OBJECT-ORIENTED REFACTORNG PROCESS DESIGN FOR THE SOFTWARE REUSE

Jong-Ho Lee *Nam-Yong Lee* *Sung-Yul Rhew*
jhlee@selab.soongsil.ac.kr, {nylee, syrhew}@computing.soongsil.ac.kr

ABSTRACT

The company invests its time and money for temporary maintenance to satisfy the fast change of the computer use environment and the user's demands. Therefore, various problems occur including low performance because of duplication of codes and unstable structures from the restructuring and redevelopment.

Furthermore, if a developer, who did not participate in the initial process of development, wrote additional program codes to upgrade or restructure, it would cause many problems such as lack or loss of development documentation, understanding of documentation and reuse of existing program language.

This study, Object-oriented Refactoring Process, suggests that the developers can reuse object unit to overcome the limit of the reusing code. In addition, developers are also able to have the positive results such as improvement of system performance, a decrease in the cost of development and maintenance, and optimizing structure and class by adjusting the project called " D2D™ ", which is a case tool for developing windows system from Company D.

1. INTRODUCTION

Systems developed and used by several developers have to be maintained in a short time and on a large scale because of the rapid changes of the internet-environment and increasing users' demands. But, code, which is executed from other developer, has a problem hard to understand and reuse because of missing and insufficient document, the existing system developer's absence. And that code causes decline in performance. It also needs much time and costs in order to solve these problems [1].

Software Reusing is to reuse products, like descriptions, design information or documents produced during new software development, when new software is developed. But enterprise's software reusing is not organized through their own documents and experience which is pertinent to the developer. Effective software reusing is not achieved because pertinent developer is not available or the source code's maintenance is not systematic. So analyzing structure and function of system, solving problems and reusing system software operating through reengineering is more economic and effective. [3,4]. This research presents object-oriented refactoring process, applies *D2D*™ software, which is a window system development tool developed by D company with C++ and shows process-achieved step.

2. RELATED RESEARCH

2.1 Serge Demeyer and Stephane Ducassse's Refactoring Process

It is a five step technique based on class diagram. [5]

Step 1: Create Subclass
Step 2: Move Attribute
Step 3: Move Method
Step 4: Split Method + Move Method
Step 5: Clean-up

The merit of this technique is that it is easy to find elements, which are easily passed by because these elements change their position after the creation of a subclass. That is, it is easy to make a list, manage and change elements.

The weak point is that it makes an error to move essential elements because it makes the subclass in advance and start to refactor or recognize different elements as the same one wrongly, without different definitions.

2.2 Martin Fowler's Refactoring Process

It is a four step technique based on class diagram. [3,6]

Step 1: Extract Method
Step 2: Move Method
Step 3: Apply Extract and Move Method (reapply step 1 and 2 which is insufficient)
Step 4: Replace Temp with Query

The merit of this technique is to abstract elements, move repeatedly and change to the right position so that it can optimize for improved system performance with short-move. This needs no addition at space and class creation because query is used as a temp.

0-7803-7090-2/01/$10.00 © 2001 IEEE.

The weak point is that it makes unnecessary repeated position moving even of the optimized element.

3. REFACTORING PROCESS DESIGN

Existing Refactoring processes show how to improve the whole project's performance through modifying some elements in class, but organized work process and target selecting was deficient.[7]

This research shows organized process, which can be applied to a real project in industry that maintenance is happened frequently. Figure 1 shows each process step in this research.

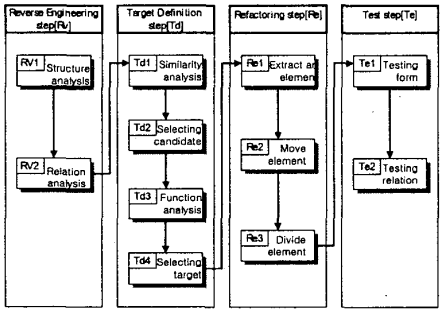

Figure 1 Detailed Refactoring Process

3.1 Reverse-Engineering Step [Rv]

Reverse-Engineering step is to analyze structure, relation and basic data that can refactor. It is to analyze what data is in function and their relation, adjust them and make them into a document.

[Rv1] Structure Analysis : This is a step to Analyze components, which belong to each project, class, and fluent including class and to understand the structure of used code for the whole project and work domain. This step makes the next production.

A. Class Structure Analysis : This is to analyze the form of each class with object-oriented Reverse-Engineering which is similar to Rational Rose.

B. Function Structure Analysis : This is to analyze the form of function (name, parameter, return type, access specifier) which belongs to class with object-oriented Reverse-Engineering which is similar to Rational Rose.

C. Component Structure Analysis : This is to make a fluent, function, class list which each component includes

[Rv2] Relation Analysis : This is a step to analyze relations with the component, which belongs to the whole project, and class, which these include. This step makes next production.

