

Work Paper : Code Optimization using Refactoring

Piyush Chandi
M.TECH(ITW),
University School of Information and Communication Technology,
Guru Gobind Singh Indraprastha University,Dwarka,
Delhi-110078
India

Abstract

Refactoring is a formal and mechanical process, used to modify existing code in such a way that it does indeed become 'better' while preserving the program's intended functionality. In addition to improving a program's overall design, the refactoring process tends to yield code which is far easier to maintain and extend in the long run. It is widely believed that refactoring improves software quality and developer productivity. This paper explains the benefits of refactoring by applying combinations of refactoring methods on an unoptimized code and identify which combination of refactoring methods results in better optimization of code. Further the paper discusses works done by various people on refactoring, various code optimization techniques, classification of refactoring techniques based on functionality, code optimization tools and benefits and challenges of refactoring. The paper discusses the results and conclusions are drawn on the basis of results. The paper concludes with identification of future scope of work.

Introduction

Refactoring in common language is simply modifying the code without changing its external behaviour. The changed code is optimized code in terms of object oriented features such as Encapsulation, Polymorphism, Inheritance etc or in terms of performance such as Response time, Execution time etc. As per Fowler[1], Patrick Cousat[2], Miryung Kim[3], Refactoring is divided into six main groups depending upon functionality - :

- 1) Composing Methods - It deals with problems related to methods and parameters. It includes methods such as Extract Method, Inline Method, and Replace Parameter with Method. Extract Method takes input as line of codes and turns it into a method. Replace Parameter with Method is used when data in one parameter is obtained by making a request of an already known object. These type or refactorings solve problems such as Duplicate code in a single class, lengthy methods and long
- 2) Moving Features Between Objects- It is used for improving the software design. It includes methods such as Move Method and Extract Class.

Move Method is used when a particular method is used more frequently in another class than in class in which is defined. These type of refactorings solve issues related to large classes and multi-level classes.

- 3) Organizing Data- It is used to simplify working with data. It includes methods such as Replace Data Value with Object and Encapsulate field. Replace Data Value with Object method is used when the field needs additional data. For Example, we have a string field called Telephone number containing Telephone number of Employee as string and telephone number needs special behavior for formatting, extracting the area code etc. In order to accommodate this type of requirement we turn data item into object.
- 4) Simplifying Conditional Expressions- It deals with optimization and simplification of conditional expressions such as inner loops, for each loops etc.
- 5) Making Method Calls Simpler - It includes methods such as Preserve Whole Object and Introduce Parameter Object. Preserve Whole Object is used when instead of several parameters the whole object is passed as parameter.

6) Dealing with Generalization – It includes methods such as Pull Up Field, Pull Up Method, Push Down Field and Method and Extract Interface. Pull Method is used when there are common fields in sub classes and the field is moved to Super class. Pull Up Method is used when there is common method in sub classes and the method is moved to Super Class. Push Down Field and Method is used when fields and methods are relevant only to some sub classes. Extract Interface method is used when a classes having methods or set of methods is to be used by another classes.

4) refactoring emphasizes good design which speeds up the development process.

Further they stated that Refactoring can be done at any stage whenever a code requires tidying up but some common thumb rules for refactoring are - :

- 1) Refactor when adding functionality, to improve code comprehension or to rearrange the code affected by new functionality.
- 2) Refactor when you need to fix a bug.
- 3) Refactor when code is reviewed.

Literature Review

There are various definitions for Refactoring - :

Fowler[1] describes Refactoring as a noun is, “a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” . As a verb, to refactor is, “to restructure software by applying a series of refactorings without changing its observable behaviour”.

Software restructuring is defined by Opdyke [7] as, “the modification of software to make the software

- a) easier to understand and to change or
- b) less susceptible to error when future changes are made.

” Roberts D. Brant [8] gave a more formal definition of refactoring is provided as a program transformation that has a precondition and a post-condition that a program must satisfy for the refactoring to be easily applied.

