



# Chasing Critical Code Smells with *JSpirit*

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

Email: [adiaz@exa.unicen.edu.ar](mailto:adiaz@exa.unicen.edu.ar)



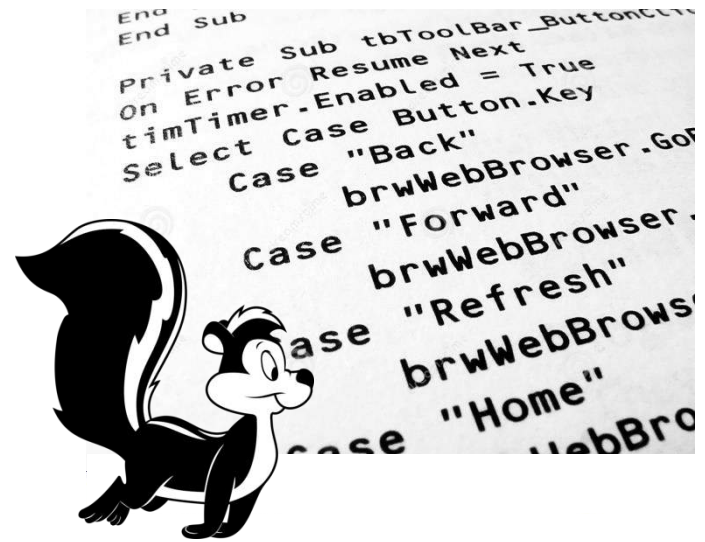
# Agenda

- 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

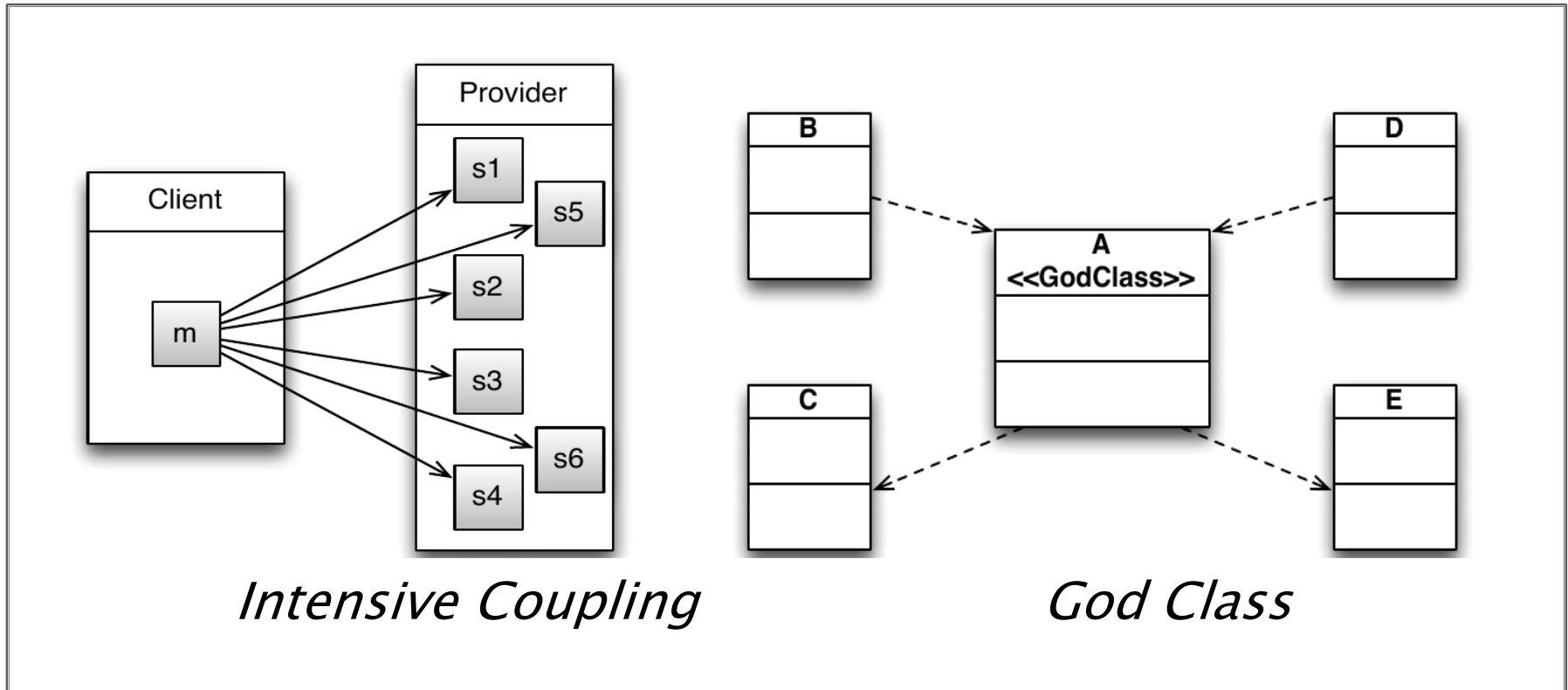
- Any software system evolves  
→ **architecture decay**
  - Drift
  - Erosion
- **Technical debt** not properly managed
- ... or simply **bad design decisions** that slipped off since early stages