A. Class Relation Analysis : This is to analyze class relation, which each component includes using object-oriented Reverse-Engineering tool of Rational Rose company. Inheritance and Parameterized Class are analyzed in points of view on structure of class and the relation like Association, Aggregation and Composition are analyzed in based on functions calling relation.

B. Component Relation Analysis : This is to draw a component relation diagram from relation which is related with class in other component.

3.2 Selecting Target Step [Td]

This is a step to set the range of refactoring by information about class and method calculated at Reverse-Engineering step, and adjust these functions and forms.

Table 1 Form and Function Similarity Standard

Section	Classification	Detailed Classification	Marking Annotation
Form	Exactly Same	Form is exactly same.	Exactly Same
	Except Annotation	Annotation is different.	Annotation (contents)
		Annotation's existence.	Annotation (existence)
	Declare Difference	Parameter name is different.	Declare (parameter)
		Fluent name is different.	Declare (fluent)
	Line Feed	Change line and coding.	Line Feed
Function	Different fluent name	Fluent name is different.	Fluent name
	Different code	Code length difference	Code (length)
		Part of code omitted	Code (omit)
		Part of code different	Code (part)
		Algorithm is alike.	Code (algorithm)
	Parameter difference	Number of Parameters	Parameter (number)
		Parameter type	Parameter (type)
	Return value difference	Return value difference	Return value

[Td1] Similar Analysis : Set similar value and enact a standard by using McCabe which is a Reverse-Engineering tool and express them as a percentage. 90-100% is Very High Similarity and 75-89% is High Similarity by form and function. This step is to analyze all other functions of the existing work-target project by similarity standard shown table 1. [8]

[Td2] Selecting Candidate : set function, which is if High Similarity is over 75%, to refactoring candidate according to result of Similar Analysis and abstract.

A. Form Similarity : This function which has different function but the same form, changes all names to an other or makes a generic class or virtual function for maintain in consistence.

Table 2 Refactoring Activity

Refactoring Activity	Pre Condition	Detailed Work Guide
Moving component to component	When class in A component group calls special class in B component group	Copy class to A component. Modify call relation from class in A component.
Coordinating to super class	If class A and B has inheritance or association relation in same component, inherited B class work nothing.	Move element and method in B Class B to A class and coordinate. Delete B class. Modify other class which has direct call relation with B class to A class.
Expressing super class	When A and B class is in same component group but no call and these methods of two classes has some similar function.	Make same part from method which both A and B class has to super class. Modify to inherit from super class C that is made from between A and B class. If different function exists and it have to represent individually, It makes and inherits from A and B class.
Moving to upper class	When between A and B class has inheritance or aggregation relation in same component group or increase perfection of A class by moving method of B class to A class. When frequently called to use special method in other component group class.	Copy and Move pertinent method.
Making imaginary	When several class in same component group has method that has same signature but different algorithm	Realize individually algorithm from lower class inherited by imaginary function in upper class
Making normal class	When several classes are different in same component group but similar method and structure	Make normal class to upper class and relation

B. Function Similarity : The same function, but different function understands relation with class after this step, combines higher class, abstract super class and changes to one form without moving.

[Td3] Function Analysis : This is to analyze the exact achievement function of the candidate list abstracted at td2 step. This analysis understands its real function and verifies how similar it is from the code, which is highly similar, in the result showing a different region function name and fluent but similar algorithm by McCabe tool.

[Td4] Selecting Target : Get methods and classes from Td3's step if they function similarly from Td3's step.

3.3 Refactoring Step [Re]

This is a step to change the target elements for system performance improvement. To delete the same element, move methods and properties and divide to make an independent class is achieved. It is a step to move each element to an other class, delete the same element and move them to functions used much by methods and properties. This is the way to call from previous function after moving. Table2 shows the prior condition and detailed work guide of Refactoring Activity.

[Re1] Abstracting Element : To abstract elements that accompany Refactoring work and function and form and make an accompanying list.

[Re2] Moving Element : To delete or move same or similar elements to other class, which is called frequently.

[Re3] Dividing Element : To divide independent function class and make super class.

3.4 Verification Step [Te]

This is a step to verify form and function of class and element refactored. This tests that all functions are working correctly after refactoring and functional elements' performance wanted to improve additionally.

[Te1] Form Verification : To arrange the method of class coordinated or divided, number of parameter of property, type of parameter, annotations, function name, local fluent and return value.

[Te2] Function Verification : To verify whether the function is working correctly or not.