Following definitions also were given by developers for refactoring as observed by Fowler[1] and Miryung Kim[3] - :

“Rewriting code to make it better in some way.”

“Changing code to make it easier to maintain. Strictly speaking, refactoring means that behavior does not change, but realistically speaking, it usually is done while adding features or fixing bugs.”

Fowler[1] and Miryung Kim[3] stated that Refactoring is done due to four main reasons - :

- 1) Refactoring improves the design of existing software
- 2) Refactored software easier to understand
- 3) Bugs are easily located when code is refactored

There are many issues and challenges associated with refactoring.

Opdyke[7] addresses the issue of refactoring being a behaviour preserving operation. According to Opdyke[7], the following seven properties must be fulfilled to define a refactoring as being behaviour preserving. Firstly, all classes must have at most one unique superclass. All classes must have distinct class names and distinct member names. Inherited member variables must not be redefined. Signatures must be compatible in member function redefinition. That is, when a subclass is redefining a method found in a superclass, the signatures of the method must be the same. There must be type-safe assignments, meaning the values assigned to an attribute must be of the same type as the attribute itself. Lastly, there must be semantically equivalent references and operation. That is, when given a set of inputs to a program and generating a set of outputs from these inputs, after applying the refactoring and running the program with the same set of inputs, the outputs must be the same as the original set of outputs.

The idea of behaviour preserving is defined differently by Roberts D.Brant[8]. Each program is thought to have a specification for it and that specification is satisfied (or unsatisfied) by a test suite. A refactoring is therefore behaviour preserving if it satisfies the original test suite. If a new component is added to the program, the program must satisfy the original test suite plus any additional tests. When satisfying the original test suite, one must recognize that this is the conceptual original test suite that is satisfied.

In general, Fowler[1] noticed that refactoring in and of itself is rather risky and has the potential to set back a programmer days and even weeks. Consequently, the first step in refactoring is providing a solid set of test cases to run before and after the refactoring.

There are some studies done on Refactoring. Multi layer refactoring has been applied on Windows softwares by a centralized refactoring team which developed refactoring

tools and processes. The binary modules refactored by the refactoring team had significant reduction in the number of inter-module dependencies and the number of post-release defects. Zimmermann and Nagappan[3] built a system wide dependency graph of Windows Server 2003. The quantitative analysis of Windows 7 version history shows refactored modules experienced higher reduction in the number of inter-module dependencies and post-release defects than other changed modules. The study by Zimmermann and Nagappan[3] is one of the first to show that refactoring changes are likely to be relatively more reliable than regular changes in a large system.

There are few Code Optimization techniques on which studies have been conducted. They are - :

1) Profiling

In this technique, analysis of the relative execution time spent in different parts of the Program is done because generally spent most of the time in few parts of the code. Optimization is done to only those part of the program where time consumed is Maximum.

2) Using a fast algorithm

Since to solve a program(say for example Sorting N numbers), many algorithms exist in literature, therefore we must use algorithms whose average running time is less.

3) Local Optimizations

Local Optimization is achieved at sub-program or at module level. It can be achieved by eliminating common sub-expressions or by using registers for temporary results or by using SHIFT and AND operators instead of addition and multiplication.

4) Global Optimizations

It is performed with the help of data flow analysis and split-lifetime analysis. It includes Code Motion, value propagation and strength reductions.

5) Space Optimization

It reduces the size of object by using the techniques of constant pooling and dead Code elimination.

6) Speed Optimization

It reduces the execution time of the program. Following techniques are used to achieve Speed Optimization - :

a) Loop unrolling - Full or partial transformation of a loop into straight code.

b) Loop blocking (tiling) - Minimizes cache misses by replacing each array processing loop into two loops, dividing the "iteration space" into smaller "blocks".

c) Loop interchange - Change the nesting order of loops, may make it possible to perform other transformations.

d) Loop distribution - Replace a loop by two (or more) equivalent loops.

e) Loop fusion - Make one loop out of two (or more).

