
  - SonarQube (plugins)
  - …

Smells with different granularity

Not all tools always agree on their outputs

Smells are not created equally …

- **Problems** of existing tools
  - Numerous code smells are reported (including false positives)
  - Developers get quickly overwhelmed
  - Budget for refactoring is often limited
  - The refactoring of particular smells contribute differently to the system goals or its health

Need to
prioritize "critical" smells

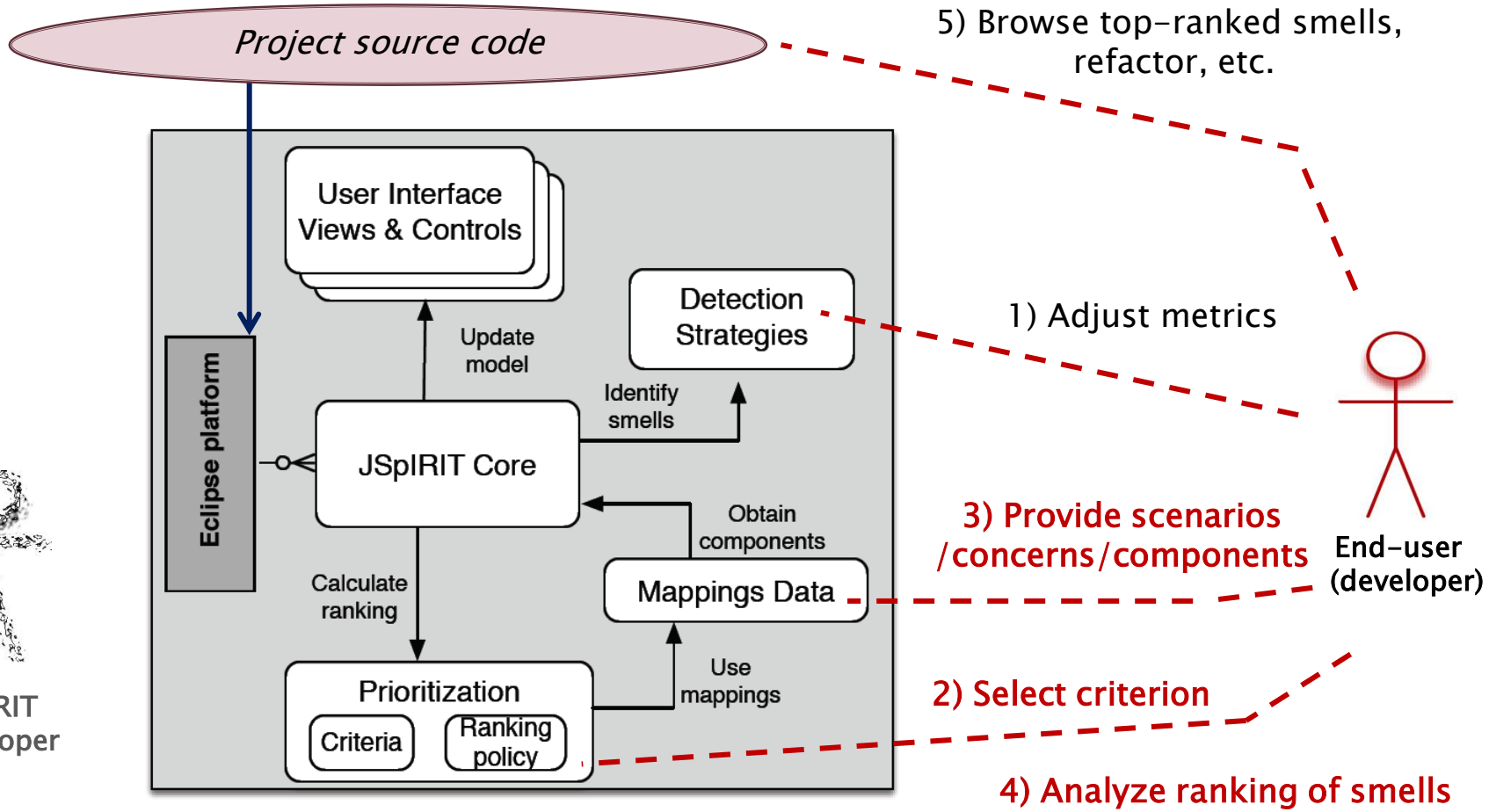
[Vidal2014, Vidal 2015]

