Evaluating the Impact of Refactoring of Class Hierarchies for Software Maintenance

Hnin Pwint Phyu
University of Computer Studies, Yangon
ahpwint@gmail.com

Thi Thi Soe Nyunt
University of Computer Studies, Yangon
thithisn@gmail.com

Abstract

The class hierarchy is an important step of an objected-oriented software development. Designing and maintaining the class hierarchy for reverse engineering is a difficult task. However, Formal Concept Analysis (FCA) is a strong tool which provides a natural theoretical framework for class hierarchy design and maintenance because it can guarantee maximal factorization while preserving specializing relationships. Relational Concept Analysis (RCA), an extension of FCA, is an efficient approach to allow dealing with entities described by binary attributes and by relations with the other entities. Moreover, object-oriented design metrics are essential part of software environment. So, our proposed system consists of two main tasks. First, we propose an approach for refactoring class hierarchy in object-oriented design with the theoretical strength of FCA and RCA. Next, we focus on a set of object-oriented design metrics that can be used to evaluate the impact of the resulted object-oriented design.

***Keywords*— FCA, RCA, Class Hierarchy, Refactoring, Object-oriented Metrics**

1. Introduction

When an object oriented software or model becomes bigger and bigger, duplicated elements start to appear, decreasing the readability and the maintainability of the software. Designing and maintaining class hierarchy is a crucial task in reverse engineering. A well designed class hierarchy makes the software easier to understand, maintain and reuse. In designing a class hierarchy, classes, properties/attributes, associations and methods are added and modified as the software evolves [8]. So, we propose Formal Concept Analysis (FCA) for factoring class hierarchies. FCA provides a formal framework for identifying groups of elements sharing sets of properties. It forms the clusters of objects having common attributes [15]. After applying FCA and forming a lattice of given class specifications, one can easily analyze the structure and can comment on various aspects such as code duplication, design patterns, relevant domain concepts, opportunities for refactoring and quality of source code etc. Although Formal Concept Analysis is powerful to distribute attributes in a class hierarchy, but is unable to deal with relational descriptions. Thus, we propose to apply Relational Concept Analysis to extend FCA.

Relational Concept Analysis, an extension of FCA, is designed to take into account entities described by binary attributes with relations linking them [8]. The contribution proposed in this paper is an approach using Formal Concept Analysis and Relational Concept Analysis to refactor class hierarchy design in reverse engineering, and evaluating the impact of class hierarchy with object oriented design metrics.

Design metrics play a vital role in helping developers to appreciate design aspects of software i.e. improve software quality and developer productivity [21]. The metrics for object oriented design focus on measurements that are applied to the class and design

characteristics for making changes that will reduce complexity and improve the continuing capability of the design. Nowadays, a quality engineer can choose from a large number of object–oriented metrics. Among them, we emphasize on object oriented metrics used include inheritance related measures, cohesion measures and coupling measures such as DIT, NOC, ANDC, ANIC, LCOM and CBO, etc to evaluate the impact of the class hierarchy design.

This paper is organized as follows. We present some related works in section 2. Then in section 3 and 4, we discuss the background theory and detail the framework we adopt for refactoring class hierarchy using FCA and RCA and present some object-oriented design metrics for evaluating the impact of class hierarchy design. We describe conclusion in section 5.

2. Related Work

FCA has been used in many research areas. Boussaidi [5] present an approach to detect design defects in source code using two techniques: Formal Concept Analysis (FCA) and metrics and detect antipatterns through the detection of the well known design defect Blob. In this work, they measure the size of the classes in terms of number of declared attributes (NAD) and methods (NMD) and use the LCOM5 metric to compute the cohesion of each class. Moha, Rezgui, and Gueheneuc proposed a novel approach for defect removal in object-oriented programs that combines the effectiveness of metrics with the theoretical strength of FCA. They used a model of the source code which is simpler to manipulate than the raw source code and therefore eases the subsequent activities of detection and correction. However, they did not describe how to measure with the metrics [13].

In [12], they applied an automated approach for suggesting defect-correcting refactoring using Relational Concept Analysis (RCA). The added value of RCA consists in exploiting the links between formal objects which abound in a software re-engineering context. They also validated this approach on instances of the Blob design defect taken from four different open-source programs. A large body of some work focused on problems related to hierarchy construction and reconstruction. Depending on the design goals and available specifications, several alternative hierarchy types are considered within this framework. A set of normal forms for class hierarchy design is described, all of them based on the FCA framework [15], [10].

