JDeodorant: Identification and Application of Extract Class Refactorings

Marios Fokaefs
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
fokaefs@ualberta.ca

Nikolaos Tsantalis
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
nikos@java.uom.gr

Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
stroulia@ualberta.ca

Alexander Chatzigeorgiou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
achat@uom.gr

ABSTRACT

Evolutionary changes in object-oriented systems can result in large, complex classes, known as "God Classes". In this paper, we present a tool, developed as part of the JDeodorant Eclipse plugin, that can recognize opportunities for extracting cohesive classes from "God Classes" and automatically apply the refactoring chosen by the developer.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring, reverse engineering, and reengineering*

General Terms

Design

Keywords

refactoring, software reengineering, object-oriented programming, clustering

1. INTRODUCTION

Software is being developed throughout its life cycle and may constantly change even well after its release. Changes usually include bugs being corrected, new features being added, the code base being modified and extended, the design being adjusted, the development team and its practices evolving. This often results in complex, unwieldy, inelegant, difficult to understand and maintain modules, which in the case of object-oriented software manifest themselves as large and non-cohesive "God Classes" [4].

The problem of "God Classes" can be addressed by several refactorings, all of which aim to extract some of their data and functionality into other classes. The extracted elements can be redistributed to the collaborators of the "God class" (through the "Move Method" and "Move Attribute" refactorings) or repackaged as separate new classes (through the "Extract Class" refactoring). In principle, the latter solution is preferable, since it is less likely to reduce the cohesion of the recipient classes.

The tasks of identifying problematic "God classes" and choosing specific refactorings to improve them are not trivial and require substantial experience and effort by the developers. Even when an opportunity for the "Extract Class" refactoring has been spotted, its actual application to the code base is not trivial considering the need of ensuring (a) the syntactic correctness of the codebase and (b) the preservation of the observable behavior of the system. For example, in some cases the dependencies between class members span across the entirety of the system or, in some other cases, the refactoring cannot be performed because of inheritance or synchronization violations. State-of-the-art IDEs sometimes fail to address sufficiently these issues. For example, Eclipse 3.6 allows the user to extract attributes in a new class and then perform a sequence of the "Move Method" refactorings, to add behavior to this new class. This adds to the required human effort to apply the refactoring. It would therefore be desirable to build in IDEs or expand their existing toolkits with the ability to suggest and carry out properly "Extract Class" refactorings. Our tool automatically applies "Extract Class" refactorings making all the necessary changes for the system's behavior and its syntactic correctness to be preserved.

The development of our plugin relies on two novel contributions. First we developed a clustering method that examines the entities (i.e., attributes and methods) in all the system classes to suggest refactorings that would potentially improve the overall system design. The system-design quality is measured by the *Entity Placement* metric [5], com-

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA
ACM 978-1-4503-0445-0/11/05

puted as the ratio of cohesion over coupling. The second contribution of this work is the automation of the "Extract Class" refactoring application. The user can preview the changes using the Eclipse Preview Wizard. The tool is developed as part of the JDeodorant [1] Eclipse plugin. The tool is available at the JDeodorant website[1]. A video of the tool is also available on Youtube[2] .

2. THE REFACTORING PROCESS

2.1 Identifying Extract-Class Opportunities

The tool identifies extract-class opportunities by applying a hierarchical agglomerative clustering algorithm on a single class using the Jaccard distance as the distance metric. The distance between the attributes and methods of the class was calculated by comparing the similarity of their *entity sets*. The entity set of an entity (attribute or method) contains all the members of the class that use or are used by the entity in question.

Given a class, the algorithm starts by computing the entity sets of each class entity, i.e., each attribute and method, and placing each individual entity in a separate cluster. Then, at each step, the algorithm chooses to merge the two clusters (and their entity sets) that are closest to each other according to the single-linkage criterion. The process stops, when no more clusters can be merged since they are all more distant to each other than the distance threshold. In our experiment, we ran the algorithm for different values of the threshold ranging from 0.1 to 0.9 with a 0.1 step because a single fixed threshold is not sufficient to obtain all possible clusters that are produced by the hierarchical clustering algorithm.