Proposed Work

The thesis proposes to understand the impact on code in Visual Studio 2010(C#) on applying Extract Method and Encapsulate Field together or Extract Method and Extract Interface(in combination) together and identify which one optimizes code in better way.

Extract Method is one of Refactoring method available in Visual Studio 2010 which takes input as line of codes and gives output as a method containing the line of codes. This helps in increasing the modularity of code. Applying Extract Method yields restructured code which is concise and new methods are created based on code selection. New methods created can be parameterized or non-parameterized based on code selection. Return types of new method and parameter types are automatically decided by Refactoring tool thus there is no need to take care of return types and parameter list.

Encapsulate Method is used to protect class field from direct access by outside world.

Applying Encapsulate method yields encapsulated code. The law of encapsulation begs us to define a type's field data as private, while providing safe and controlled access to the underlying value using .NET properties.

Extract Interface method is used generalize classes by creating Interfaces containing common functionalities which are to be used across several classes. Applying Extract Interface method allows developers to select a group of existing type members to yield a new interface abstraction. Interfaces define a set of abstract members (properties, methods, and events) that a given type may support. The beauty of interfaces becomes clear when you understand that types in completely different hierarchies can implement the same interface. Given this, interfaces allow

us to obtain polymorphism across hierarchies, namespaces, assemblies, and .NET programming languages.

Assumptions: For study and research purposes, we assume our code is unoptimised such that all three methods - : Extract Method, Extract Interface and Encapsulate Field can be applicable to the code.

Algorithm I

- 1) Identify the unoptimised code.
- 2) Apply Extract method first preferably on code pages(.cs files) to obtain partial optimization as shown below
- 3) Next Apply Encapsulate field method on Class files as shown below to achieve second level optimization as shown below.
- 4) Use the optimized code for further programming

Algorithm II

- 1) Identify the unoptimised code.
- 2) Apply Extract method first preferably on code pages(aspx.cs files) to obtain partial optimization as shown below
- 3) Next Apply Extract Interface method on Class files as shown below to achieve second level optimization as shown below.
- 4) Use the optimized code for further programming.

Implementation

The three Refactoring Methods namely - Extract Method, Encapsulate Field Method have been analyzed and studied in stand alone fashion as well as in combination. All three methods when applied in combination help in optimizing the code in better way as compared to applying stand alone methods on the code. These Refactoring methods are available in Visual Studio 2010 only with C# language. Detailed Implementation of each method in stand alone fashion as well as in combination is shown below - :

1) Extract Method

This type of method helps in combing line of codes into a single Method. It is useful for modularizing, readability and understandability of code. It also improves extensibility of code.

2) Encapsulate Field Method

This method is used when we have a public data member of a class. Accessing public member directly by use of objects is not a good way of programming. .NET introduces a concept Properties which allows the programmer to set(writeonly), get(readonly) or both set and get the values of scalar variable. The advantage of property is that we can define the property mode as readonly or writeonly as per the requirement. Also we can apply validation before setting the values

3) Extract Interface Method

Once comfort level of working with interface types is achieved, it is very hard not to leverage their usefulness whenever possible. In a nutshell, interfaces define a set of abstract members (properties, methods, and events) that a given type may support. If so, the implementing type fleshes out the details as it sees fit. The beauty of interfaces becomes clear when you understand that types in completely different hierarchies can implement the same interface. Given this, interfaces allow us to obtain polymorphism across hierarchies, namespaces, assemblies, and .NET programming languages. The Extract Interface refactoring allows developers to select a group of existing type members to yield a new interface abstraction.

4) Extract Method and Encapsulate Field

Applying these two methods simultaneously will yield optimized code which is concise and encapsulated in nature

5) Extract Method and Extract Interface

Applying these two methods simultaneously will yield optimized code which is concise and extensible(reusable) in nature.