The paper proposes an approach based on Formal Concept Analysis and one of its variants, Relational Concept Analysis, to refactor a use case diagram as a whole in order to make it clearer by introducing generalized actors and use cases to factorize relations [6]. In this paper [8], Falleri, Huchard and Nebut present a generic approach and tool using Model-Driven Engineering and Relational Concept Analysis to perform class model normalization. By translating the source data from different languages (UML, EMOF, Java,..) into the input FCA data format before applying FCA or RCA. Our proposed work seems similar with this work. However, our work evaluates the impact of the class hierarchy design with OOD metrics for considering maintainability.

This paper considers the inheritance metrics of F.T. Sheldon et al (2002) and Henderson Seller's (1996) for comparison with proposed inheritance metric suites [18]. We emphasize on this proposed inheritance metrics in our work.

3. Background Theory

3.1. Formal Concept Analysis

Formal Concept Analysis (FCA) is a strong tool which provides a framework for class hierarchy design and maintenance. FCA provides a formal framework for identifying groups of elements sharing sets of properties. An FCA engine requires contexts as inputs and generates a set of concepts. A concept is a group of elements and their shared properties. Relationships among concepts lead to a concept lattice.

Formal Context (K): Formal context is a simple mathematical structure used to define the fact that ’Object has an attribute’. A formal context K is a triple (G, M, I) where,

- G is a set of objects
- M is a set of attributes
- I is a (incidence) binary relation between G and M

Here, we consider classes as set of objects. Properties of objects are attributes, methods, and associations. Cross or value in particular cell shows that object and attribute are related.

Formal Concept (X, Y): Formal concept (X, Y) is an abstract idea which gives a view about objects, attributes and interaction between them. In formal context, there is set of objects consisting which are closely connected to the set of attributes. This pair forms a formal concept. X is called extent and Y is called intent of a concept (X, Y). This ordered set of concepts is called a concept lattice of context (G, M, I). Class hierarchy can be represented using well defined structure - Concept Lattice. There are three ways to form concept lattice of class hierarchy are attribute factored lattice form, method factored lattice form and associations factored lattice form.

3.2. Relational Concept Analysis

Relational Concept Analysis is an extension of FCA. It is designed to take into account entities described by binary attributes with relations linking them. In RCA, instead of having just one formal context, there is a set of formal contexts or a set of tables (a relational context family or RCF) such that some tables represent object-attribute relations and some tables capture object-object relations. Then these formal contexts are filled out with other contexts that show relations between entities coming from one context and entities coming from another context. More formally, a Relational Context Family (RCF) is a pair F = (K, L) where K is a set of formal contexts and L a set of relational contexts.

3.2. Object Oriented Design Metrics

In fact, object-oriented development requires not only a different approach to design and implementation, but also a different approach to software metrics. Object oriented design include attributes, methods, objects (classes), relationships and class hierarchies. Object oriented design metrics is an essential part of software environment. A set of object oriented metrics that can be used to measure the quality of an object-oriented design. They look at the quality of the way the system is being built. Design metrics can be divided into

• Traditional Metrics
• Object oriented Design Metrics.

3.3. Metrics for Class Diagrams

The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. For example, metrics proposed by Chidamber & Kemerer metrics (CK metrics, 1994), MOOD metrics, Lorenz and Kidd metrics etcs. CK metrics are the most popular among them. Another comprehensive set of metrics is MOOD metrics. Chidamber and Kemerer proposed six metrics are Weighted Method per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response for a Class (RFC) and Lack of Cohesion in Methods (LCOM).

The MOOD **(**Metrics for Object Oriented Design**)** metrics set refers to a basic structural mechanism of the OO paradigm as encapsulation, inheritance, polymorphisms, message-passing and are expressed as quotients. The set includes the following metrics are Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF) and Coupling Factor (CF).

Inheritance Metrics of F. T. Sheldon and Henderson Seller are Average Degree of Understandability Metric (AU), Average Degree of Modifiability Metric (AM), and Average Inheritance Depth (AID). Another inheritance Metrics of K. Rajnish, A.K. Choudhar, and A. M. Agrawal are Derive Base Ratio Metric (DBRM), Average Number of Direct Child

(ANDC) Metric, Average Number of Indirect Child (ANIC) Metric.

4 .Framework of the Proposed System

4.1. Applying FCA and RCA in Our Proposed System

The approach takes as input source code, encodes it into FCA (or RCA) contexts, generates the corresponding concept lattices, and finally produces as output the refactored class diagram. In our proposed system, we imagine that there are three steps of processes.