2.2 Ranking the Extract-Class Opportunities

The identified refactoring opportunities are virtually performed in order to calculate the resulting Entity Placement metric value without causing changes on source code. This is achieved by changing the entity sets accordingly and recalculating the distances. To improve the performance of our tool, we only recalculate the distances between the changed entities (classes, methods and attributes) from the rest.

The resulting suggestions are then grouped by the source class and presented in a tree-table format - see Figure 1, point 5. The rows of the table correspond to candidate refactorings. The columns of the table correspond to the source class and the resulting entity-placement metric for the system if the suggested refactoring is applied. The suggestions are organized in groups which contain all the candidate refactorings suggested for a single source class. Each group is assigned the lowest Entity Placement value among its contained refactoring candidates. Eventually, the groups are sorted in ascending order according to their assigned Entity Placement value. The suggested refactorings that produce a worse value than the one of the current system are discarded. More details about the identification process can be found in [2].

2.3 Applying the Chosen Refactoring

Refactoring, as defined by Fowler [3], is a change, which alters the internal structure of the system's code base but does not affect its observable behavior. To ensure that we fulfill the latter constraint, we have defined a set of preconditions [2] that ensure - up to a certain degree - the stability of the system's functionality. At the same time, we have to make sure that the changes effectuated by the refactoring do not introduce any syntactic errors. To apply the refactoring we used the ASTRewrite of Eclipse's Java Development Toolkit (JDT). To preview the changes we used the Preview Wizard in Eclipse Language Toolkit (LTK).

The application of the Extract Class refactoring involves the following steps. First, the algorithm removes the extracted entities from the source class and adds the new class in the same package as the source class. Next, the bodies of the extracted methods and the types of the extracted fields are inspected and the algorithm adds the required import declarations in the new class. Next, the extracted fields are added in the new class as private attributes with public accessors, in order to preserve the encapsulation principle. Before adding the extracted methods to the new class, a few issues must be considered. First, if the extracted method assigns a field or invokes a method of the source class, it is likely that it may change the state of source class instances. Therefore, the source class must be passed as a parameter to the new method when it is added to the newly extracted class, so that the same change is feasible. On the other hand, if an attribute of the source class is only read, it suffices to add a parameter of the type of the accessed attribute in the new method; in this way, the method does not unnecessarily increase the coupling between the source and the extracted class. Because these changes may alter the method's signature, the algorithm has to modify the invocations of this method in the rest of the extracted methods. As the final step, the newly modified methods are added in the newly extracted class. Finally, the source class is modified. First, the algorithm checks if the extracted methods are also invoked by a third class (other than the source or the new class). If this is true, the original source method is turned into a method that delegates to the extracted one, so that its public interface does not change. Then, a field having the type of the newly created class is added in the source class, it is instantiated, and the accesses of any members of the new class are appropriately modified in the source class. For example, if a method's signature is changed, its invocations in the source class need to be modified as well.

3. EXPERIMENTAL RESULTS

The effectiveness of the tool has been evaluated on the JHotDraw system (version 5.3). We applied 16 of the suggested refactorings and asked a professional in the business of software quality assessment, to give his expert opinion and answer three questions for each refactoring:

Q1. Does the extracted class describe a new concept?

