Improving Design Smell Detection for Adoption in Industry

Doctoral meeting
Khalid Alkharabsheh

Jose Taboada (USC), Yania Crespo (UVA)
Outline

- Introduction.
- Thesis Proposal.
- State of the Art.
- Thesis plan
Introduction

- Software quality is one of the main important problems for all software engineers and researchers.

- According to Brown*, a survey of hundreds of software development projects show that five from six projects are considered unsuccessful.

- The majority of software development cost (budget) is devoted to maintaining processes.

- More difficulties in controlling the maintenance process than in other phases of the software development life cycle.
  - Reasons:
    - Complexity of source code.
    - Experience of developers.
    - Amount and frequency of maintenance tasks (Adaptive, Corrective, Perfective).
    - Different tools required. (adapting, correcting, documenting, etc).
Refactoring

- A set of restructuring operations that support the design and evolution of software but preserving its observable behavior. [Opdyke1992]*
- A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. [Fowler1999]**

- Identifying pieces of code need to be refactored making the upcoming maintenance tasks easier.

- Refactoring is a technique used to:
  - Make software easier to modify and increase understandability.
  - Remove design smells (Decrease coupling & Increase cohesive).
  - Improve the design of software.

- Well known refactoring operations (Extracting class, Extracting method, Move method).

Design smell

- Design smells are indicators on weak software design that can potentially decrease software maintainability.

- Do not produce compile-time or run-time errors.

- But negatively affect system quality properties, such as understandability, testability, extensibility, reusability and maintainability.

- These problems can appeared in several software artifacts from fine grained to coarse grained including (variables, instructions, operations, methods, classes, packages, sub systems, layers and their dependencies).
Introduction

Historical Data

- Design smells concept cover whole problems related to the software structure (code, design).

- Design smell appear in the state of the art under different terms:
  - Design heuristics 1996.
  - Bad smell 1999.
  - Disharmonies 2006.
  - Design flaws 2006.
  - Design Smell 2011.

- Different terms used to describe the same type of design smell such as:
  - Large class bad smell (class is trying to do too much).
  - God class disharmony (class performs too much work on its own).
  - Blob antipattern (class with responsibilities that overlap most other parts of the system).
Design Smell Example (1)

Bad Smell (Feature Envy):
- Occurs when a method in one class uses primarily data and methods from another class to perform its work.