Results and Discussion

Applying Extract Method and Encapsulate field together will yield a optimized code which is concise , encapsulated , object oriented in nature. The code can be well managed because of the modularity. Extract Method has also an advantage that it can also create a new method containing parameters as per the selection of line of codes as shown in the above chapter.

Applying Extract Method and Extract Interface together the will refactored code will contain features of Encapsulation as well as Polymorphism.

The goal is to achieve best possible level of Object Orientation in the code.

Properties combine aspects of both fields and methods. Properties have many uses: they can validate data before allowing a change; they can transparently expose data on a class where that data is actually retrieved from some other source, such as a database; they can take an action when data is changed, such as raising an event, or changing the value of other fields. They can also be used for binding data to controls. Use of properties in the code can enhance the performance of code. It increases the reliability and well as degree of object oriented in the code. Also validations or data checkings can be applied in Properties.

By not using Extract Method and Encapsulate field refactoring methods together the code will be without Properties whose advantages are explained above. Also the code would be unstructured and unoptimised in nature.

Applying Extract Method and Extract Interface together will result in optimized code which is object oriented (with the concept of inheritance implemented) , concise , centralized in nature.

The benefits of using interface based programming, are outlined below - :

- 1) Enables developers to write loosely coupled systems. With the help of interfaces, if required in future, implementation can be simply changed thus making it easy for the application to adapt to changes.
- 2) Developers' roles are segregated. Developers writing interface based components don't have to worry about how it's being implemented. They only need to know how to invoke functionalities available within it. UI developers can start working on the UI without a concrete implementation of the interface. At the same time, component developers can concentrate in implementing the agreed interface definition. This enables parallel development as one does

not have to depend on another's work during the development process.

3) As long as the interface definitions are not changed, adding new features or re-implementing existing features for various reasons like changes in business rules will not break the existing system, rather makes it rich and efficient over a period of time.

4) Generates maintainable and clean source code - UI related code is separated from other parts of the system like data access layer and business/domain related layer. Unit testing that every developer should embrace can be utilised. It can be performed independent of the UI or any other layer consuming functionalities of the interface based classes.

5) With Interfaces multiple inheritance can be implemented in C#.

6) Code readability: An interface constitutes a declaration about intentions. It defines a capability of the class, what the class is capable of doing. If Isortable interface is implemented then it is clearly stated that the class can be sorted, same for IRenderable or IConvertible.

7) Code semantics: By providing interfaces and implementing them the concepts are separated in a similar way HTML and CSS does. A class is a concrete implementation of an "object class" some way of representing the reality by modeling general properties of real life objects or concepts. An interface define a behavioral model, a definition of what an object can do. Separating those concepts keeps the semantics of the code more clear. That way some methods may need an instance of an animal class while other may accept whatever object you throw at them as long as it supports "walking".

8) Code maintainability: Interfaces helps to reduce coupling and therefore allowing developers to easily interchange implementations for the same concept without the underlying code being affected. implementation of an interface can be changed easily by defining a new class that implements the interface. Compare that to replacing all references from a particular class and changing it in other classes.