4. APPLYING TO REFACTORING PROCESS BASED OBJECT-ORIENTED

4.1 Environment Application

This chapter investigates strong and weak points of its process and measures its result. Applying the Refactoring process this research presented for increasing performance of *D2D*™ window environment system developing tool developed by D company, *D2D*™ is now used on C/S environment, Server application is developed by C language, and Client application is developed by C++. *D2D*™ of which client code for refactoring has more than 300 classes and 4000 methods is large scale system. *D2D*™ deals with a few million task of data on real-time per a day and it's speed of performance is very important thing. So it is suggested that the running module size of

software should be smaller and the speed of object's creation be faster.

To solve these problems, in this research, methods that have duplicated function and are in several classes are put into a super class. The focus of this work is to change in the same name's method to do the same things. As a result, maintenance and reusability are improved because the code size become s smaller and specific documentations are deduced.

4.2. Step of Reverse-Engineering [Rv]

4.2.1 [Rv1] Structure analysis

Using the reversing function of Rational Rose2000 tool we can do reengineering analyzing. It can analyze a class of each component in the structure of class, and elements of the class like methods' name, parameters and structure. Figure 2 shows the results.

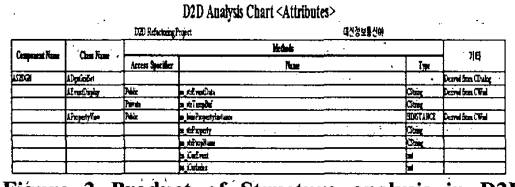

Figure 2 Product of Structure analysis in D2D project

4.2.2 [Rv2] Relation Analysis

Using reversing function of Rational Rose2000 tool, we can extract a relationship of each class like inheritance, dependency. And we're able to get a relationship of all classes and classes that are concentrated on. figure 3 is a production of analyzing a relationship of D2Dcommon's component's classes.

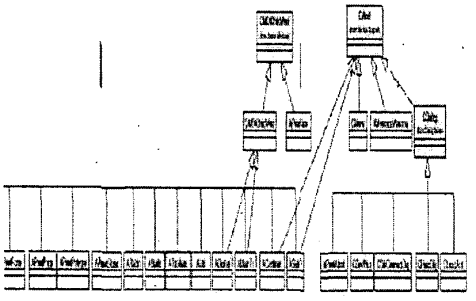

Figure 3 Product of relationship analysis in D2Dcommon group

4.3 Step of selecting target [Td]

This is the selected step to do refactor. It analyzes the function and similarity of data obtained from the reengineering stage. Among the data we selected refactoring candidates, we decide which targets to work refactoring practically.

4.3.1. [Td1] Similarity analysis

The reengineering tool, McCabe 6.0 of McCabe company, deduces similar form functions from all sources. It is operated on the basis of McCabe similar complexity v(g) 100%~ 75% similarity. We can know most similar methods are concentrated to D2Dcommon and D2Dcore component in figure 4.

● Source Component : D2DCommon, Class : AAPPObj, Method : CreateTrWindow

Component	Class	Method	%	P/C	P/T	Comments	Method	Local	Return	Type	Function	++	--	Check
D2DCom	CMemoryData	ReadToken(char*)	75	X		X	X	X	X	X	X		0	
D2DCom	AListRow	GetPropertyTab()	75	1	1	X	X	0	X	X	X		0	
D2DCom	CStream	GetDataPtr()	75	X	0	0	0	0	0	0	0		0	

※ P/C : Parameter Count, P/T : Parameter Type,

Figure 4 Product of similarity analysis

4.3.2 [Td2] Selection of Refactoring Candidate

As a result of similarity analysis , we deduce a candidate to do refactor from those with the similarity value over 75 %. In this research A Class group and Q Class group are selected as candidates because they have 90% similarity at least in the form and function in D2Dcore and D2Dcommon.

4.3.3. [Td3] Function Analysis

This stage is for analysis and comparison between the functions of method that have similar form in the similarity analysis step. In figure 5 we can see the result of source code products that are classified work unit documents again for function analysis, and functions analysis that the source code performs practically.

4.3.4 Selection of Refactoring Target

In the classes of the selection of candidate step, we choose the D2Dcommon component and D2Dcore component that perform the most important function and are visual component collection. Each of the two components has 18 classes. And each all of the classes in D2Dcore Class has

dependency 1:1 on a similar naming Class included in D2Dcommon.

D2Dcommon Class is a class that MFC supplies additional functions to basic control using API and MFC classes. D2Dcore Class is made to perform necessary tasks for creating an interface design tool. Because these two classes' methods have many similar forms and functions, these are selected as refactoring targets.

4.4 Step of the Refactoring [Re]

4.4.1 [Re1] Extraction of an element

We abstract to have a identical function and form, selecting high functional similarity(Alist and Qlist) with similar form among 36 classes, 1196 methods of D2Dcommon and D2Dcore to select as a candidate class.

5. PERFORMANCE ESTIMATION

5.1 Compare with existing Process

The next Table displays a comparison of the existing process research.