Using **architectural information** to inform prioritization criteria

1. **Modifiability scenarios**
2. **Architectural concerns**
3. **Architectural components**
4. **Change history**

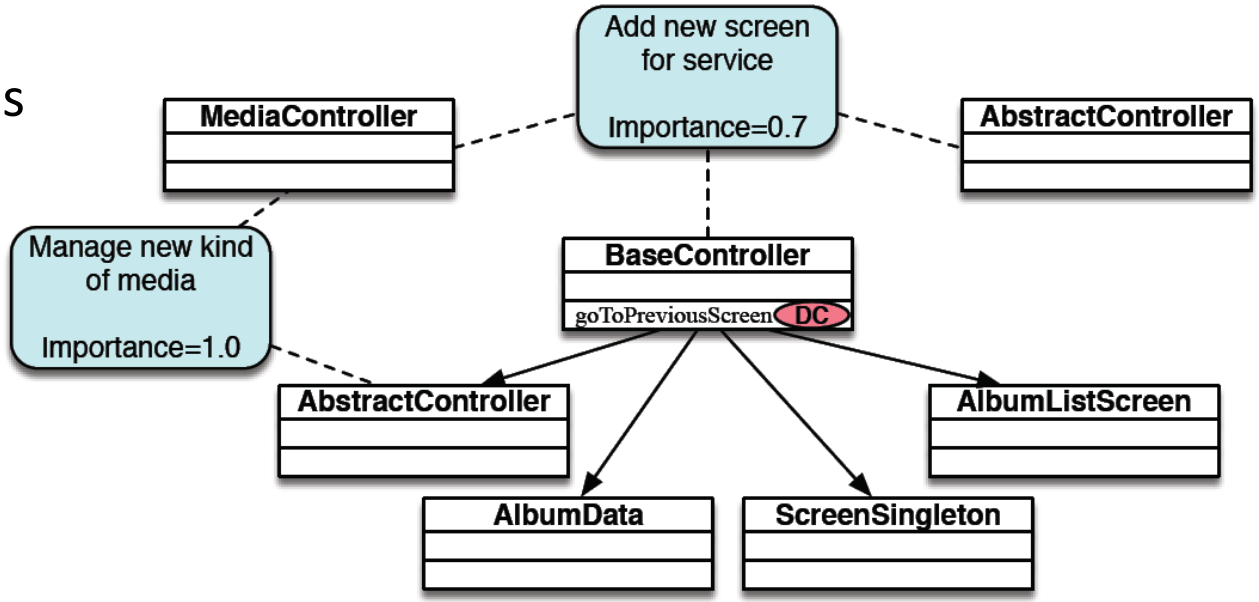
JSpIRIT - Architecture

http://tinyurl.com/j2qoypc



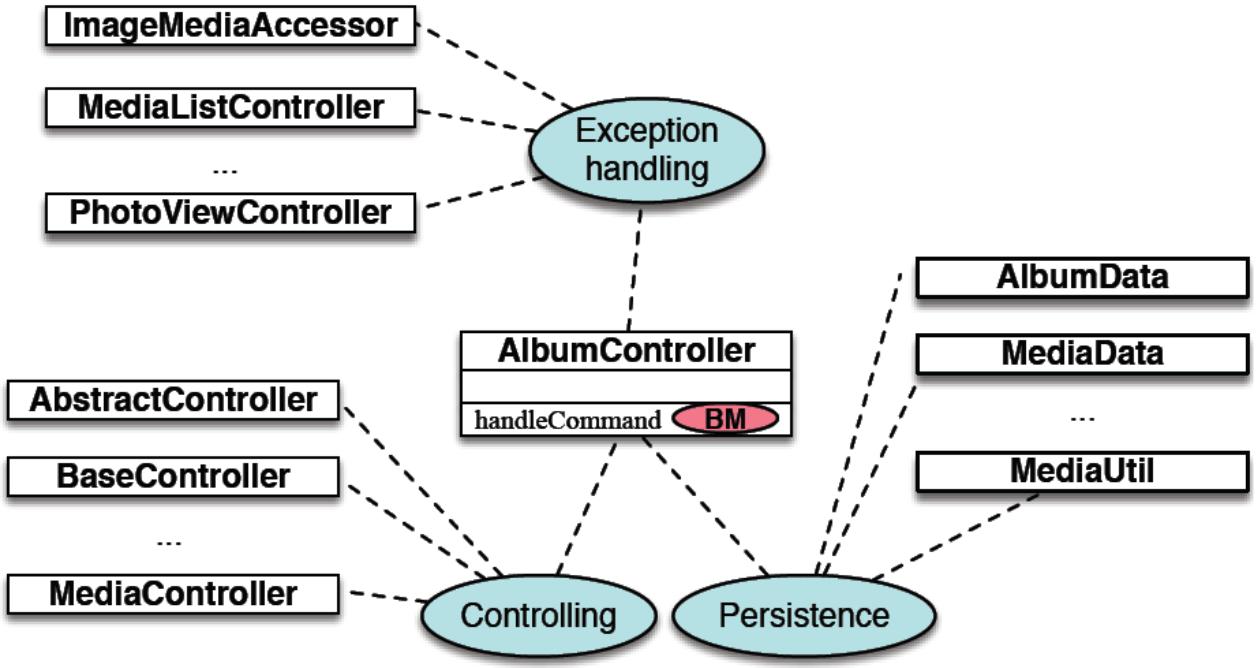
Criterion #1: Modifiability Scenarios

- Relationship of code smells with **quality-attribute scenarios**
  - A **change-related property** that is desirable in the system
    - e.g., *A developer wishes to add a new kind of screen for displaying a picture. This change should be made at design time, and it should take less than 1 hour to make and test the change, with no side-effect changes.*
- Intuition: Smells that intersect with the elements affected by a scenario are more critical (than other smells)
  - Rippling effect to other classes
- Scenario mappings to code are necessary



