Refactoring Support for the Ruby Development Tools

Thomas Corbat, Lukas Felber, Mirko Stocker

Abstract: We present our refactoring plug-ins for Eclipse's Ruby Development Tools IDE. Refactoring is a very important technique for every software engineer and a cornerstone of agile software development. In a term project and diploma thesis, we have implemented several automated refactorings for example Rename Variable and Extract Method.

1 Introduction

Refactoring is a very useful concept for every software engineer. Martin Fowler, the author of the prominent book Refactoring [Fow99], says:

> "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure."

During software development, the programmer often has to modify the existing code to make it more robust and less error-prone. While this does not change the functionality of the product, it certainly improves the maintainability, understandability and testability. This process is called refactoring. Doing refactoring by hand is often quite tedious and generally engineers are afraid of changing working code. A more recent and very popular automated refactoring implementation can be seen in Eclipse's JDT. These tools offer many automated refactorings which assist the programmer and make his life easier.

Examples of common used refactorings are:

- Renaming of classes, methods and variables.
- Moving code, for example methods or variables.
- Extracting parts of code into other methods or even classes.

The Ruby Development Tools are an IDE for Ruby based on the Eclipse platform. Although they are having a lot of features they didn't support automated refactorings. The goal of this project had been to implement refactoring support for the Ruby Development Tools.

In Ruby the main difficulty is also one of its greatest language features: dynamic typing. Dynamic typed languages offer a lot of freedom to the programmer, however, it is

hard and sometimes even impossible for an IDE to figure out the type of an object. Thus refactorings with a large scope, like the renaming of public methods, are a real challenge.

2 Implemented Refactorings

During a term project and a diploma thesis, we implemented the following refactorings:

- Override Method
- Extract Method and Class
- Rename Class, Method, Field and Local Variables
- Inline Class, Method and Temp
- Convert Local Variable to Field
- Split Temporary Variable
- Move Method and Field
- Generate Accessors, Contructor
- Merge Class Parts

We tried to follow the list of refactorings from the JDT and adapted them for the Ruby language.

2.1 Example: Extract Method

Extract Method removes a block of instructions out of an existing method and creates a new one. The new method will be called from the existing method where the instructions had been removed. The local variables from the existing method that are used in the affected code block are passed to the new method as parameters. If any of those local variables are set inside the selected block they will be returned from the new method as return values. Fortunately, Ruby supports multiple return values and multiple assignment of variables. An example of the Extract Method refactoring is shown in figure 1.

3 Results

The plug-ins are now integrated into the official RDT repository and will be part of the next release. We also plan to maintain and extend the plug-ins to keep up with the latest

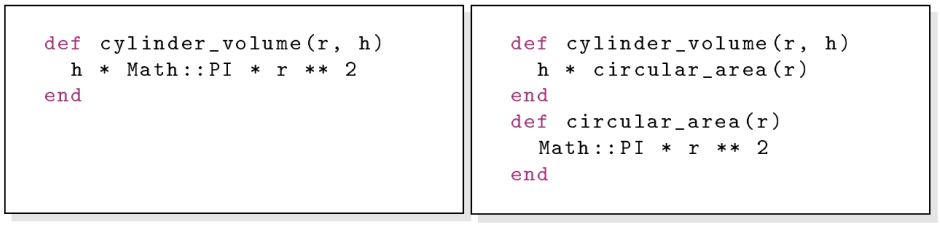

Figure 1: Extract Method Refactoring

developments in JRuby and RDT. A possible extension of our refactorings could be the integration into RadRails, so you could rename your controllers or views and the file names were automatically changed too.
The term project and the diploma thesis were rewarded with the grade six, which is the maximum in Switzerland. We even won a price for one of the best diploma theses in our year.

4 Procedure

The project was done in a fourteen-week long term project and an additional eight-week diploma thesis. During the term project we worked about 20 hours per week and 45 during the diploma thesis. A lot of the time from the term project was spent on foundation work, like understanding the existing code base, bug fixing and implementation of functionality we needed for our main tasks, the refactorings. A lot of this time wasn't planned and messed up our initial time plan, so we had to adjust it a lot over time. The diploma thesis in contrary went quite different, thanks to our previous knowledge. We created a milestone for each week and a task for each refactoring. During the weekly meetings with our supervisor, we reviewed our progress. rescheduled the remaining tasks and set up a goal for the next week.
We used a very pragmatic process. Before each new refactoring task we spent some time researching the features and special cases of the Ruby language and started writing tests and code immediately. If we came to new knowledge we refactored and built up the architecture of the plug-ins this way. Up front design wouldn't have worked for us, we simply hadn't the experience and knowledge about the domain and we believe the results speak for themselves.

References

[Fow99] Martin Fowler. *Refactoring*. Addison-Wesley Professional, 1999.