A Survey : Code Optimization using Refactoring

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

Abstract

This position paper identifies emerging trends in refactoring research particulary Refactoring, and enumerates a list of open questions, from practical as well as a theoretical point of view. We suggest these directions for further research based on our own experience with refactoring, as well as on a detailed literature survey on software refactoring.

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.In terms of implementation , Refactoring is divided into six main groups depending upon functionality - :

1) Composing Methods - deals with problems related to methods and parameters. It includes methods such as Extract Method, Inline Method, and Replace Parameter with Method. These type or refactorings solve problems such as Duplicate code in a single class,lengthy methods and long parameter lists.[1][2][4]

2) Moving Features Between Objects- Used for designing the software design.

It includes methods such as Move Method and Extract Class. It is used to solve related to large classes and multi-level classes. [1][2][4]

3) Organizing Data- used to simplify working with data. It includes methods such as Replace Data Value with Object. [1][2][4]

4) Simplifying Conditional Expressions- It deals with optimization and simplification of conditional expressions. [1][2][4]

5) Making Method Calls Simpler – It includes methods such as Preserve Whole Object and Introduce Parameter Object. [1][2][4]

6) Dealing with Generalization – It includes methods such as Pull Up Field, Pull Up Method and Push Down Field and Method. [1][2][4]

Extract Method

This method turns a group of lines into function or method. It helps in modularizing the code. It helps in reusability of code and Method overriding.

Example

Original Code

```
void printOwing(double amount) {
printBanner();
//print details
System.out.println ("name:" + _name);
System.out.println ("amount" + amount);
}
```

Refactored code after applying Extract Method

```
void printOwing(double amount) {
printBanner();
printDetails(amount);----------------------→ Refactored line
}
void printDetails (double amount) {
```

```
System.out.println ("name:" + _name);
System.out.println ("amount" + amount);
}
```

Inline Method

If the function contains relatively less code lines so that it can be expanded in the calling function body, then we should use this method to refactor the code.

Example -:

Code before Refactoring

```
int getRating()
{
return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
```