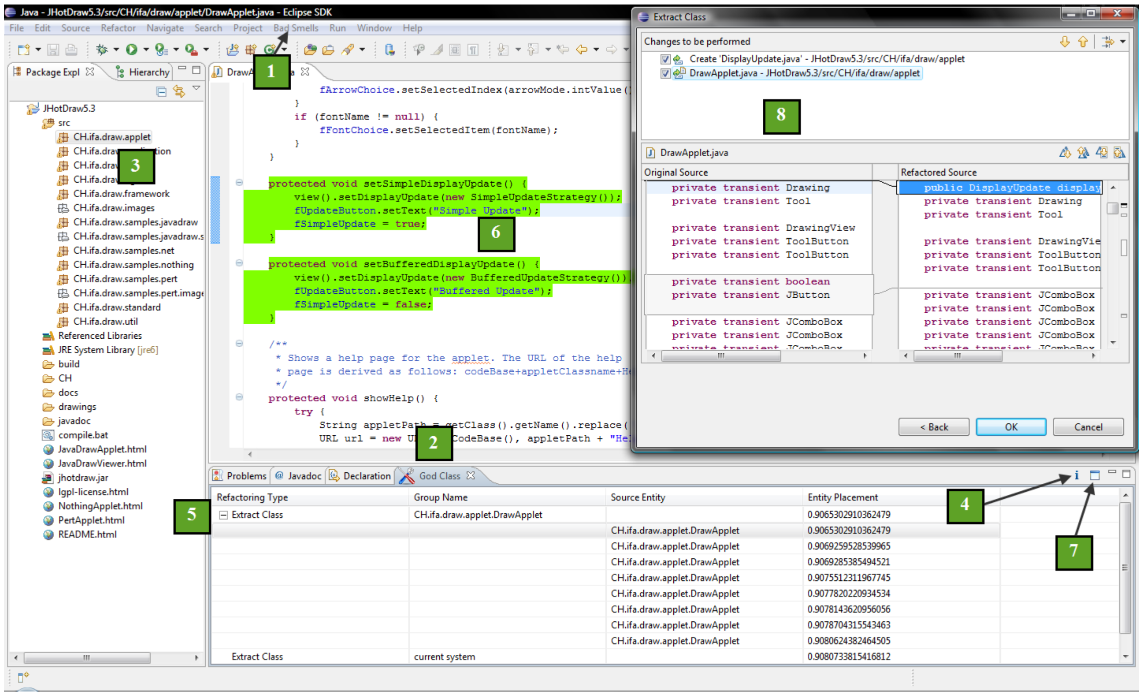
Q2. Would you actually perform this refactoring, if a tool suggested it?

Q3. Does it improve the understandability of the code?

In 12 cases (75%), the evaluator confirmed that the classes suggested to be extracted indeed described a separate concept. According to the expert two of these classes could be extracted and used as utility or helper classes. Although, they do not actually describe a new concept they can still be extracted as new classes. In 9 cases (56%), the expert agreed that he would perform the refactoring if it was suggested by a tool. Interestingly, in 3 cases he claimed that he would

[1]http://www.jdeodorant.com

[2]http://www.youtube.com/watch?v=h8K2M-lbDYo



```
            fArrowChoice.setSelectedIndex(arrowMode.intValue(
        }
        if (fontName != null) {
            fFontChoice.setSelectedItem(fontName);
        }
    }

    protected void setSimpleDisplayUpdate() {
        view().setDisplayUpdate(new SimpleUpdateStrategy());
        fUpdateButton.setText("Simple Update");
        fSimpleUpdate = true;
    }

    protected void setBufferedDisplayUpdate() {
        view().setDisplayUpdate(new BufferedUpdateStrategy());
        fUpdateButton.setText("Buffered Update");
        fSimpleUpdate = false;
    }

    /**
     * Shows a help page for the applet. The URL of the help
     * page is derived as follows: codeBase+appletClassname+He
     */
    protected void showHelp() {
        try {
            String appletPath = getClass().getName().replace(
            URL url = new U   CodeBase(), appletPath + "He
```

Changes to be performed
- ☑ Create 'DisplayUpdate.java' - JHotDraw5.3/src/CH/ifa/draw/applet
- ☑ DrawApplet.java - JHotDraw5.3/src/CH/ifa/draw/applet