**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)*

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

Automated detection strategies

→ **metric-based**

[Lanza&Marinescu2016]

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**

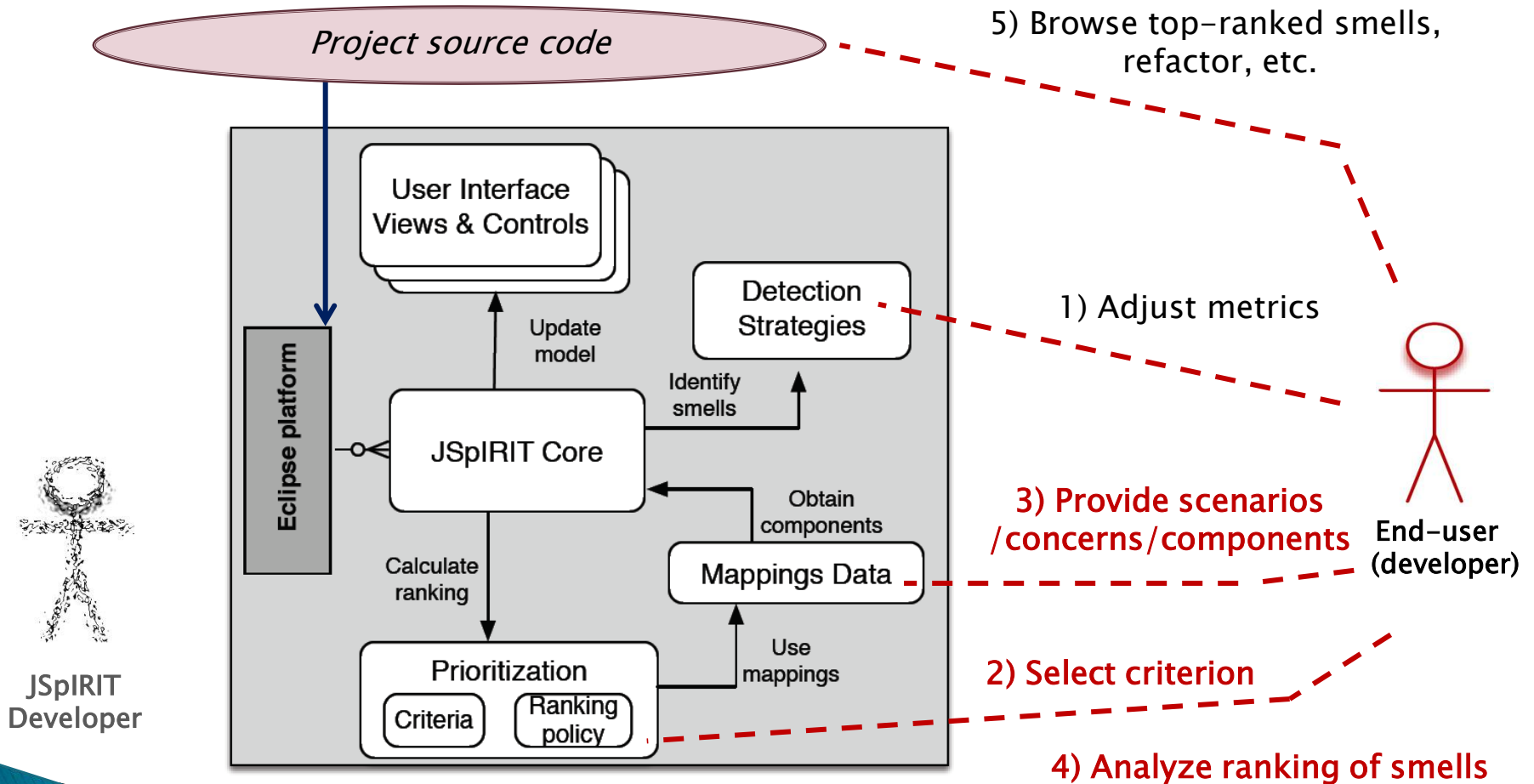


Java Smart Identification of Refactoring opportunITies

