Refactor Code : A Review

Seema Kansal
Faculty in Computer Science and Technolgy,Singhania University, Rajasthan

Abstract
Refactoring is the process of modifying a program's source code without changing its behavior, with the motive of improving program's readability. Code refactoring is a disciplined way to restructure code", undertaken in order to improve some of the nonfunctional attributes of the software. Typically, this is done by applying series of "refactorings", each of which is a (usually) tiny change in a computer program's source code that does not modify its functional requirements. This paper tells various advantages of refatoring. Also it proposes new advantage of code refactoring. By doing them in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time. This paper describes various techniques used in refactoring. The paper also proposes new techniques for refactoring.

Keywords
Refactor code, techniques of refactoring, advantages of refactoring, Refactoring.

I. Introduction

Code refactoring is a disciplined way to restructure code", undertaken in order to improve some of the nonfunctional attributes of the software. Typically, this is done by applying series of "refactorings", each of which is a (usually) tiny change in a computer program's source code that does not modify its functional requirements. By continuously improving the design of code, we make it easier and easier to work with. This is in sharp contrast to what typically happens: little refactoring and a great deal of attention paid to expediently adding new features. If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code.

By continuously improving the design of code, we make it easier and easier to work with. This is in sharp contrast to what typically happens: little refactoring and a great deal of attention paid to expediently adding new features. If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code.

Refactoring does not take place in a vacuum, but typically the refactoring process takes place in a context of adding features to software.

Refactoring and adding new functionality are two different but complementary tasks- Scott Ambler.

"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."

The developer should be confident that code refactoring is not damaging any existing functionality.

II. Advantages of Refactoring

Advantages of code refactoring include:
1. Improved code readability
2. Less complexity
3. Maintainability of the source code
4. More expressive internal architecture
5. Easy to extend the code.
6.More chance of reusability of code. We propose here this new advantage of code refactor. As we divide code into modules during refactoring these modules can be reused easily.

III. Techniques Used In Refactoring

Here are some techniques of code refactorings; some of these may only apply to certain languages or language types:

1. Encapsulate Field

It forces code to access the field with getter and setter methods. In computer programming, field encapsulation, also called data hiding, involves providing methods that can be used to read/write to/from the field rather than accessing the field directly. Sometimes these accessor methods are called getX and setX (where X is the field's name), which are also known as mutator methods. Usually the accessor methods have public visibility while the field being encapsulated is given private visibility - this allows a programmer to restrict what actions another user of the code can perform. Compare the following Java class in which the name field has not been encapsulated:

```
public class NormalClass
{
        public String name;
        public static void main(String[] args)
        {
                NormalClass example1 = new
NormalClass();
                example1.name = "myName";
                System.out.println("My name is " + example1.
name);
        }
 }
```

with the same example using encapsulation:

```
public class EncapsulatedClass
{
private String name;
  public String getName()
   {
{
return name;
   }
  public void setName(String newName)
   {
     name = newName;
   }
  public static void main(String[] args)
   {
    EncapsulatedClass example1 = new  EncapsulatedClass();
    example1.setName("myName");
    System.out.println("My name is " + example1.getName());
   }
}
```

2. Generalize Type

It creates more general types to allow for more code sharing.

Type generalization is a technique commonly used in refactoring. The idea is to draw on the benefits of object-orientation and make more-generalized types, thus enabling more code sharing, leading to better maintainability as there is less code to write. Too-general code can, however, become completely useless, leading to spaghetti code doing effectively nothing.

Type generalization refers to making more general or more abstract some subset of the traits of a specific type. A superclass has wider use than a specific subclass, and so is more 'general'.
An example of generalizing a type would be moving a method from a child to a parent class for common use by all the parent class' children, not just the original child.

Another example, in the Java programming language, would be access to an object via an interface which isn't tied in to a specific implementation of that interface.

3. Extract Method
It turns part of a larger method into a new method. By breaking down code in smaller pieces, it is more easily understandable. This is also applicable to functions. Extract moves part of the code from an existing class into a new class.
You have one class doing work that should be done by two.

4. Rename method or Rename Field
Changing the name into a new one that better reveals its purpose. One of the simplest refactoring methods is called Rename method. There are many reasons why one may want to change the name of some method. I think that main reason to rename a method is to give it a name that describes better what method is supposed to do.

Let's see some code.
```
public class ProfileImporter
{
   private List<Profile> _profiles;

   public void Import()
   {
      foreach (var profile in _profiles)
         ImportOne(profile);
   }
   public void ImportOne(Profile profile)
   {
      // import profile data
   }
}
```
We can easily understand what this class does. It imports user profiles. Import method iterates through the list of previously loaded user profiles and imports them one by one. ImportOne method imports given profile and returns. Everything seems to be perfect but there is something annoying.

When we are looking at this class we may not notice that ImportOne is not informative name. But when we are using object based on that class then ImportOne doesn't seem nice name. What it means "import one"? One what? How one is imported? It doesn't make me feel comfortable when I have to call this method because I'm not sure what it exactly does.

Now I want to give better name to this method so I can be sure what it does and also other team members or customers can understand what this method exactly does. New name of the method will be ImportProfile. It is clear to everybody what this method does – it imports one profile. And as we can see from argument list we have to provide the profile we want to import.

5. Adding new functions
To make code more readable it should be divided into small modules. We can do it by converting a general code of section into a function.

6. Replacing the macro definition codes with proper aspect program and advice
The macros cause a lot of problems in refactoring as the refactoring tools cannot be applied to them as we had discussed in the earlier sections. Thus we are looking for appropriate replacements of the macros that would also preserve the behavior. The replacements can be done using appropriate aspects and executing the advice on the appropriate join points. But to what extent would this refactoring help, has to be validated.

IV. Conclusion
This paper takes a review of various refactoring techniques like Encapsulate Field, Generalized Type, Extract Method, Rename Method, Adding New Functions, Replacing Macro Definition. It also tells how we can do a technique implementation and tells advantages of refactoring. It also proposes new technique for refactoring Replacing Macro Definition Codes with Proper Program and Advice. Replacements can be done using appropriate aspects.

References
[1] [Online] Available : http://www.cs.usfca.edu/~parrt/course/601/lectures/refactoring/refactoring.html
[2] [Online] Available : http://weblogs.asp.net/gunnarpeipman/archive/2009/02/07/refactoring-rename-method.aspx
[3] Refactoring HTML: Improving the Design of Existing Web Applications By Elliotte Rusty Harold
[4] xUnit Test Patterns: Refactoring Test Code By Gerard Meszaros
[5] Fowler, M. Refactoring. Improving the Design of Existing Code. Addison-Wesley,1999.

Seema Kansal has done MCA from Punjabi University, Patiala. She is doing her Ph. D. degree in 'Refactoring Framework Development' from Singhania University, Rajasthan, India. She is currently teaching as assistant, lecturer, associate professor in Department of Computer Science at a reputed engineering college.