An Interactive Visualization of Refactorings Retrieved from Software Archives

Peter Weißgerber, Stephan Diehl
Computer Science Department
Catholic University Eichstätt
85072 Eichstätt, Germany
pweissgerber@ku-eichstaett.de,
diehl@acm.org

Carsten Görg
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
goerg@cc.gatech.edu

ABSTRACT

We perform knowledge discovery in software archives in order to detect refactorings on the level of classes and methods. Our REFVIS prototype finds these refactorings in CVS repositories and relates them to transactions. Additionally, REFVIS relates movements of methods to the class inheritance hierarchy of the analyzed project.

REFVIS creates visualizations that show these refactorings in two different layouts and uses color-coding to distinguish between different kinds of refactorings. Moreover, our visualizations are interactive as they can be zoomed, scrolled and filtered; mouse-over-tooltips allow to examine details of the particular refactoring on demand.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Miscellaneous;
K.6.3 [**Software Management**]: *Software maintenance*

Keywords

CVS, Data Extraction, Refactorings, Visualization

1. INTRODUCTION

The application of refactoring is a standard task of programmers and many integrated development environments support (semi-) automated refactoring. Refactoring aims at improving the internal structure of software without changing its external behavior.

There are several reasons to identify refactorings performed in a software project: Knowing the refactorings helps developers who have been absent to refamiliarize themselves with the code, or library users to adapt their program to newer library versions [3]. Furthermore, one can identify incompletely performed refactorings [2] to prevent errors. Additionally, one can compare quality metrics of the software before and after a refactoring.

In this paper we present a technique to extract and visualize the following refactorings based on the CVS repository:

Structural Refactorings. Move Class, Move Method, Pull Up Method, Push Down Method.

Local Refactorings. Hide Method, Rename Method, Add Parameter, Remove Parameter.

REFVIS consists of two components: The REFVIS extractor, which is described in Section 2, extracts refactorings from a CVS repository and stores them in a data file. This file is the input of the REFVIS visualizer which creates interactive visualizations of these refactorings and is described in Section 3.

2. EXTRACTING REFACTORINGS

The extraction of the refactorings from the CVS repository is done in two steps: First, we preprocess the data from the CVS repository. After that, we analyze the preprocessed data for refactorings that have been performed in the software history.

This paper provides a brief description of the extraction. The preprocessing is described in further detail in [4] and the analysis in [1].

Preprocessing the Repository

Unfortunately, direct access of the data is quite slow. Furthermore, some information must be recovered from different places within the repository. Thus, the first step of our technique is to extract the repository completely, recover information when necessary, and store this data in a relational database. After the extraction we can access the following information:

Versions. A *version* describes one revision of a file in the CVS repository (e.g. file `org/epos/epos.java` in revision 1.4).

Transactions. A *transaction* is the set of versions that have been committed to the repository by a developer using one commit operation.

Analyzing for Refactorings

To gather information about which syntactical entities (classes and methods) are contained in a JAVA file (and thus, may

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

be affected by refactorings) we use a lightweight parser that identifies a) classes in versions and b) methods in classes.

To obtain refactorings that have been performed in a transaction we compare the syntactical entities of the versions in the transaction with the entities in the predecessors of these versions. Currently, we use only simple heuristics to decide whether a change is a refactoring or not.

To decide if a moved method has actually been pushed down or pulled up, we construct the inheritance tree of the examined JAVA project after the transaction that contains the move refactoring.

3. VISUALIZING REFACTORINGS

The REFVIS visualizer takes the refactorings obtained by the extraction as input, uses graph drawing methods to generate a visualization and stores this visualization as a Scalable Vector Graphics (SVG) file that includes a Javascript for interactive operations.

The REFVIS Visualization

REFVIS visualizes refactorings on the level of classes and methods, but currently not on the level of source code. Classes and interfaces are represented by boxes containing the fully-qualified class name and a list of the signatures of the methods in this class/interface. REFVIS provides two different views on the classes after a transaction: the class hierarchy view which relates the classes using UML-style arrows to the class hierarchy (see Figure 1), and the package view which is shown in Figure 2.

The different types of relations are color-coded: extend edges are red, implement edges blue, and aggregate edges black. Color-coding is also used to distinguish the different kinds of refactorings:

Move Refactorings are displayed in red tones (move class: red, pull up method: orange, push down method: dark orange),

Changed Parameters are displayed in green tones (add parameter: light green, remove parameter: dark green),

Hide Method Refactorings are displayed in gray,

Rename Method Refactorings are displayed in magenta.

Mouse-over-tooltips (as in Figure 1) show a short description of all refactorings in a class.

Filtering and Context

In most cases developers are not interested in all classes but only in those they are working on or that are contained in particular transactions. Thus, REFVIS provides a set of filters:

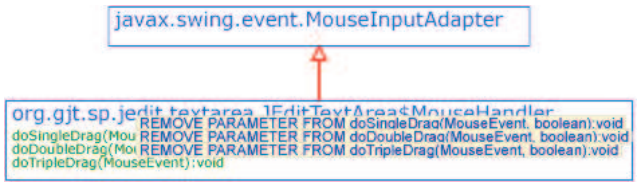

Figure 1: Hierarchical Layout of a Refactoring.

Figure 2: Package Layout of a Refactoring.

transaction filter: consider only transactions with (particular) refactorings from a certain period of time,

class filter: consider only classes that contain (particular) refactorings and/or are contained in particular packages and/or contain particular methods,

method filter: consider only methods involved in (particular) refactorings.

It is also important to build the appropriate context of refactorings. The visualization of *Push Down Method* and *Pull Up Method* refactorings, for example, should not only show the target class of the refactoring, but the source class as well. Thus, displaying sub- and superclasses, as well as aggregated and aggregating classes can be enabled and disabled.

Interactive Views

REFVIS stores the created visualization as an SVG file that includes a script that offers free zooming and scrolling features. These provide the user with the possibility to get an overview of all classes affected by refactoring (by zoom out), as well as a closer look at a certain class (by zoom in). The SVG file can be viewed with any web browser with an SVG plugin.

4. REFERENCES

[1] C. Görg and P. Weißgerber. Detecting and Visualizing Refactorings from Software Archives. In *Proc. 13th International Workshop on Program Comprehension (IWPC 2005)*, St. Louis, Missouri, U.S., 2005.

[2] C. Görg and P. Weißgerber. Error Detection by Refactoring Reconstruction. In *Proc. 2nd International Workshop on Mining Software Repositories (MSR 2005)*, St. Louis, Missouri, U.S., 2005.

[3] J. Henkel and A. Diwan. CatchUp!: Capturing and Replaying Refactorings to Support API Evolution. In *Proc. 27th International Conference on Software Engineering (ICSE 2005)*, St. Louis, Missouri, U.S., 2005.

[4] T. Zimmermann and P. Weißgerber. Preprocessing CVS Data for Fine-Grained Analysis. In *Proc. International Workshop on Mining Software Repositories (MSR 2004)*, Edinburgh, Scotland, U.K., 2004.