Code after Refactoring

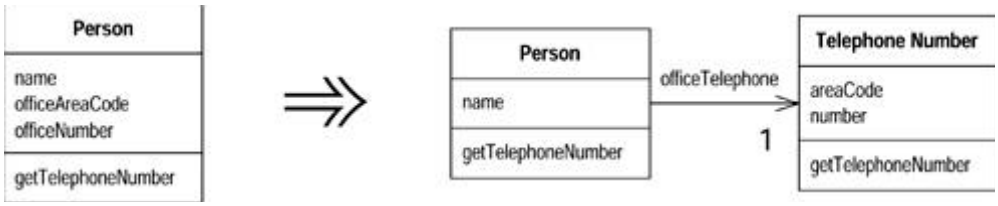
```
boolean moreThanFiveLateDeliveries()
{
return _numberOfLateDeliveries > 5;
}
int getRating()
{
return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Move Method

This method refactoring is used when a method of one class is to be used by another class very frequently. We identify source method, source class and target class. The chosen method gets copied into target class. Now the method can directly be called by target class member functions instead of referencing it through another class.

Extract Class

This method is used when a class becomes too big and complicated so that it is unmanageable. This method helps in splitting a large class into smaller classes so that code becomes reusable and manageable[1]



Refactoring eases Software Maintenance by reducing post production bugs but some empirical studies paints a contradictory picture. Ratzinger found that, if the number of refactoring edits increases in the preceding time period, the number of defects decreases. On the other hand, Weißgerber and Diehl found that a high ratio of refactoring edits is often followed by an increasing ratio of bug reports[5]. Various studies were conducted to prove the Refactoring benefits on the software development and maintenance.

A) The refactoring definition in practice seems to differ from a rigorous academic definition of behavior-preserving program transformations. The survey participants perceived that refactoring involves substantial cost and risks, and they needed various types of tool support beyond automated refactoring within IDEs.[5] B) The interviews with a designated Windows Refactoring team provide insights into how system-wide Refactoring was carried out in a large organization. The team led a

centralized refactoring effort by conducting an analysis of a de-facto dependency structure and by developing custom refactoring support tools and processes.[5]

C) 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.[5]

The studies have shown errors occurred during testing or occurred on live application are due to unrefactored code or due to lack of support of refactoring by Application platforms.

There are platforms that support Code Refactoring.We briefly discuss some Refactoring tools.

1) **SmallTalk Refactoring Browser[1]**

The SmallTalk Refactoring browser was the first refactoring tool developed and is the basis for the structure of the refactoring support provided in eclipse.

The browser bases all its refactorings on three principles. First, is that the refactorings can be automated, (although not completely automated) second is that they are provably correct and lastly is that more complex refactorings can be created by composing primitive refactorings. The Smalltalk Browser was motivated partially by the desire to refactor to improve program comprehension. By restructuring the code, one sees where the errors lie and one is able to better understand the system.The browser follows four specific criteria. Firstly, the refactorings are integrated into standard development tools, as occurs with this refactoring browser, IntelliJ. Secondly, the refactorings must be fast, at least as fast and more than likely faster than it takes to manually refactor. Thirdly, the refactorings should not be purely automatic, that is, as in Eclipse and the Smalltalk browser, the refactoring should to require at least some user interaction. Lastly, the refactorings must be reasonably correct generate user trust of the tool. The SmallTalk refactoring browser assumes there is developer intelligence in executing the refactoring process.

In the implementation of the browser, the basis of the source code transformation is a parse tree to parse tree transformation, not a string to string transformation.

Consequently, a tree matching system was developed in1999. Each refactoring also satisfies a certain number of preconditions before implementing the transformation.

Finally, functionality to support undo via a change history was next in line to be

worked on, at the time of writing of report.

2) **Design Pattern Tool[1]**

The tool was developed by Cinneide to refactor to design patterns. The Design Pattern Tool is implemented as a four layer architecture of

a) Design Pattern Transformations

b) Minitransformations

c) Helper Functions, predicates and refactorings

d) Abstract Syntax Tree operations.

The tool focuses on behaviour preservation. It outlines minitransformations

as the method by which a pattern is introduced into an application. The tool

contains mini-transformations conceptually similar to the sequential and complex refactorings introduced in this research, but lacking the integration of existing tool support. There is no follow up work to this research and thus its popularity and use of the tool is minimal**.**

3) Eclipse[1]

Eclipse is an open source development environment by IBM [Eclipse03]. Eclipse supports numerous plugins, and the capability to develop and support self-made plugins. Its refactoring support is ever-growing and the refactoring class structure is based on that of SmallTalk.

Eclipse was chosen as the development environment for this research due to its

availability of code, plug-in support and refactoring support. Eclipse contains

Refactoring wizards, refactoring classes and change classes to handle each individual refactoring.When a refactoring is called, the specific refactoring wizard is launched and user input is taken. When given the "go-ahead" from the user to perform the refactoring (i.e. 'OK' button clicked), preconditions are first checked. If the preconditions pass, an instance of a Change class is created and a

change operation is performed. A ChangeOperation class performs the actual are modification at the source code level. After the change operation, postconditions checked. All three of these steps: preconditions, change operations and post-

conditions are encompassed in a specific refactoring class, a subclass of a

generic refactoring class that exists in every refactoring wizard. The refactoring wizard is launched from any one of the available refactoring menus

4) IntelliJ[1]

IntelliJ, created by JetBrains, advertises itself as the industry leading Java

integrated development environment. It boasts features such as new

Refactoring support, intelligent code assistance, Java development features for rapid web application,code inspection tools, integrated versioning system and an open application programming interface for third party

plug-in support has three types of code-completion, 100% of its features accessible by mouse and a friendly user interface. While the refactoring source could not be viewed, a trial version of IntelliJ is available for download and some experimentation was performed with this version.The refactoring support variable or method are of course supported, as are more complicated rename refactorings provided by IntelliJ is extensive.