SATURN 2016

# JSpIRIT - Architecture

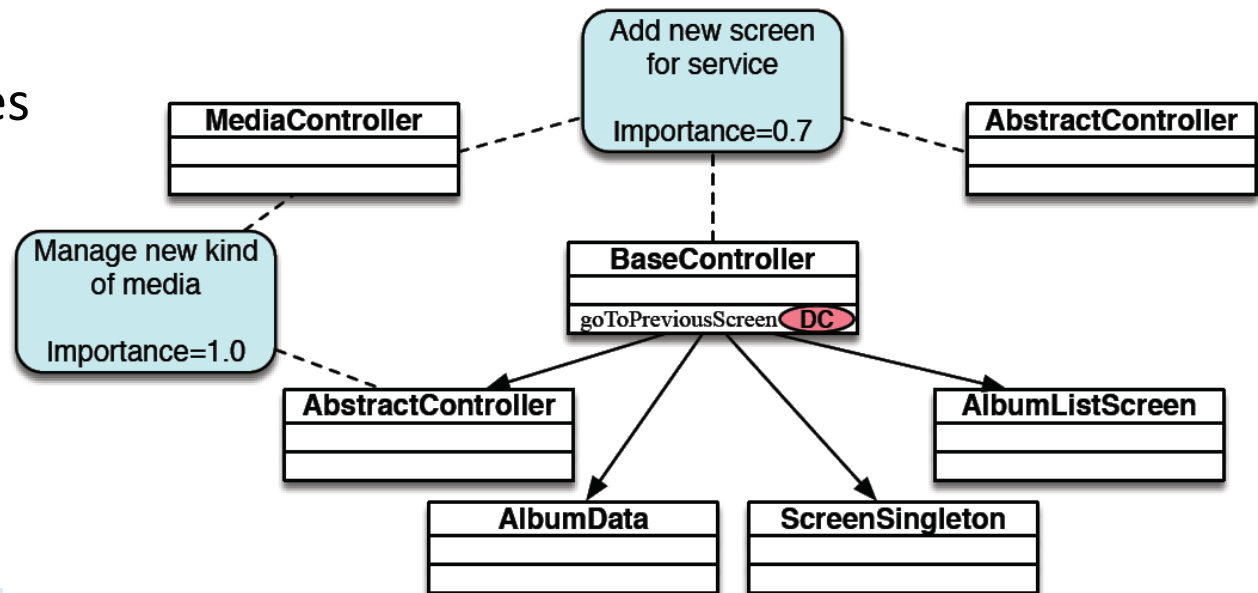
<http://tinyurl.com/j2qoypc>



# 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

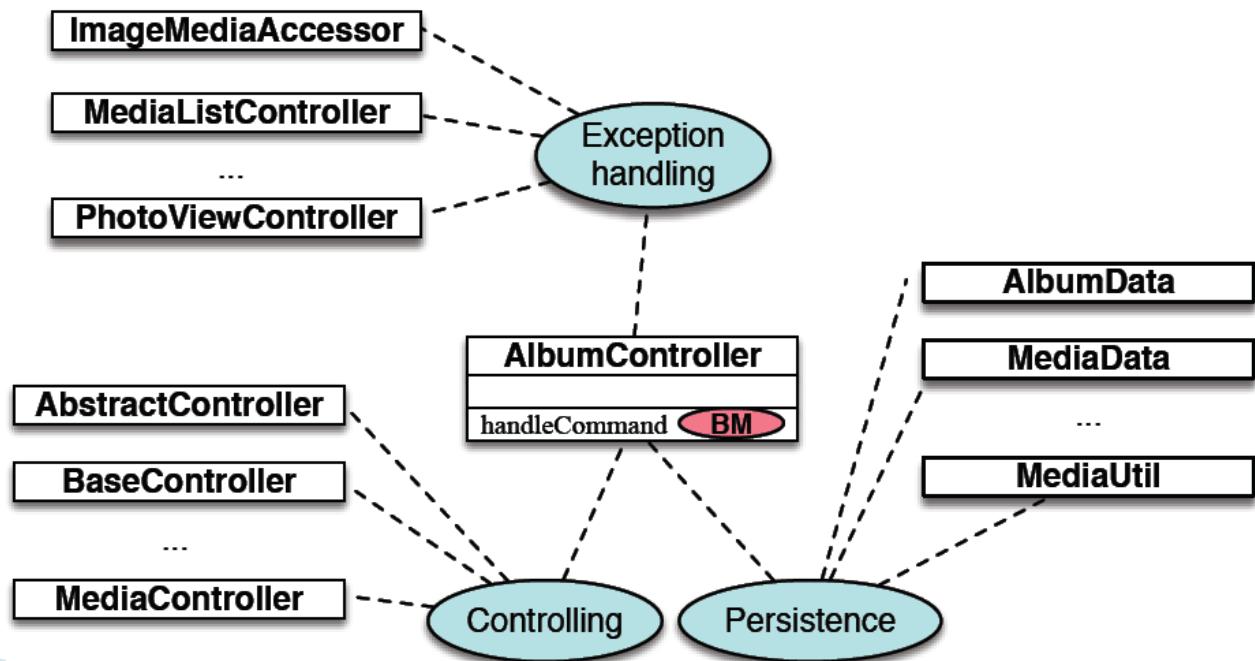




# Criterion #2: Architectural Concerns

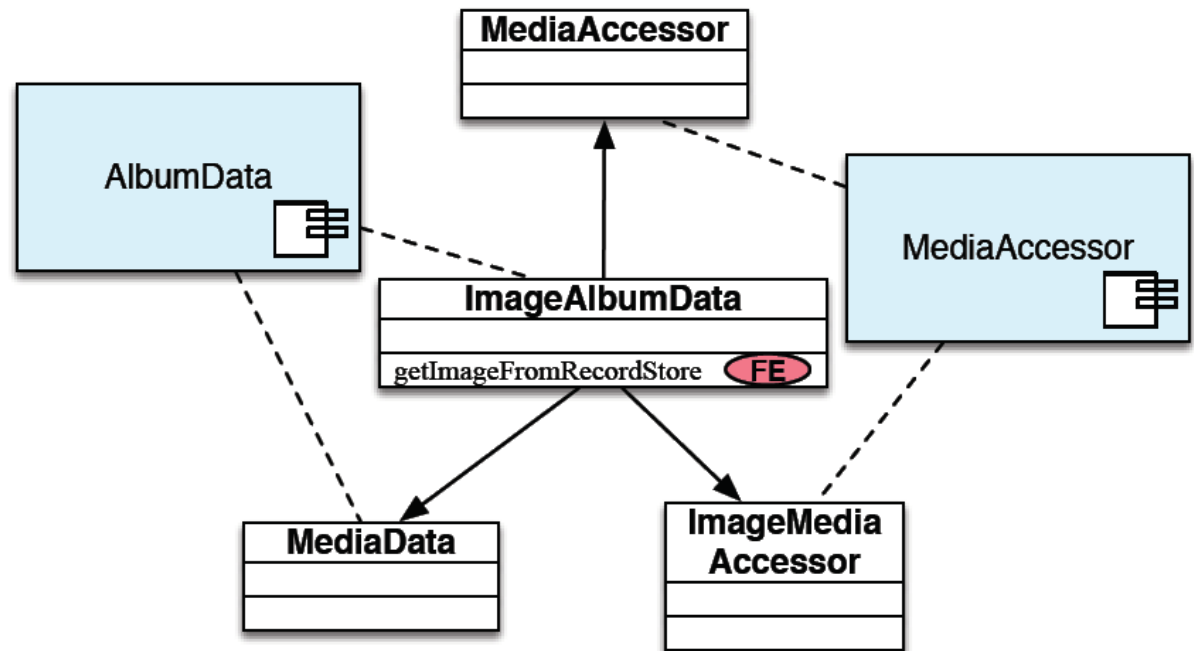