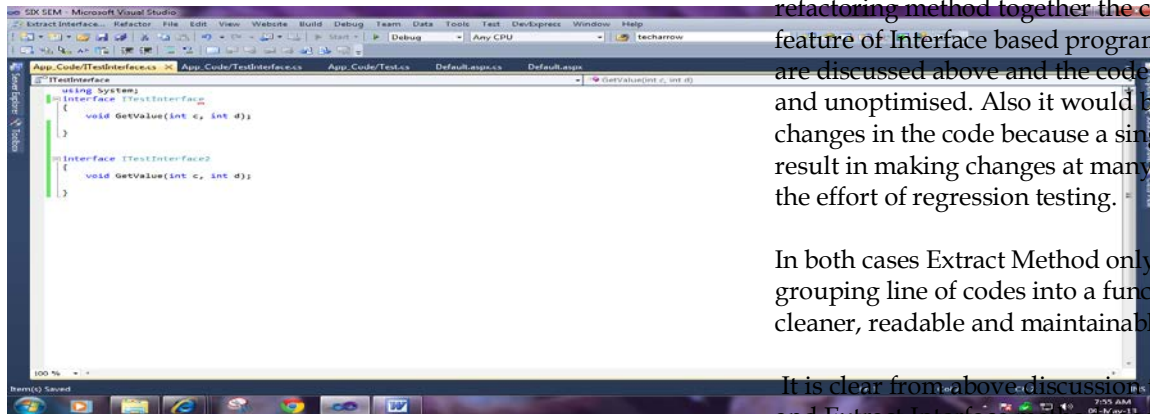
Also a particular interface can be used by many classes. Hence changes in Interface will automatically be reflected in all inherited classes thus maintaining modularity and reducing side effects of changing code. Also an interface can be implemented implicitly or explicitly to class as shown below. Explicit Interface Implementation is useful when two interfaces have same functions (same signature, parameter and type) as shown below. This is also another advantage of using Extract Interface method.

Fig 5.1 Implementing Explicit Interface on a class

By not using Extract Method and Extract Interface refactoring method together the code would lack the feature of Interface based programming whose advantages are discussed above and the code would be unmanaged and unoptimised. Also it would be difficult to track changes in the code because a single change in code would result in making changes at many places and also increase the effort of regression testing.

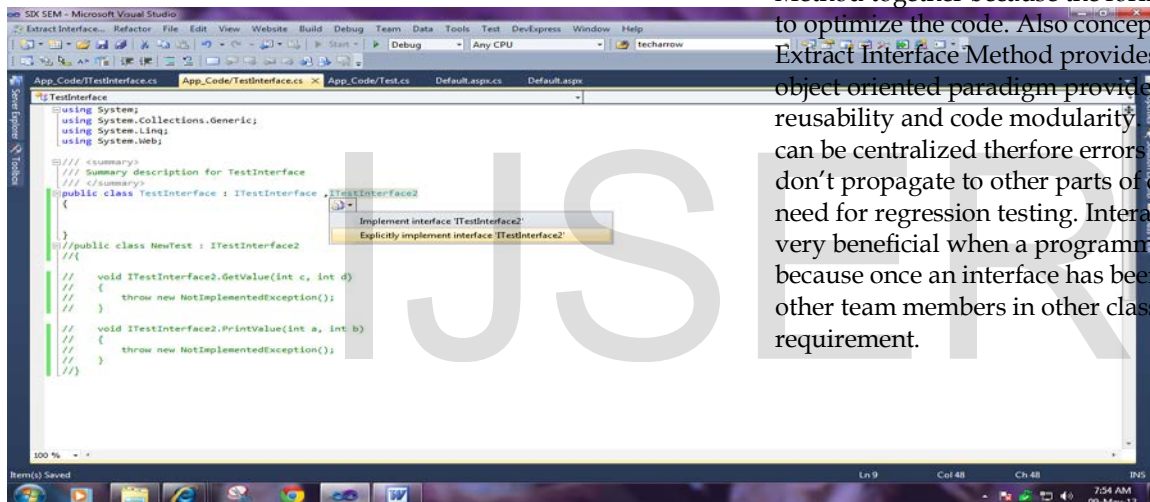
In both cases Extract Method only provides functionality of grouping line of codes into a function thus making the code cleaner, readable and maintainable.

