JDeodorant: Identification and Removal of Feature Envy Bad Smells

Marios Fokaefs, Nikolaos Tsantalis, Alexander Chatzigeorgiou
Department of Applied Informatics, University of Macedonia
54006 Thessaloniki, Greece
{marios, nikos}@java.uom.gr, achat@uom.gr

Abstract

In this demonstration we present an Eclipse plug-in that identifies Feature Envy bad smells in Java projects and resolves them by applying the appropriate Move Method refactorings. The main contribution is the ability to pre-evaluate the impact of all possible Move refactorings on design quality and apply the most effective one.

1. Introduction

Placement of attributes/methods within classes in an object-oriented system is one of the most important analysis/design activities. However, this process is heavily dependent on the human factor and thus might lead to non-optimized systems in terms of design quality. Moving state and behavior between classes [3] can help to reduce coupling and increase cohesion, but it is non-trivial to manually identify where such refactorings should be applied. The identification of refactoring opportunities is one of the essential steps in the refactoring process [4].

To this end the proposed tool: a) automatically identifies problems related to the misplacement of methods (Feature Envy bad smells) in Java programs, b) ranks the refactorings that resolve the identified problems according to their impact on the design and c) automatically applies the most effective Move refactoring.

2. Underlying methodology

The identification of Feature Envy bad smells is based on the notion of distance between methods and system classes. The distance between a method *m* and a class *C*, expresses the dissimilarity between the set of entities (methods and attributes) accessed by *m* and the set of entities belonging to *C*. A Feature Envy bad smell is identified (and a corresponding Move Method refactoring candidate is extracted) if the distance of a method from a system class is less than the distance of this method from the class that it belongs to.

In a well designed system, the distances of the entities belonging to a class (inner entities) from the class itself, should be the smallest possible (high cohesion). At the same time the distances of the entities not belonging to a class (outer entities) from that class, should be as large as possible (low coupling). For each class, the ratio of average inner to average outer entity distances is a measure of how well entities have been placed in a class. The closer this ratio to zero is, the safer it can be concluded that inner entities have correctly been placed inside the class and outer entities to other classes. A system-level design quality measure (termed *Entity Placement*) is obtained by extracting the average for all classes. This measure can serve as a criterion for selecting the most effective solution among several suggested refactorings. One of the advantages of the proposed approach is that the effect of each candidate refactoring is calculated without actually applying it. Rather, refactorings are evaluated "virtually" meaning that only the corresponding entity sets are updated without altering the source code.

3. Tool overview

The tool employs the ASTParser of Eclipse Java Development Tools (JDT) to analyze the relationships between system entities and apply move refactorings on source code. To the best of our knowledge there is no API in Eclipse to perform Move Method refactorings in Java projects. The tool can be downloaded from [1].

3.1. Identification of Feature Envy bad smells

The user imports the system under study as a Java Project and opens Navigator View in Java Perspective.

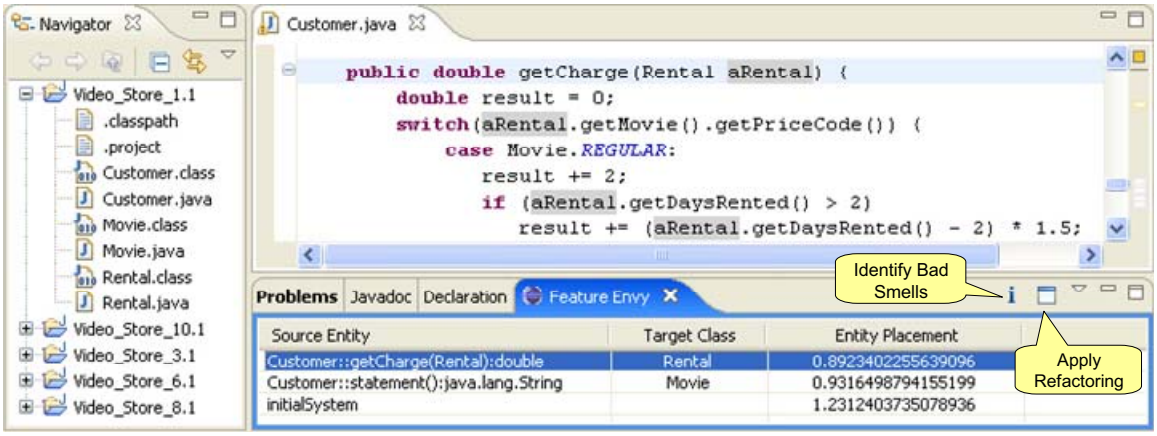

Figure 1. JDeodorant output showing identified Feature Envy bad smells

Then, the user selects the *Bad Smells* item in the menu bar and triggers the *Feature Envy* action, which in turn opens the corresponding view. After pressing the *Identify Bad Smells* button the *Feature Envy* view lists both the identified bad smells as well as the candidate refactorings that resolve them, as shown in Figure 1. The view contains three columns: a) the *Source Entity* column, showing misplaced methods (and the class to which they currently belong) representing an identified Feature Envy bad smell, b) the *Target Class* column, showing the class to which the *Source Entity* should be moved, thus representing a candidate Move Method refactoring, c) the *Entity Placement* column, showing the Entity Placement value resulting from the "virtual" application of the corresponding candidate refactoring.

3.2. Ranking of Move refactorings

The candidate Move refactorings are presented in ascending order according to the value of the *Entity Placement* column, in order to clearly highlight the most effective one. Moreover, the *Feature Envy* view includes an entry `initialSystem`, which corresponds to the *Entity Placement* value for the initial system. This helps to distinguish the candidate refactorings leading to an improvement of the design from those possibly causing deterioration.

3.3. Application of Move Method refactorings

In the *Feature Envy* view the user selects the entry corresponding to the Move Method refactoring he/she wishes to perform. The methodology suggests the selection of the refactoring with the lowest Entity Placement value; however, the user is free to select any of the candidate refactorings. The user should press the *Apply Refactoring* button (Figure 1) to actually perform the selected refactoring on source code.

4. Evaluation

The demonstrated tool has been applied on two widely known refactoring examples, namely Video Store [3] and LAN-simulation [2]. We have generated versions of both systems exactly before the application of the Move Method refactorings proposed by the authors. These versions are available at [1]. To check the validity of the tool we compared the refactorings proposed by the authors at each version to those automatically suggested. In the Video Store example, six out of six cases of Feature Envy bad smells have been successfully identified. In the LAN-simulation example, seven out of eight cases of Feature Envy bad smells have been successfully identified (the single missed case is due to the authors' choice to move a method based on conceptual rather than data access criteria).

5. References

[1] Bad Smell Identification for Software Refactoring, http://java.uom.gr/~nikos/bad-smell-identification.html, 2007

[2] S. Demeyer, F. Van Rysselberghe, T. Girba, J. Ratzinger, R. Marinescu, T. Mens, B. Du Bois, D. Janssens, S. Ducasse, M. Lanza, M. Rieger, H. Gall and M. El-Ramly, "The LAN-simulation: A Refactoring Teaching Example", *8th Int. Workshop on Principles of Software Evolution* (IWPSE'05), Lisbon, Portugal, pp. 123-134, September 5-6, 2005.

[3] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, Boston, MA, 1999.

[4] T. Mens, T. Tourwé, "A Survey of Software Refactoring", *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126-139, February 2004.