Volume 3, Issue 2, February 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Refactoring Practices

Ishu Jain and Anju Saha

USICT, GGSIPU, Dwarka

DELHI, India

Abstract— All the people either software development team or the user of software always requires quality software. Quality software is the one which is robust, reliable and easy to maintain, and which reduces the cost of software development and maintenance. Several methods have been applied to improve software quality. Refactoring is one of those methods. The goal of this paper is to elaborate in more simple way how to implement already developed different techniques of refactoring for structuring the code.

Keywords— Refactoring, Techniques of Refactoring, Testing, Structuring of code.

I. Introduction

Refactoring is the process of modifying the structure of a program while preserving all of its actual functionality. There are various ways of refactoring like renaming a class, changing the method signature, or extracting some code into a method. While using every refactoring technique, you perform a sequence of steps that keep your code consistent with the original code.If we do refactoring manually; there is a large probability of occurrence of errors into your code such as spelling mistakes etc. To remove these errors, testing should be done before and after using each refactoring technique. Refactoring is made to those programs which are poorly coded. To rectify the program from unusable or redundant code, you restructure the program to do what you want it to do [11].

Refactoring in Eclipse

Here we are performing automatic refactoring on Java projects, classes, and their members. The process of performing refactoring on any element of Java consist of following steps, firstly you must have the element(s) Selected. To select multiple elements in a view, hold down Ctrl or Shift and then select the elements. Secondly, After you have selected the element(s) you want to refactor, either choose a refactoring from the dropdown Refactor menu, or right-click and choose a refactoring from the Refactor submenu from the popup menu. In Eclipse, there are many other shortcut keys that can be used for several of the refactoring techniques. Some refactoring techniques can be applied to any type of element. Other refactoring techniques are applied only to certain types of elements for example method or a class[11].

A. Significance of Testing in refactoring

Testing is the process which we perform on every software to check the software from every perspective, whether it is giving the desired output on giving certain input or not. Testing of Java code before doing and after performing refactoring is necessary because refactoring changes the structure of your code. If the refactoring is done by hand, then a good suite of tests is a must. When using an automated tool to refactor, you should have to still test, but it is less tedious and time consuming in comparison to testing after doing refactoring manually [11].

II. Types of Refactoring in Eclipse[11]

1.1.1 *Type 1 – Physical Structure*

- Move
- Rename
- Change Method Signature
- Convert Anonymous Class to Nested
- Convert Nested Type to Top Level (Eclipse 2 only)
- Move Member Type to New File (Eclipse 3 only)

1.1.2 *Type 2 – Class Level Structure*

- Push Down
- Pull Up

- Extract Interface
- Generalize Type (Eclipse 3 only)
- User Supertype Where Possible

1.1.3 *Type 3 – Structure inside a Class*

- Inline
- Extract Method
- Extract Local Variable
- Extract Constant
- Introduce Parameter (Eclipse 3 only)
- Introduce Factory (Eclipse 3 only)
- Encapsulate Field

A. Rename

The Rename refactoring technique simply renames a Java element. It is just like performing renaming of any file in computer. The renaming of Java files and Java elements by hand, will not update any references to those files and elements. Then you search through the files in your project and replace the appropriate references manually. The Rename refactoring technique updates all of the appropriate references in a project. Sometimes the name of a Java element may not be understandable or its function may have changed. In order to maintain the readability and understandability of the code, the name of the element should be updated. Using the Rename refactoring, it is quick and easy to rename the element and update all references to it. To performing rename refactoring technique a Java element, select the element in either the Package Explorer view, the Outline view, or the Java source file and either select Rename from the Refactor menu or use the shortcut key, Alt + Shift + R. The Rename box will appear. Then You can then enter in the new name[11].

B. Move

This refactoring technique is used to move elements from one position to another position in the project. To move an element, select the element, select Move from the Refactor menu or use the shortcut key, Alt + Shift + V, and select the destination in the window that appears. Again, you can click on Preview to administer the changes, or simply press OK to carry out the refactoring without previewing [11].

C. Change Method Signature

This refactoring technique is used to make changes regarding the attributes related to Method. The Change Method Signature refactoring technique is used to change parameter names, parameter types, the parameter order, the return type, and the method's visibility. Parameters can also be added or removed to the Methods. Select the method and select Change Method Signature from the Refactor menu. The Change Method Signature dialog will appear[11].

D. Convert Anonymous Class to Nested

In this refactoring technique an anonymous class allows you to instantiate a class implementing an abstract class or interface without having to give it a name. This is often used in creating listeners for a user interface. When an anonymous class gets too large, it should be made into its own class. To do this, place the caret inside of an anonymous class and select Convert Anonymous Class to Nested from the Refactor menu. This opens the Convert Anonymous Class to Nested Class dialog where you can set the new name of the class and its visibility [11].

E. Push Down

This refactoring technique is used to push down the selected methods and fields from a class to all the classes that directly inherit it. The methods that are pushed down can optionally be left as an abstract declaration. Single methods and fields can be selected from the table. All selected elements are then pushed down to all the child classes of the current class.