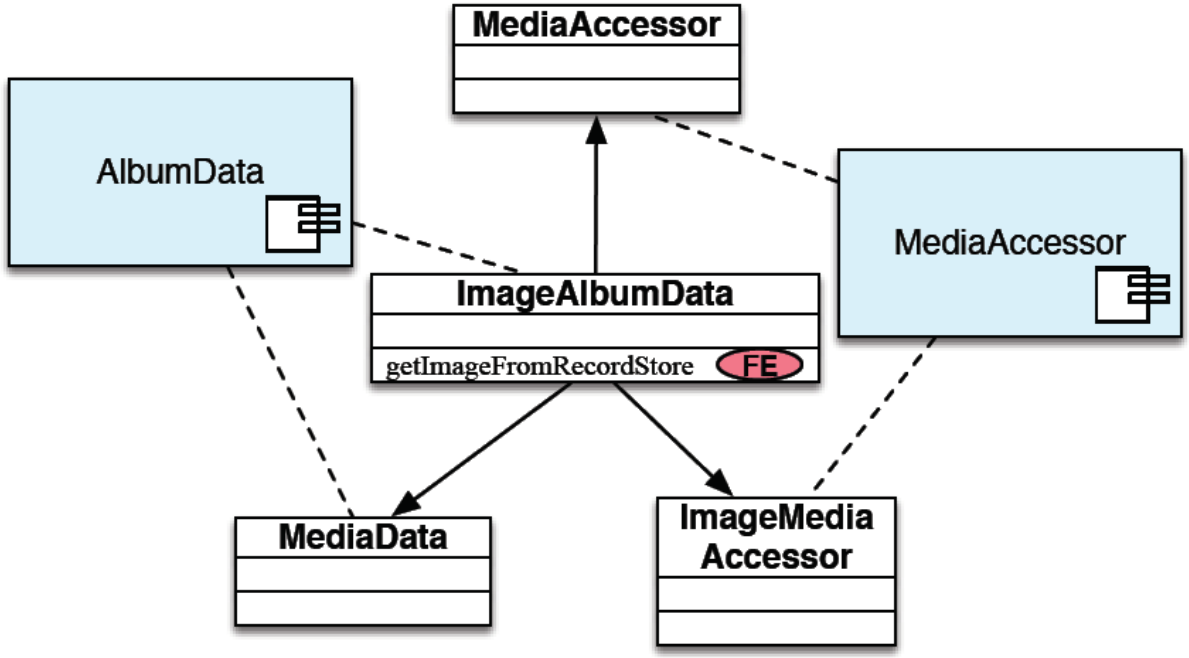
Criterion #2: Architectural Concerns

- A **crosscutting concern** represents some important part of the problem (or the domain) that designers want to treat in a modular way
- Intuition: Smells affected by several concerns are more critical
  - Possible concern entangling
  - Rippling effect
- Concern mappings to code are necessary

