Cataloging and Detecting Architectural Bad Smells

Joshua Garcia, Daniel Popescu, and Nenad Medvidovic, University of Southern California

Yuanfang Cai, Drexel University
Motivation – Refactoring Architecture

• Successful systems are maintained over multiple years
• System’s Life-Cycle Properties worsen over time
  • Understandability
  • Testability
  • Extensibility
  • Reusability
• Restructuring/Refactoring helps to improve life-cycle properties
  • Code Smells
• When and where to refactor a software system’s architecture?
Motivation – iRODS - Prescriptive

Source: https://www.irods.org/index.php/Introduction_to_iRODS
Motivation – iRODS - Descriptive
Contribution and Goals

- Categorization of Architectural Smells
  - Components, Connectors, Interfaces, Data Elements, Concerns
  - Separation of Concerns
  - Coupling and Cohesion
- Novel Architectural Recovery Technique
  - Identification of Elements
  - Concern Meta-Classification
- Novel Architecture Representation
  - Extended Augmented Constraint Network
  - Design Structure Matrix
- Architectural Smell Detection
• Component exhibiting interaction-related functionality that should be delegated to connector
  • Reusability, understandability, testability

• StrategyAnalyzerAgent from Emergency Response System
Architectural Recovery for Smells

- **Component identification**
  - Hierarchical clustering
  - Concerns through topic modeling
    - SAA – “strategy,” “rule,” “region”

- **Connector identification**
  - Pattern matching, Supervised Learning

- **Interface and Data Element Identification**

- **Concern Meta-classification**
  - Application-specific or connector-oriented concern
• Extended Augmented Constraint Network
  • Uniform, formal way of capturing of architectural decisions
  • Constraint network, design rule, cluster set, concerns from topic models
Design Structure Matrix of ERS

The matrix shows the relationships between different components and concerns for the ERS system. Each cell indicates the strength of the relationship, with values ranging from 0 to 1. Components are listed on the right and concerns on the top. For example, the relationship between Component_RenderingAgent and Component_ResourceMonitor is indicated by the value in the corresponding cell.
Thank You

• Thank You
Smells of Different Granularities

- **Code smell**
  - Code smells are *implementation* structures that negatively affect system lifecycle properties
- **Defined in terms of implementation-level constructs**
  - Classes
  - Methods
  - Statements
- **Examples**
  - Long parameter list
  - Large methods
- **Code smells do not necessarily address architectural decisions**

- **Architectural smell**
  - A commonly used *architectural* decision that negatively impacts lifecycle properties
- **Possible Causes**
  - Applying a design solution in an inappropriate context
  - Mixing design fragments that have undesirable emergent behaviors
- **Architectural Refactoring – The remedy**
  - Altering the internal structure of the system
  - Altering the behaviors of internal system elements
  - Avoid changing external system behavior
Ambiguous Interfaces – Description

• An *Ambiguous Interface* offers only one public interface
• Internally dispatches to multiple services
•Appears especially in event-based publish-subscribe systems
  • Example: JMS
• User has to inspect the component’s implementation before knowing about its offered services
• Negatively affects
  • Analyzability
  • Understandability
Connector Envy – Description

- **Connector roles**
  - Communication
  - Coordination
  - Conversion
  - Facilitation

- **Components with Connector Envy** encompass extensive interaction-related functionality
  - Example: Gridfarm Filesystem Daemon

- **Violates separation of concerns**

- **Negatively affects**
  - Reusability
  - Understandability
  - Testability
Scattered Functionality – Description

• Multiple components are responsible for realizing the same high-level concern
• Some of those components are responsible for orthogonal concerns
  - Example: Linux’s Status Reporting
• Violates the principle of separation of concerns twice
• Negatively affects
  - Reusability
  - Understandability
  - Testability
Extraneous Connector - Description

• Two connectors of different types are used to link a pair of component
  • Example: Events vs. Procedure Calls
  • Example system: Example: Old MIDAS version

• Benefits of each connector type may cancel each other out

• This example negatively impacts
  • Understandability
  • Reusability
  • Adaptability