- A **crosscutting concern** represents some important part of the problem (or the domain) that designers want to treat in a modular way
- Intuition: Smells affected by several concerns are more critical
  - Possible concern entangling
  - Rippling effect

- 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

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development. The left sidebar shows the Project Explorer with the following structure:

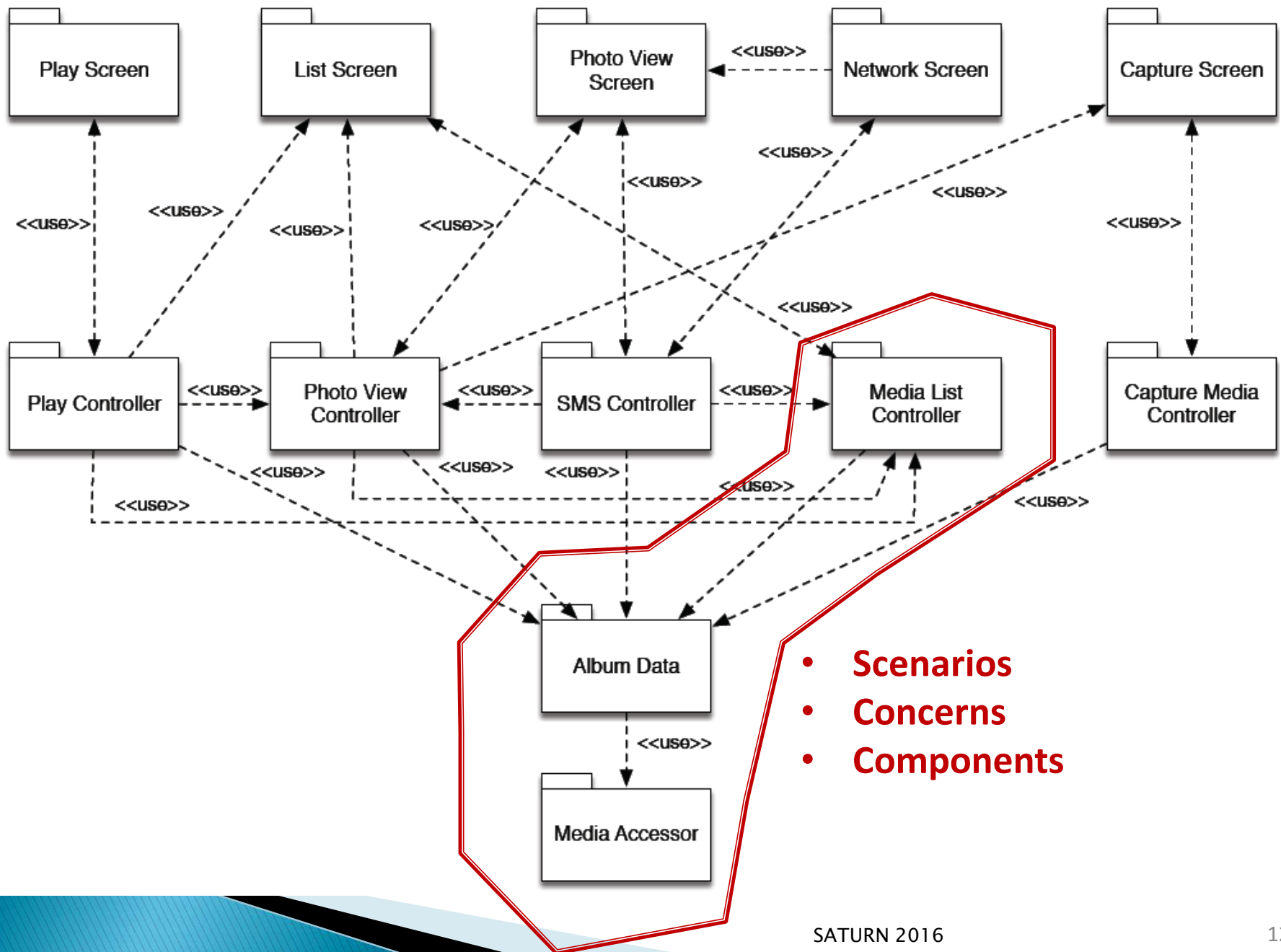
- RevistaKolbe
  - src
  - images
  - JRE System Library [JavaS
  - Referenced Libraries
    - kolbedb.tmp
  - labels
  - lib
  - kolbe.bat
  - Thumbs.db
  - SportsTracker
  - SweetHome