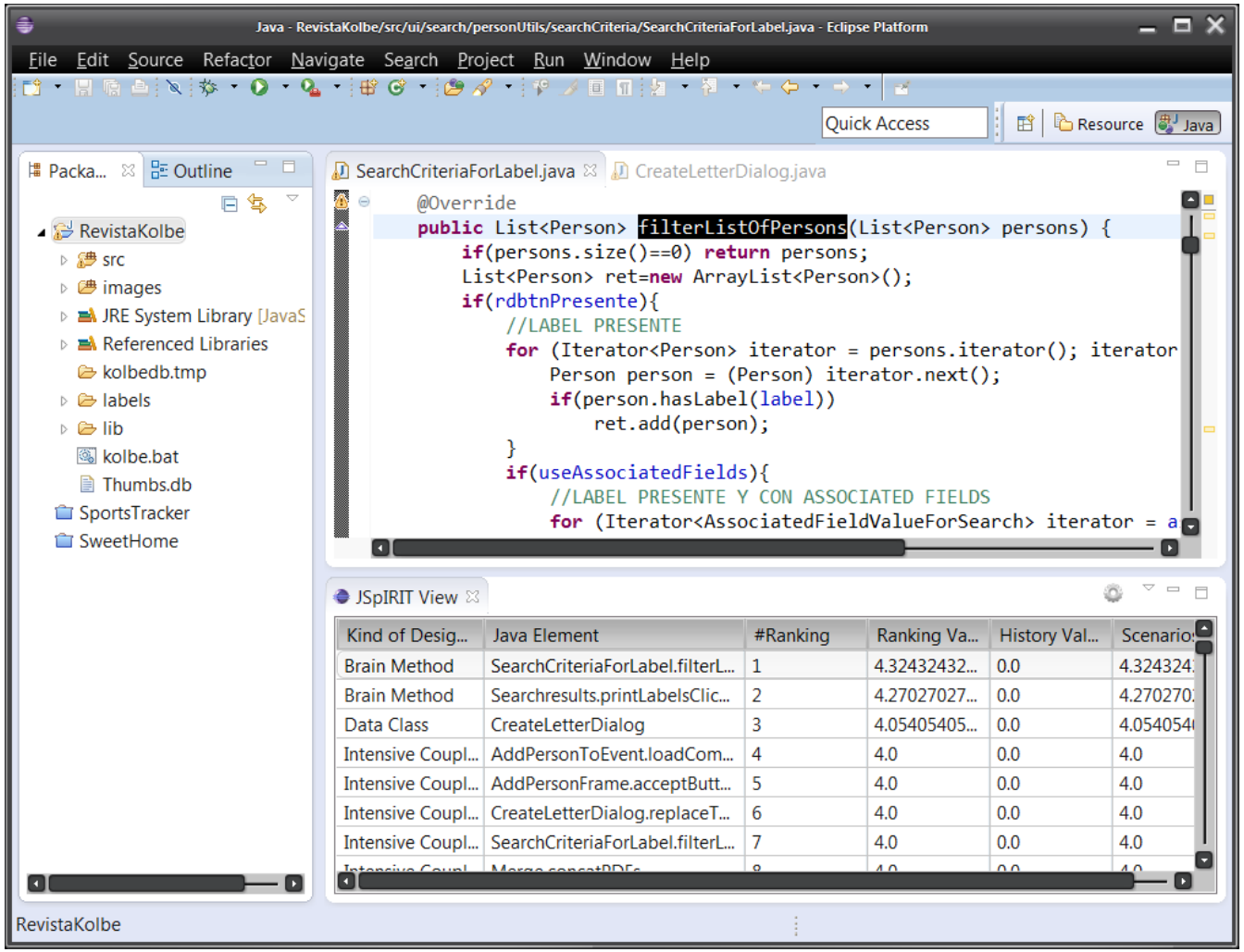


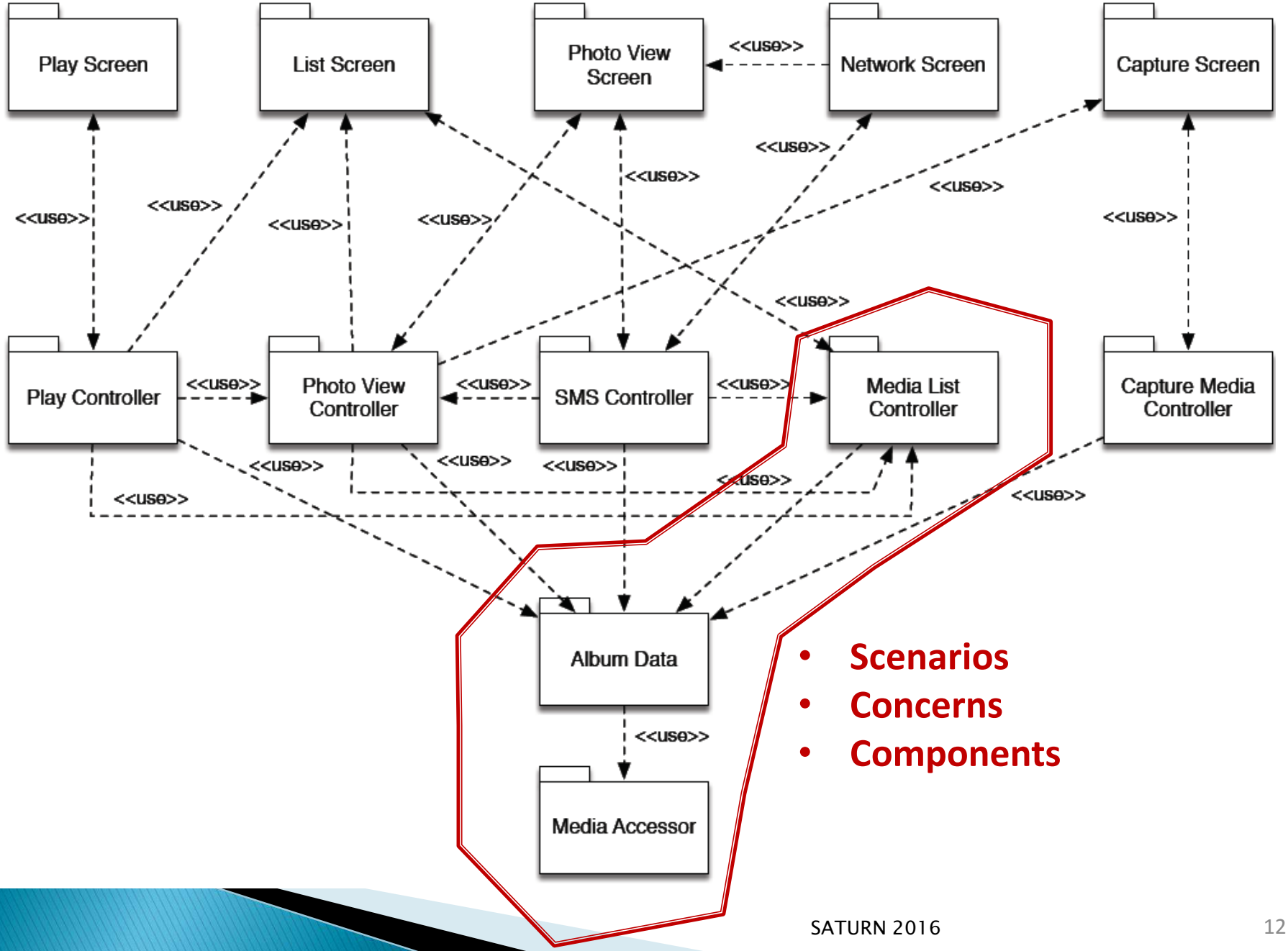
Criterion #3: Architectural Components

- Filtering smells that belong to a given architectural component
- Intuition: Assumption that **certain components are more critical** (or sensitive to changes) than others
  - Business logic
  - Legacy



JSpIRIT Demo: Mobile Media




Play Screen
List Screen
Photo View Screen
Network Screen
Capture Screen
Play Controller
Photo View Controller
SMS Controller
Media List Controller
Capture Media Controller
Album Data
Media Accessor
<<use>>
• Scenarios
• Concerns
• Components

Experiences: Understand & do refactoring

- So far applied in **5 Java systems**
  - Health Watcher (academic, 8 KLOC, 372 classes, Web, 58 smells)
  - Mobile Media (academic, 5 KLOC, 50 classes, mobile SPL, 45 smells)