DrawApplet.java

Original Source

```
    private transient Drawing
    private transient Tool

    private transient DrawingView
    private transient ToolButton
    private transient ToolButton

    private transient boolean
    private transient JButton

    private transient JComboBox
    private transient JComboBox
```

Refactored Source

```
    public DisplayUpdate display
    private transient Drawing
    private transient Tool

    private transient DrawingVie
    private transient ToolButton
    private transient ToolButton

    private transient JComboBox
    private transient JComboBox
    private transient JComboBox
    private transient JComboBox
```

Refactoring Type	Group Name	Source Entity	Entity Placement
⊟ Extract Class	CH.ifa.draw.applet.DrawApplet		0.9065302910362479
		CH.ifa.draw.applet.DrawApplet	0.9065302910362479
		CH.ifa.draw.applet.DrawApplet	0.9069259528539965
		CH.ifa.draw.applet.DrawApplet	0.9069285385494521
		CH.ifa.draw.applet.DrawApplet	0.9075512311967745
		CH.ifa.draw.applet.DrawApplet	0.9077820220934534
		CH.ifa.draw.applet.DrawApplet	0.9078143620956056
		CH.ifa.draw.applet.DrawApplet	0.9078704315543463
		CH.ifa.draw.applet.DrawApplet	0.9080624382464505
Extract Class	current system		0.9080733815416812

Figure 1: JDeodorant in action. 1) The "Bad Smells" menu 2) The "God Class" view 3) The Project Explorer 4) The identification button 5) The expand icon and the tree-table format 6) The highlighted editor 7) The apply button 8) The preview wizard

not have been able to identify the refactoring opportunities by himself. In 9 cases, the expert notes that the performed refactorings have a positive impact on the understandability of the system.

During the evaluation, we discovered a few by-products of the tool. In some cases, the code suggested to be extracted was duplicated in the same or other classes. Another approach would be to extract one of them in a new class and replace both instances of the duplicated code with references to the extracted class. This type of duplication detection is outside the scope of this tool at this point, and we are considering it as a potential extension. The proposed changes can be manipulated further by the user to achieve better results. In another interesting case, the extracted code was totally disconnected from the rest of the class and was not used anywhere else, indicating possibly dead code that needs to be removed.

4. CONCLUSION

In this paper, we presented an extension for the JDeodorant Eclipse plugin. The tool employs a clustering technique to identify extract-class refactoring opportunities and allows the user to automatically perform the suggested refactorings ensuring the syntactical correctness of the code and preserving its behavior. We evaluated the tool on the JHotDraw system and asked for the expert assessment of a professional evaluator on the effect of the applied refactorings as these were suggested by the tool. The results of the evaluation confirmed our method's ability to identify new concepts. Furthermore, the expert confirmed that a good percentage of the proposed refactorings were good solutions that also improved the understandability of the code.

In the future, we would like to explore the possibility of combining our identification method with others, like code duplication detection techniques, to improve our results. We also plan to improve the interface of the tool with visualizations so that the user can better understand the results of the identification process.

5. ACKNOWLEDGMENTS

The authors would like to thank Eric Bouwers and Yiannis Kanellopoulos for their participation in the evaluation process. This work was supported by NSERC, iCORE and IBM.

6. REFERENCES

[1] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou. JDeodorant: Identification and Removal of Feature Envy Bad Smells. *23rd International Conference on Software Maintenance*, pages 519–520, 2007.

[2] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander. Decomposing Object-Oriented Class Modules Using an Agglomerative Clustering Technique. *25th IEEE International Conference on Software Maintenance*, pages 93–101, 2009.

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

[4] A. J. Riel. *Object-Oriented Design Heuristics*. Addison Wesley, Boston, MA, 1996.

[5] N. Tsantalis and A. Chatzigeorgiou. Identification of Move Method Refactoring Opportunities. *IEEE Transactions on Software Engineering*, 35(3):347–367, 2009.