The main editor displays the code for `SearchCriteriaForLabel.java`. The code is as follows:

```
@Override
public List<Person> filterListOfPersons(List<Person> persons) {
    if(persons.size()==0) return persons;
    List<Person> ret=new ArrayList<Person>();
    if(rdbtnPresente){
        //LABEL PRESENTE
        for (Iterator<Person> iterator = persons.iterator(); iterator
            Person person = (Person) iterator.next();
            if(person.hasLabel(label))
                ret.add(person);
        }
    }
    if(useAssociatedFields){
        //LABEL PRESENTE Y CON ASSOCIATED FIELDS
        for (Iterator<AssociatedFieldValueForSearch> iterator = a
```

The JSpIRIT View at the bottom shows a table with the following data:

Kind of Desig...	Java Element	#Ranking	Ranking Va...	History Val...	Scenario...
Brain Method	SearchCriteriaForLabel.filterL...	1	4.32432432...	0.0	4.324324...
Brain Method	Searchresults.printLabelsClic...	2	4.27027027...	0.0	4.270270...
Data Class	CreateLetterDialog	3	4.05405405...	0.0	4.054054...
Intensive Coupl...	AddPersonToEvent.loadCom...	4	4.0	0.0	4.0
Intensive Coupl...	AddPersonFrame.acceptButt...	5	4.0	0.0	4.0
Intensive Coupl...	CreateLetterDialog.replaceT...	6	4.0	0.0	4.0
Intensive Coupl...	SearchCriteriaForLabel.filterL...	7	4.0	0.0	4.0
Intensive Coupl...	Merge concatPDFs	8	4.0	0.0	4.0



# 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)



- **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



**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](mailto:svidal@exa.unicen.edu.ar)



J. Andres Diaz-Pace

[adiaz@exa.unicen.edu.ar](mailto:adiaz@exa.unicen.edu.ar)



Claudia Marcos

[cmarcos@exa.unicen.edu.ar](mailto:cmarcos@exa.unicen.edu.ar)



Alessandro F. Garcia

[afgarcia@inf.puc-rio.br](mailto:afgarcia@inf.puc-rio.br)