  - SubscriberDB (8 KLOC, 193 classes, desktop, 82 smells)
  - Beef-Cattle Farm Simulator (industrial, 38 KLOC, desktop, 372 classes, **523 smells**)
  - HR Appraiser (industrial, 83 KLOC, **884 smells**, in progress)

Liveware
Ingeniería de Software S.A.

- **Good precision: ~70% of top-10 smells were critical**
  - Based on validation of rankings against "key classes" or design problems judged by system experts
  - Combination with **change-related historical information**, which helps to discard smells in "stable/unchanged" classes (not covered here)
    - Depends on quality of code mappings

Conclusion & Next Steps

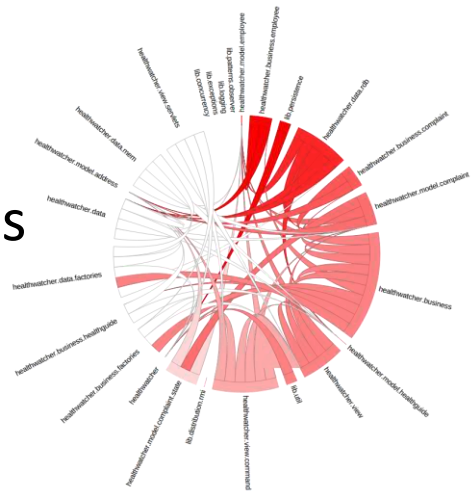
- Analysis/prioritization for **other quality-attribute scenarios**
  - Performance (static + dynamic analyses)
    joint project with RSS Group, at Stuttgart University
  - Battery consumption (Android)
- Groups of code smells → **agglomerations**
  - Stronger indicators of design problems
    joint project with Opus Group, at PUC-Rio University

RSS Reliable Software Systems

Kieker

OPUS Group

- **New version of JSpIRIT to be released**
- Better visualization capabilities
  - For both code smells and agglomerations
- Automated search for **refactoring** suggestions
  - Initial experiments with Brain Method



Thank you! & QA

Santiago Vidal
svidal@exa.unicen.edu.ar

J. Andres Diaz-Pace
adiaz@exa.unicen.edu.ar

Claudia Marcos
cmarcos@exa.unicen.edu.ar

Alessandro F. Garcia
afgarcia@inf.puc-rio.br

JSpIRIT

Java Smart Identification of Refactoring opportunITies