Low-level refactorings such as such as Extract Interface, Replace Inheritance with Delegation, Encapsulate Fields, and Replace Constructor with Factory Methods.

An apparent limitation of IntelliJ however, is its somewhat blind refactoring capabilities. When extracting an interface or extracting a superclass, one is

not provided with any options (e.g. what package to place the class, imports to include, etc). The package in which the selection occurred when the

refactoring was activated is where the class/interface is created. The strong

developer only enters the name of the new class. Secondly, IntelliJ does not

provide a refactoring preview.

A small box appears showing the code of the new refactoring and the

number of changes that will occur, but it is not as intuitive as the preview support provided in eclipse. The developer is left relying on somewhat

blind trust that the refactoring will be performed correctly.

Nevertheless, IntelliJ does perform the refactoring correctly, and makes updates through the entire source, respective to the refactoring.

Several studies have been done to assess the benefit of Refactoring. Some studies

favour refactoring by advocating that Refactoring directly effects bug count of Application, reduces Design complexities , modularizes and optimizes the code, increases Software Quality and reusability factor. Ward Cunningham drew the comparison between debt and a lack of refactoring: a quick and dirty implementation leaves technical debt that incur penalties in terms of increased maintenance costs.

But some studies found out Refactoring increases bug count, maintenance cost and also leads to many structural and design changes.

There are several challenges associated with Refactoring. Developers point out that large –sized code, lack of time and motivation, lack or limited availability of Refactoring tools, regression testing execution , difficulty in merging code, working and coordinating with large teams , maintaining code history and versions are some of the challenges they face while implementing Refactoring.

There are various definitions for Refactoring . Many researchers have given different definitions for refactoring based on their experience and study. The various definitions that exists are - :

1) Refactoring is a fix . It is done to correct a bug and ensure that the same bug does not occur again.
2) Refactoring is a change. Change the code to satisfy the requirements.
3) Refactoring is a “do-over”.
4) Refactoring is cleaning up code, improving readability, enhancing usability, fixing, changing, doing over, the concept of refactoring is a common- sense recognition that developers can continually improve

There are several risks associated with refactorings. The primary risk is failure while doing regression testing . The refactored code may result in failure of already correctly running Application or modules. Also it is difficult to measure the size ,correctness and success rate of refactored code.

Literature Review

There are various definitions for Refactoring defined by various persons.

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” [1]. As a verb, to refactor is, “to restructure software by applying a series of refactorings without changing its observable behaviour” [1].

Software restructuring is defined in [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.

” Similar to the first definition given, this could be a definition for refactoring.

In [8] 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[1][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."

Refactoring is done due to four main reasons [1][3] - :

1) refactoring improves the design of existing software
2) Refactored software easier to understand
3) Bugs are easily located when code is refactored
4) refactoring emphasizes good design which speeds up the development process.

Refactoring can be done at any stage whenever a code requires tidying up but some common thumb rules for refactoring are[1][3] - :

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. There are may 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[7].

The idea of behaviour preserving is defined differently in [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 riginal test suite. If a new component is added to the program, the program must satisfy the original test suite plus any additional tests. [8] When satisfying the original test suite, one must recognize that this is the conceptual original test suite that is satisfied.

In general, refactoring in and of itself is rather risky and has the potential to set back a programmer days and even weeks [1]. 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 built a system wide dependency graph of Windows Server 2003[3]. 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. Our study is one of the first to show that refactoring changes are likely to be relatively more reliable than regular changes in a large system[3]

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
Maximuim.

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).

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] There are various tools and methodologies to implement Refactoring . The tool

and Methodology to be chosen depends upon nature of Application and problem.

Future Scope

Based on our study we propose future work on Refactoring tools such as more refactoring methods which includes inclusion of combination of Refactoring methods, Easier Code reviews, etc. Also they may be able to solve the problem of readability and portability as well.

References

[1] M.Fowler : Refactoring:-Improving the Design of Existing Code, 2000.
[2] Patrick Cousot, Radhia Cousot, Francesco Logozzo Michael Barnett: - An Abstract Interpretation Framework for Refactoring with Application to Extract Methods with Contracts,2011
[3] Miryung Kim, Thomas Zimmermann, Nachiappan Nagappan: A Field Study of Refactoring Challenges and Benefits,2010.
[4] T. Mens and T. Tourwe: A survey of software refactoring. IEEE Transactions on Software Engineering,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[2010]
[7] Opdyke W.; Refactoring Object-Oriented Frameworks,1992
[8] Roberts, Donald B.; Practical Analysis for Refactoring,1999
[9] Roberts, D., Brant, J., Johnson, R.; A Refactoring Tool for Smalltalk,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 byrefactoring reconstruction,2005
[13] K. Beck. extreme Programming explained, embrace change., 2000
[14] D. Dig and R. Johnson. The role of refactorings in API evolution,2005
[15] R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi. Refactoring a legacy component for reuse in a sopftware product line ,2005