Table 5 Compare with existing Process

Process	Process to present on this thesis	Existing process	
		Demeyer &Ducassse	Fowler
Core Diagram	Class Diagram Sequence Diagram Component Diagram	Class Diagram	Class Diagram
Main Refactoring Technique	Polymorphism Inheritance	Sub Classing	Method Moving
Destination code selection Level	Project	Class	Method
Application of process	Easy	Difficult to find target Source	Difficult to Determine Iteration Phase
Perfection of Reengineering Process	All Process Represent	Technique level	Technique level
Maintenance	High	Low	Low
View of destination system	Behavior Relation Architect	Relation	Relation

Existing research is the technique that only refines the existing code by emphasizing some partial characters of object-oriented. On the other hand, this thesis makes it possible for users to apply in the actual work easily by presenting each phase and details on the aspect of reengineering. Moreover, it is possible to overcome errors se easily made in concentrating only on the relationship of classes, because we present techniques that can Reverse destination system in various points of view.

5.2 Improvement of Class's Perfection

Through the process we have presented in this thesis, we made classes have enough attributes and enabled them to improve performance as an refining object-objected system through deleting unnecessary relationship of classes by applying various equipment in object-oriented supporting for software system refining.

Table 6 Refactoring Accomplish Work table

Working content	D2Dcore	D2Dcommon
Class moving among component	In D2D common D2Dcore (3)	0
Integrated to super class	3	5
Abstraction super class in same component to include similar method	2	3
Method moving to higher class	34	21
In higher class creation by virtual function	21	12
General class creation	2	2

We replicated the three classes in D2Dcommon called by a compounding method in D2Dcore and integrated classes in D2Dcommon and D2Dcore used by unnecessary inheritance into higher class. Also the class has several methods which afford similar function in the same component abstract the super class and can inherit it. By the principle of inheritance, we treated the method necessary to move into a higher class. Moreover the method having the same form and different algorithm is declared in a higher class as a virtual function, embodied in a lower class, and makes the same-form class having different forms among classes as general class. Table 6 shows contents of Refactoring work accomplished.

5.3 Execution speed improvement

D2Dtm is a tool that in one day treat on average one million data by real time. Execution speed is very important as element software execution module size must be small and the necessary generation speed must quickly be instituted.

For evaluation generation speed of object, this test has a PC environment that Intel Pentium III 700Mhz CPU and 256M RAM and in this environment evaluated that by generation speed each 1000 pieces to modified Qdit class and Alist class. Table 7 display comparison value to Refactoring work execution size change of before and after and speed change.

Table 7 comparison with before and after Refactoring work execution

Work content	Before modify		After modify	
	D2D Common	D2D Core	D2D Common	D2D Core
Execution module size (DLL Size)	913k	1529k	809k	813k

6. CONCLUSION AND AFTER THIS RESEARCH DIRECTION

This research defines similarity analysis technique in reusing software and core process for refining object-oriented software and improving reuse.

And this research examined the merits and demerits of this Process which presented in this study object-oriented Refactoring applied to D2D™ System of D Company design and measured the result.

Because the Scale of Software and maintenance costs are increasing day by day, reuse of object or component becomes more important for conquest of newly software crisis' occurrence. Accordingly further in the Refactoring process fined elements force unit, functional database storage development techniques of reuse analysis element and techniques for making a database must be researched, detail process to each process phase must be developed, a prototype model must be researched in life usable software realization.

Especially, to apply component techniques that increase interest in reuse, by function unit partition or movement have become easier, object-oriented component abstraction that make component and research of development field of automatic tool making component must progress all the more.

7. REFERENCE

[1] Ivar Jacobson, Martin Griss, Patrik Jonsson, "Software Reuse", Addison- Wesley, 1997.

[2] Carma McClure, "Three Rs", Prentice Hall, 1992.

[3] Martin Fowler, "Refactoring - Improving the Design of Existing Code", Addison Wesley Longman Inc., 1999.

[4] Ivar Jacobson, "Re-engineering of old systems to an object-oriented architecture", OOPSLA, 1991.

[5] Serge Demeyer, Stephane Ducassse, "Object-Oriented Reengineering", OOPSLA, 1999.

[6] Martin Fowler, "Refactoring: Improving the Design of Existing Code", OOPSLA, 1999.

[7] Jin-Ho Park, Jong-Ho Lee, Sung-Yul Rhew, "A Study of Re-Engineering Refactoring Technique for the Software Maintenance and Reuse", KISS, Proceeding of The Spring Conference, 2000.

[8] Hac-Yun Na, Jong-Ho Lee, Sung-Yul Rhew, "A Study of similarity analysis of function unit module for reusing code", KISS, Proceeding of The Fall Conference, 2000.