Refactoring Support for the C++ Development Tooling

Emanuel Graf Guido Zgraggen Peter Sommerlad

IFS Institute for Software at HSR Rapperswil
Oberseestr. 10, CH-8640 Rapperswil, Switzerland
{emanuel.graf,guido.zgraggen,peter.sommerlad}@hsr.ch

Abstract

This article reveals our work on refactoring plug-ins for Eclipse's C++ Development Tooling (CDT).

With CDT a reliable open source IDE exists for C/C++ developers. Unfortunately it has been lacking of overarching refactoring support. There used to be just one single refactoring - Rename. But our plug-in provides several new refactorings which support a C++ developer in his everyday work.

Categories and Subject Descriptors D [*2*]: 7—Restructuring, reverse engineering, and reengineering

General Terms Design, Languages

Keywords Refactoring, C++, Eclipse

1. Introduction

A C++ programmer is often burdened with routine editing tasks when refactoring code. Wearing the refactoring hat, keeping corresponding header and implementation files consistent is very time-consuming

In the JDT[1] the Java development environment of Eclipse a lot of routine refactoring work is automated. The CDT[2] contains only the rename refactoring. This refactoring provides the functionality to change names of methods, fields, etc in a global context. No further refactorings are supported by CDT yet.

A big difficulty in automating refactorings for C++ is the immense variety of the language and particularly the handling of macros as well as templates is a challenge.

A first approach was the PhD Thesis[3] of William F. Opdyke. But it didn't result in wide spread use.

[1] `http://www.eclipse.org/jdt/`

[2] `http://www.eclipse.org/cdt/`

[3] `http://www.laputan.org/pub/papers/opdyke-thesis.pdf`

2. Implemented Refactorings

After the realisation of one term project, two diploma theses and a lot more work we finally have a plug-in to provides the following refactorings:

- **Declare Method**
  Builds the method declaration from an existing method implementation.
- **Extract Baseclass**
  Creates an abstract class from an existing class and applies its inheritance structure.
- **Extract Method**
  Creates a new method from existing code and replaces the selected code with a method invocation.
- **Extract Subclass**
  Creates a subclass for an existing class and applies the inheritance structure.
- **Hide Method**
  Changes the visibility of a method to private.
- **Implement Method**
  Builds the method definition for a method declaration.
- **Move Field / Method**
  Moves a field or a method to an other class.
- **Replace Number**
  Creates a new constant with a given name and replaces all occurence with it.
- **Separate Class**
  Moves a class into an own/new file.

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

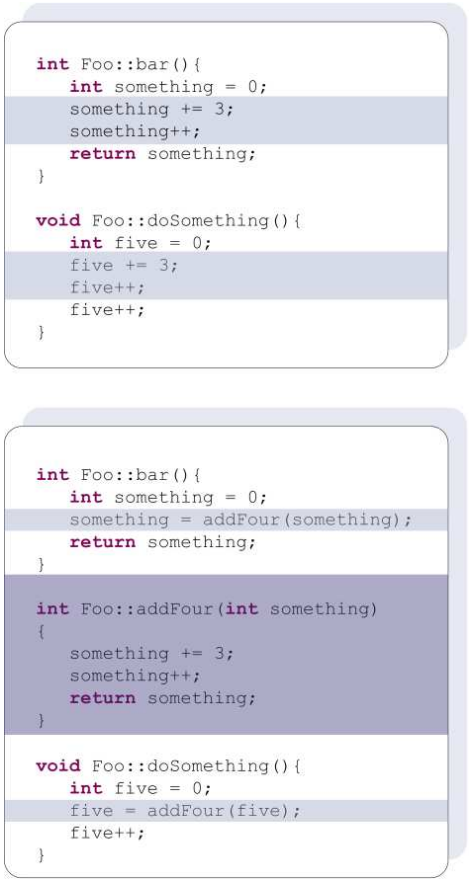

Figure 1. Extract Method Refactoring

2.1 Example: Extract Method

The key idea of extract method is to scale down the length of methods or to gather redundant code in one method. As you can see in figure 1 in the upper box there are redundancies in the code. To extract this piece of code, the code lines must be marked and the refactoring extract method must be called. After refactoring the result will look like the code in the lower box. Variables which are used in the extracted code will be passed into the method as parameter or reference. If an result of the extracted code is used in the further code, it will be given back as return value. In the case that there are more than one return value the user can choose a return value and the other parameters are passed as reference.

3. Refactoring Approach

Our refactoring implementation uses the AST of CDT and augments it with comment information. A Refactoring is then restructuring this almost concrete syntax tree from which the resulting code is generated. The Eclipse Refactoring infrastructure provides the remaining user interaction, like preview.

This simplified overview doesn't explain the C++-specific challenges we needed to conquer.

The idea for the virtual preprocessing handling was taken from the PhD Thesis[4] from Alejandro Garrido which we implemented partially. Other C++ specifics besides the preprocessor were tackled individually.

Code generation from the AST is provided through a specified writing visitor. Comment information is added to the AST through dynamic proxies. New AST proxy nodes hold comments for a particular existing AST node.

The existing CDT code can still traverse and use the augmented AST without any difference. If further processing, such as the AST writing visitor, requires comment handling, visitor behavior can be adjusted to deal with them.

4. Results

The first diploma thesis provided a plug-in called cerp[5]. The outcome of the following term project and diploma thesis was an extended CDT with refactoring capabilities. Unfortunately the hole "Refactoring CDT"[6] had to be downloaded to use these features. After some contributions to the CDT the base for an independet plug-in was founded. The two projects had been merged together and the result is a refactoring plug-in which provides all the refactoring listened above.

5. Future

We plan to commit our work into the next major CDT release, so that you don't have to download our plug-in seperately anymore.

Along the way we will also add more refactorings to expand the refactoring support further.

6. Contact

If you have wishes or recomendations please contact us at: ifs@hsr.ch or contact the head of the institute directly peter.sommerlad@hsr.ch.

[4] Program refactoring in the presence of preprocessor directives, Alejandro Garrido, 2005

[5] `http://ifs.hsr.ch/cerp`

[6] `http://wiki.hsr.ch/cdt/`