Fix: (Move Method Refactoring)
Move the method with feature envy to the class containing the most frequently used methods and data items.

```java
public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }
    public String getAreaCode() {
        return unformattedNumber.substring(0, 3);
    }
    public String getPrefix() {
        return unformattedNumber.substring(3, 6);
    }
    public String getNumber() {
        return unformattedNumber.substring(6, 10);
    }
}

public class Customer...
    private Phone mobilePhone;
    public String getMobilePhoneNumber() {
        return "" + mobilePhone.getAreaCode() + "" + mobilePhone.getPrefix() + "" + mobilePhone.getNumber();
    }
}

public class Phone {
    private final String unformattedNumber;
    public Phone(String unformattedNumber) {
        this.unformattedNumber = unformattedNumber;
    }
    private String getAreaCode() {
        return unformattedNumber.substring(0, 3);
    }
    private String getPrefix() {
        return unformattedNumber.substring(3, 6);
    }
    private String getNumber() {
        return unformattedNumber.substring(6, 10);
    }
    public String toFormattedString() {
        return "" + getAreaCode() + "" + getPrefix() + "" + getNumber();
    }
}

public class Customer...
    private Phone mobilePhone;
    public String getMobilePhoneNumber() {
        return mobilePhone.toFormattedString();
    }
}
Design Smell Example (2)

- Architectural Smell (Large Class or God Class or Blob):
  - Occurs when a class is trying to do too much responsibilities or have many methods or instance variables.
  - **Fix:**
  - (Extract Class Refactoring)
    Take a subset of the instance variables and methods and create a new class with them and this makes the initial (long) class shorter.
  - (Move Method Refactoring)
    Move one or more methods to other classes.
Introduction

What is the problem?
- Design smells detection tools are not widely adopted in industry.

Why is it a problem?
- Currently software has huge dimensions and Manual detection is not realistic.
- The available tools can not be identified as useful design smells detection tools that perfectly fits to different software companies/organizations.

Why it is an important problem?
- Increasing the maintainability time and cost.
- Negatively impacts on software quality.
- As a consequence, software lifetime can be shorten.
Thesis Proposal

- **Main Goal**
  - Improve the usefulness of design smell detection tools for adoption in industry to aid in the increase of software quality and maintainability.

- **Sub goals**
  - Study in depth the similarities and differences among smell detection techniques to identify the efficiency factors in design smell detection.
  - Organize the knowledge on design smell detection.
  - Analyze the inter-rater agreement between software smell detection tools (automatic experts), human experts and both of them in determining the expected problems in industrial software projects.
  - Make a comparison between techniques to identify the optimal algorithm.
  - Improve the usefulness of algorithm *(introduce subjectivity, improve adaptability, gray scale, improve efficiency)*.
  - Validation in industrial environment.
Thesis Proposal

- Activity Diagram

2013/2014

2014/2015

2015/2016

2016/2017
State of the Art

State of the Art Activity.

- A comprehensive systematic mapping.
  - Identify state of the art problems.
  - Select a set of design smell detection tools.
  - Select a set of design smells.

- Analyse agreement in detection
  - Tools comparison.
  - Evaluate the tools on a medium size project.
  - Web-based questionnaire survey.
  - Compute inter-raters agreement between tools, human expert and both of them.
Different design smell classifications:

- **Bad Smells (Code Smells)**
  - Defined in terms of implementation level (subsystem, package, class, fields, methods, parameters and statements).

- **Architectural Smells**.
  - Defined in terms of architecture level abstractions (components, connectors and styles).

- **Software Product Line Smells (Variability Smells)**.
  - Design smells specific to SPLs. They can be divided in parts, such as architectural smells and code smells.

- **Hybrid smells**
  - Combine architectural and code smells.

### State of the Art

#### Classifications

- **Popular Design smells**

  - Feature Envy
  - Blob
  - Long method
  - God class
  - Data class
  - Large class
  - Shotgun surgery
  - Spaghetti code
  - Long parameter list
  - Functional decomposition
  - Duplicate code
  - Lazy class
  - Refused Bequest

  **Most interested design smells**

  - Feature Envy
  - Blob
  - Long method
  - God class
  - Data class
  - Large class
  - Shotgun surgery
  - Spaghetti code
  - Long parameter list
  - Functional decomposition
  - Duplicate code
  - Lazy class
  - Refused Bequest

  **Most detected design smells**

  - God class
  - Duplicate code
  - Large class
  - Long method
  - Data class
  - Spaghetti code
  - Long parameter list
  - Feature Envy
  - Lazy class
  - Functional...
  - Blob
  - Refused Bequest
  - Shotgun surgery

#### Approaches

#### Tools

#### Conclusions
Design Smell Detection Approaches

- **Metric-based approach**
  - Detect design smells using existing and new quality metrics by finding relative thresholds values using different techniques and strategies.

- **Rule-based approach**
  - Detect smells depending on facts and rules and relation between metrics.

- **Machine learning approach**
  - Detect smells using learning techniques derived by specific classifiers.

- **Graph-based approach**
  - Represent software artifact in vertices and node to extract the important data and to reason on this model.

- **UML approach**
  - Use UML meta-model.
Design Smell Tools

- A few tools deal with:
  - More than one programming languages and Platform.
  - Analyze large size software.

- Most tools deal with:
  - Limited set of design smells.
  - Mainstream languages (C, C++, Java, C#).
  - Use one input source.
  - Use one representation type.

- Some of tools generate own metrics to identify design smells and others use metrics generated by other tools. (Demo)
Conclusions

- The attention of researchers community modified from Duplicate code to Feature envy design smells and God class is the most detected design smells in software.

- Metric-based, Rule-based and Machine learning approaches related with each others and the majority of researchers like to detect smells using them.

- The most used tools are: JDeodorant, DÉCOR, Together, iPlasma, PMD and SourceMiner.

- Poor inter-rater agreement between:
  - Design Smell Detection tool.
  - Human experts.
  - Tools and Experts.

- All detection tools that identify design smell automatically, detect smells as binary decision (having the smell or not).

- Lack in Empirical studies and Benchmarks availability.
Thesis plan
Improve algorithm

- Activity Diagram
Techniques Comparison & Identify Optimal Algorithm Activity.

- Preparing a dataset with:
  - Wide set of metrics.
  - Classes classification based on UML stereotypes.
  - Different projects size.
  - Different project domains.
  - Different project status.

- Make a comparison between different machine learning techniques.

- Develop improved techniques that are useful for satisfying our goals.

- Validation experiment on improved techniques.
Thesis plan
Improve the usefulness

- **Develop Algorithm Activity.**
  - Improve the dataset in Phase three.
  
  - A gray scale in certain percentage, (God class in class X 70%).
  
  - Priorities on their impacts on Maintainability.
  
  - Implement the improved technique with gray scale.
  
  - Validation experiment on improved algorithm.
Validation The Proposed Algorithm Activity.

- Experiment to evaluate the efficiency of gray scale algorithm on detecting God Class and Feature Envy smells.

- Produce a report were include:
  - Detected smells with gray scale.
  - Priorities on the highest impact on maintainability.

- The developer will compare the last report with the actual state of software.

- Iterative process of modifying the algorithm designed and validation until satisfy the goals.
Thank you for your attention!!!