It is clear from above discussions that using Extract Method and Extract Interface method together will be more beneficial than using Extract Method and Encapsulate field Method together because the former provides more options to optimize the code. Also concepts of inheritance which Extract Interface Method provides is a powerful attribute of object oriented paradigm provides facility for code reusability and code modularity. By using interfaces, code can be centralized therefore errors due to changes in code don't propagate to other parts of code hence reducing the need for regression testing. Interface based programming is very beneficial when a programmers are working in a team because once an interface has been created it can be used by other team members in other classes as per the requirement.



```
using System;
interface ITestInterface
{
    void GetValue(int c, int d);
}

interface ITestInterface2
{
    void GetValue(int c, int d);
}
```

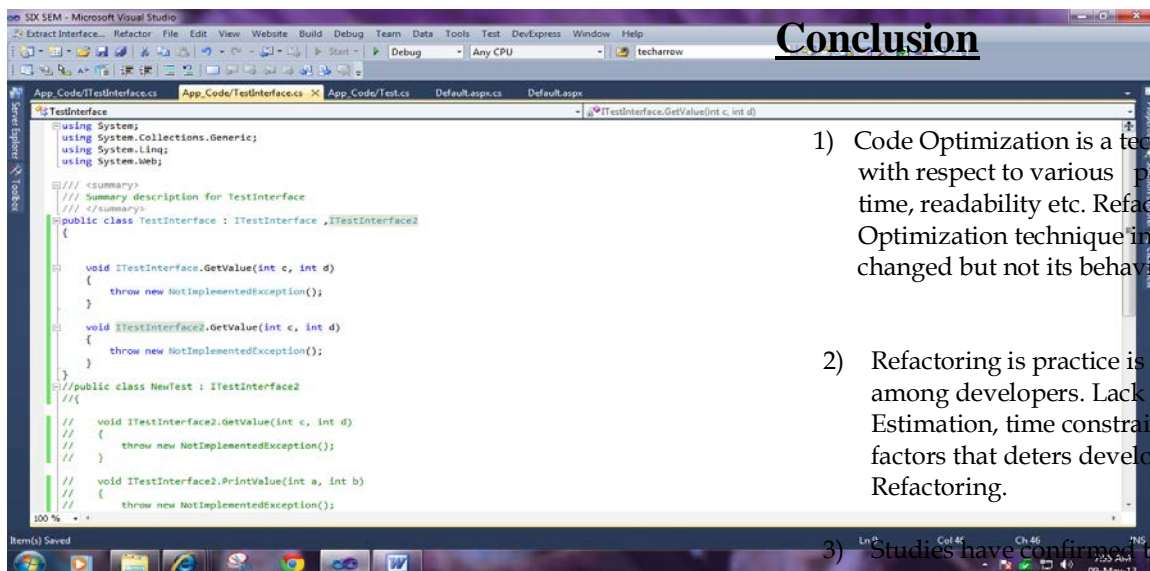


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

/// <summary>
/// Summary description for TestInterface
/// </summary>
public class TestInterface : ITestInterface, ITestInterface2
{
}

public class NewTest : ITestInterface2
{
    ///
    /// void ITestInterface2.GetValue(int c, int d)
    /// {
    ///     throw new NotImplementedException();
    /// }

    ///
    /// void ITestInterface2.PrintValue(int a, int b)
    /// {
    ///     throw new NotImplementedException();
    /// }
}
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

/// <summary>
/// Summary description for TestInterface
/// </summary>
public class TestInterface : ITestInterface, ITestInterface2
{
    void ITestInterface.GetValue(int c, int d)
    {
        throw new NotImplementedException();
    }

    void ITestInterface2.GetValue(int c, int d)
    {
        throw new NotImplementedException();
    }
}

public class NewTest : ITestInterface2
{
    ///
    /// void ITestInterface2.GetValue(int c, int d)
    /// {
    ///     throw new NotImplementedException();
    /// }