The Edit button is enabled when one or more methods are selected in the table in the Push Down dialog. When the Edit button is clicked, the Edit Members dialog appears. In this dialog you can choose to leave an abstract declaration in the current class for the selected method(s) or to simply remove the method declaration(s) in the current class [11].

F. Pull up

This Pull Up refactoring technique is similar to the Push Down refactoring technique. This technique moves selected methods and fields between classes. The Pull Up refactoring technique places the methods and fields from a class to one of its superclasses. It selects one or more methods or fields and then selecting Pull Up from the Refactor menu

will open up the Pull Up dialog box. The "pull up" action will simply copy the members to the superclass and give the option to remove the members from the current class. The "declare abstract in destination" action will create an abstract method in the superclass, make the superclass abstract if it is not already, and leave the method in the current class [11].

G. Extract Interface

This refactoring technique creates an interface from an existing class. The selection of methods are done which is to be included in the interface from the current class. To open the Extract Interface dialog, select a class and then select Extract Interface from the Refactor menu. In the Extract Interface dialog, type the desired name for the interface and select the methods that you want to be in the interface. Only public methods are listed in the dialog. Click on OK to apply the refactoring technique [11].

H. Generalize

This refactoring simply modifies the type of an object in its declaration to one of its supertypes. Select a declaration of a variable, parameter, field, or method return type then click Generalize Type from the Refactor menu. In the Generalize Type dialog, select the new type of the object and click OK to apply the refactoring technique Generalize Type on the Java project [11].

I. Supertype

The 'Supertype' refactoring is used to replace references to a certain type of object by one of its supertypes. To get the Use Super Type Where Possible dialog box,select a class and then select User Supertype Where Possible from the Refactor menu. Then select the supertype to use. If the "Use the selected supertype in 'instanceof' expressions" check button is checked, it simply means that instanceof expressions will also be modified during the refactoring. Since Eclipse 3 does not have the check button so it will always modify the instanceof expressions during the refactoring [11].

J. Inline

The Inline refactoring technique takes a reference to a method, static final field, or a local variable and replaces it with its code or value. For example, if you inline a method call, the call will be replaced by the body of code in the called method. To inline a method, static final field, or a local variable, select the element and select Inline from the Refactor menu. the shortcut key Alt + Shift + I can also open the Inline dialog box. In the Inline dialog, there is a option either to inline "All invocations" or "Only the selected invocation". If the inline all invocations option is checked, then there is an option to delete the declaration itself. this refactoring technique sometimes increases the length of code but it becomes more readable [11].

K. Extract Method

This refactoring technique is used to reduce the amount of task performed by single method. If a method is performing more than one distinct operation or if some code in a method is used several times, it is better to extract a part of the method into its own method. This refactoring technique is simple to implement and makes the code easy to read. To extract a method, select the lines of code you want to extract and select Extract Method from the Refactor menu or use the shortcut key Alt + Shift + M. In the Extract Method dialog, type in the name of the new method, select the appropriate access modifier and choose whether you want thrown runtime exceptions to be added to your method signature. The "Signature preview" line on the bottom of the dialog displays what your new method's signature will look like [11].

L. Extract Local Variable

This refactoring technique makes the code simpler. The use of variable instead of an expression can have several advantages. This may accelerates the performance if the same expression is used at different locations and it can also make the code readable and understandable. To extract an expression to a local variable, select the expression and then select Extract Local Variable from the Refactor menu. This refactoring technique can also be used by using the shortcut key Alt + Shift + L [11].

M. Introduce Parameter

The Introduce Parameter refactoring technique defines new parameter in a method and then replaces an instance of a field or local variable with the new parameter. The Introduce Parameter refactoring techniques is simply used by selecting a reference to a field or a local variable in a method and then selects Introduce Parameter from the Refactor menu [11].

III. Conclusions

This paper simply describes the already existing refactoring techniques for structuring the code in a more human readable and understandable form . In this paper, we explained the importance of testing in refactoring. The objective of this paper is to assist the software engineers in understanding the ways of structuring the code.

References

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

[2] W.C. Wake, Refactoring Workbook, Addison Wesley, 2003.

[3] K. Beck, Extreme Programming Explained: Embracing Change, Addison Wesley, 1999.

[4] M. Alshayeb, W. Li, S. Graves, An empirical study of refactoring, new design, and error-fix efforts in extreme programming, in: Proceeding of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001), Orlando, Florida, 2001

.[5] T. Mens, T. Tourwe, A survey of software refactoring, IEEE Transactions on Software Engineering 30 (2004) 126–139.

[6] IEEE, Standard 1061-1992 for a Software Quality Metrics Methodology, IEEE, ed., New York, Institute of Electrical and Electronics Engineers, 1992.

[7] ISO/IEC, 9126 Standard, Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their use, Switzerland, International Organization For Standardization, 1991.

[8] R. Pressman, Software Engineering: A Practitioner's Approach, sixth ed., McGraw-Hill, 2005.

[9] F. Simon, F. Steinbrückner, C. Lewerentz, Metrics based refactoring, in: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, 2001, pp. 30–39.

[10] B.D. Bois, T. Mens, Describing the impact of refactoring on internal program quality, in: International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA), September 2003.

[11] http://www.cs.umanitoba.ca/%7Eeclipse

[12] http://www.refactoring.com

