RIT

Efficient Feature Envy Detection and Refactoring using Graph Neural Networks

Phase 1 - Understanding the Problem and Solution

By: Group4

Srujan Vaddiparthi,

Roopikaa Konidala,

Matt Ballerini

KEYWORDS

- **Technical Debt (TD):** the implied cost of additional work in the future resulting from choosing an expedient solution over a more robust one in software development.
- **Code smell:** serious form of TD which is confusing, complex, or harmful design patterns in source code.
- **Feature Envy:** most common code smell. Refers to a method that is more interested in an external class than its enclosing class. Misplaced method.
- **SMOTE Call Graph (SCG):** a GNN-based approach to address the problems of detecting and refactoring feature envy.
- **Calling Strength:** is a concept that quantifies the strength of method invocations in the call graph.
- **Symmetric Feature Fusion Learning (SFFL):** Obtains graphical representations of methods-classes. Another approach to address the problems of detecting and refactoring feature envy.
- **Invocation:** number of times method is called.
- **Ownership:** the class which the method belongs to.
- **Position:** where the method is located in the code.
- **Coupling:** a class A is coupled with class B if A effects B and vice versa.

INTRODUCTION

Feature envy is a major type of Technical Debt (TD) and one of the most prevalent code smells. When developers choose short-term fixes over better, long-term solutions, the result is TD, which is basically that extra work that accumulates later. Feature Envy causes a method in one class to become overly dependent on another, which weakens the bonds within that class and strengthens the coupling between them. Code becomes more rigid and difficult to maintain when classes are closely coupled because modifications to one have an immediate impact on the other. Because it keeps the code more flexible and manageable, it's always a good idea to keep coupling low. The best way to address Feature Envy is to simply move that method to the class on which it depends too much. This eliminates superfluous dependencies, improves the class design, and cleans up the code.

OPEN INVESTIGATION

Existing Studies:

- Identified that **feature envy** is present in all long methods and god classes, which are well known code smells.
    - Long Method**-** A Method that does too many tasks and is excessively long.
    - God Class **-** A Class that takes on too many responsibility.

  Hence, this points out a major problem that needs to be detected, and gives clues to develop processes to refactor it.

- Existing approaches fail to exploit the fundamental method call relationships.
    - They manually designed expressions to convert the method invocation information into distances, which results in bad detection performance.
    - The input format used was not good enough to identify related samples, for example:
        - Method - Class relationships could help in identifying related samples.

  Hence, this leads to poor refactoring capability.

OPEN INVESTIGATION

Limitations:

- No real understanding of method behavior – Just counting method calls isn’t enough.
- No automated fix – Tools tell you there’s an issue but doesn’t move the method for you.
- Poor generalization – A model trained on one codebase doesn’t always work well on another.
- Method invocation relationship is not given much significance in detecting Feature Envy, which can be very important for a project.

PROBLEM STATEMENT & THE PROPOSED SOLUTION

Problem Statement: How to detect and refactor Feature Envy by improving upon existing methods?

The **proposed solution** focuses on method-class relationships and introduces 2 processes:

- **SMOTE Call Graph (SCG)**: It is a GNN based approach to address the problems of detecting and refactoring feature envy.
    - SCG formulates the detection task as a node binary classification problem on the method call graph:
        - Calling Strength as a concept is introduced to quantify the strength of method invocations in call graph.
        - It builds upon the existing studies by converting the method call graph to a method-class call graph representation.
        - Recommends moving the smelly method (feature envy method) to the external class with the highest calling strength.

PROBLEM STATEMENT & THE PROPOSED SOLUTION

- **Symmetric Feature Fusion Learning (SFFL)**: addresses the feature envy refactoring problem.
    - Collects the following details from a project:
        - Invocation: indicates which method calls which one
        - Ownership: indicates which method belongs to which class
        - Position: where the methods is located in the code
        - Semantic information
    - Encodes them into four directed heterogeneous graphs, where the:
        - Nodes are methods and classes
        - Edges are invocation or ownership relationship between them.
    - Hence, SFFL introduces a link prediction to generate adjacency matrix representing the ownership relationships between methods and classes.

Why are two methods used?:

- both the methods complement each other.
- SFFL exhibits better detection and refactoring performance than SCG.
- The detection performance of SCG is superior to SFFL when the training samples are extremely imbalanced.

WHAT DO WE AIM TO ACCOMPLISH?

We aim to duplicate the paper's results by the end of this project.