    ///
    /// void ITestInterface2.PrintValue(int a, int b)
    /// {
    ///     throw new NotImplementedException();
    /// }
}
```

Conclusion

- 1) Code Optimization is a technique to optimize code with respect to various parameters such as space, time, readability etc. Refactoring is one of the code Optimization technique in which code structure is changed but not its behaviour.
- 2) Refactoring is practice is still to gain popularity among developers. Lack of Refactoring tools, Cost Estimation, time constraints etc are some of the factors that deters developers to follow Refactoring.
- 3) Studies have confirmed that Refactoring reduces production bugs and also improve software design and also reduces inter module

dependencies.

- 4) Refactoring sequential programs for parallelism is time-consuming and error-prone. It also leaves the code less readable and less portable
- 5) To implement Refactoring, the developer must have complete and in-depth knowledge of Refactoring tool.
- 6) Extract Interface Refactoring method is more useful than Encapsulate Field Methodology.
- 7) Other Refactoring Methods such as Reorder parameters, Remove parameters etc when used can also optimize code

[1] M.Fowler : Refactoring:-Improving the Design of Existing Code, Addison-Wesley Professional,2000.

[2] Patrick Cousot, Radhia Cousot, Francesco Logozzo Michael Barnett: - An Abstract Interpretation Framework for Refactoring with Application to Extract Methods with Contracts,In VMCAI, 2011

[3] Miryung Kim, Thomas Zimmermann, Nachiappan Nagappan: A Field Study of Refactoring Challenges and Benefits, Technical Report, Microsoft Research, 2012

[4] T. Mens and T. Tourwe: A survey of software Refactoring, IEEE Trans. Software Eng., 30(2):126-139, 2004.

[5] Tom Mens et al. "A survey of software refactoring". TSE 30(2), 2004

[6] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim. Template-based reconstruction of complex Refactorings, IEEE International Conference on Software Maintenance, 2010

[7] Opdyke W.; Refactoring Object-Oriented Frameworks,1992

[8] Roberts, Donald B.; Practical Analysis for Refactoring, PhD Dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 1999

[9] Roberts, D., Brant, J., Johnson, R.; A Refactoring Tool for Smalltalk, Theory and Practice of Object Systems, 3(4):253-263, 1997. 1999

[10] Tokuda, Lance A.; Evolving Object-Oriented Designs with Refactorings;;1999

[11] Kerievsky, Joshua; Refactoring to Patterns; Industrial Logic; 1999

[12] C. Gorg and P. Weißgerber. Error detection by refactoring reconstruction, In MSR '05: Proceedings of the 2005 international workshop on Mining software repositories, pages 1-5, New York, NY, USA, 2005.ACM Press. 2005

[13] K. Beck. extreme Programming explained, embrace change., Addison Wesley Upper Saddle River NJ, p.103-115, 2000

[14] D. Dig and R. Johnson. The role of refactorings in API evolution,2005

Future Scope

- 1) Based on the study future work includes availability of more refactoring methods which includes inclusion of combination of Refactoring methods, Easier Code review etc which may be able to solve the problem of readability and portability as well.
- 2) We may also propose development of tools which may identify unoptimised code and which particular methodology will be suitable for the identified unoptimised code.
- 3) Refactoring must be supported for other languages such as VB,C++ etc. Currently Refactoring is only available for C# language in Visual Studio.NET 2010.
- 4) Level of Code Optimization could be measured by comparing and executing unoptimised code with optimized code under standard and same conditions. This is also one of area of research and development in Refactoring tools and would be of great help to developers. This when available will improve popularity and usability of Refactoring tools.

References

[15] R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi.
Refactoring a legacy component for reuse in a
software product line ,2005

[16] Tokuda, Lance A.; Evolving Object-Oriented Designs
with Refactorings; The University of Texas at Austin, 1999

[17] F. Kjolstad, D. Dig, G. Acevedo, and M. Snir,
"Refactoring for immutability,"2010

[18] D. Dig, C. Radoi, M. Tarce, M. Minea, and R. Johnson,
"Refactoring for loop parallelism,"

[19] J. Wloka, M. Sridharan, and F. Tip, "Refactoring for
reentrancy," 2009

[20] M. Mndez, J. Overbey, A. Garrido, F. Tinetti, and R.
Johnson, "A catalog and classification of fortran
refactorings , 2010

[21] E. Murphy-Hill, C. Parnin, and A. P. Black. How we
refactor, and how we know it,2009

[22]R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi.
Does refactoring improve
reusability? , 2006

IJSER