In first step, we extract specifications of classes from the source code by using FCA parser to get the formal context. Each class is converted into an object and the properties of the class are converted into attributes in the formal context, and the binary relation of the formal context is built according to attribute possession. Then, we feed formal context as input in the FCA engine. The FCA engine builds a concept lattice, according to the formal context. This concept lattice will contain concepts that represent the existing entities of the formal context, and new concepts that will lead to the creation of new classes. Then, FCA builds a class diagram according to the concept lattice and produces the factored class hierarchy as output.

In second step, the input of RCA is a set of tables (a relational context family or RCF). We construct Relational Context Family from the output of the first step. Then these formal contexts are filled out with other contexts that show relations between objects coming from one context and objects coming from another context. An iterative lattice construction is applied on the relational context family. The set of lattices produced after each step of the process is called a Concept Lattice Family (CLF). After building class diagram, RCA produces the refactored class hierarchy as output. Finally, we assess the impact of the output class hierarchy design with object-oriented design metrics especially with inheritance, cohesion, coupling metrics.

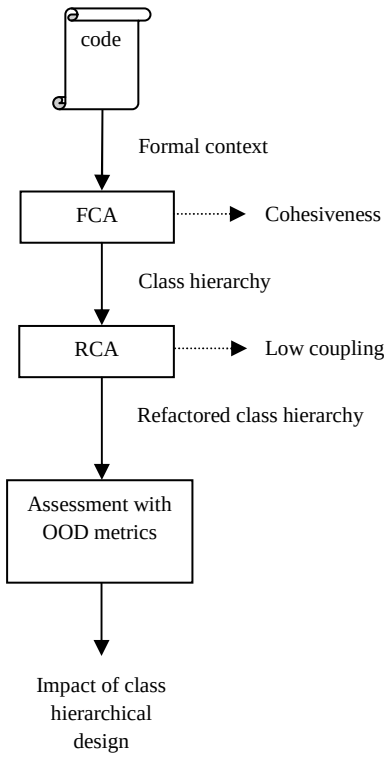

Figure.1. Process flow of the proposed system

4.4. Assessment with the object-oriented design metrics

The selected object-oriented design metrics are intended to be applied to the concepts of classes (cohesion), coupling, and inheritance for assessment of the impact of the resulted class hierarchy design.

4.4.1. Inheritance Metric: Inheritance is a type of relationship among classes that enables programmers to reuse previously defined objects including variables and operators.

Depth of Inheritance Tree (DIT): The DIT will be the maximum length from the node to the root of the tree. DIT is a measure of how many ancestor classes can potentially affect this class.

Number of Children (NOC): The number of children is the number of immediate subclasses subordinate to a class in the hierarchy.

Method Inheritance Factor (MIF): MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods for all classes.

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_a)} \quad \text{--------- (1)}$$

Attribute Inheritance Factor (AIF): AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes for all classes.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_a)} \quad \text{-------- (2)}$$

Average Degree of Understandability Metric (AU): The ease of understanding a program structure or a class inheritance structure.

$$AU = \sum_{i=1}^{t} (PRED(C_i) + 1)/t \quad \text{------- (3)}$$

Average Degree of Modifiability Metric (AM): The ease with which a change or changes can be made to a program structure or a class inheritance structure.

$$AM = AU + \sum_{i=1}^{t} (SUCC(C_i)/2)/t \quad \text{------ (4)}$$

Average Inheritance Depth (AID):

$$AID = \sum \quad (\textit{depth of each class / number of classes}) \text{ --- (5)}$$

Derive Base Ratio Metric (DBRM): DBRM is the ratio of the total derived classes to the total base classes in the class inheritance tree. DBRM measures how many derived classes which directly or indirectly affect the ancestor classes.

$$DBMR = \sum_{i=0}^{N} TD(C_i) \Big/ \sum_{i=0}^{N} TB(C_i) \quad \text{---- (6)}$$

Average Number of Direct Child (ANDC) Metric: ANDC metric is the ratio of the total number of immediate child to the total number of classes in the inheritance tree. NDC metric measures how many immediate subclasses are going to inherit the properties of classes.

$$ANDC = \left(\sum_{i=0}^{N} TDC(C_i) \right) \Big/ N \quad \text{------- (7)}$$

Average Number of Indirect Child (ANIC) Metric: ANIC metric is the ratio of the total number of indirect child to the total number of classes in the inheritance tree. This metric gives how many ancestors' classes potentially affect the subclasses.

$$ANIC = \left(\sum_{i=0}^{N} TIC(C_i) \right) \Big/ N \quad \text{------- (8)}$$

4.4.2. Cohesion Metric:

Lack of Cohesion in Methods (LCOM) is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

4.4.3. Coupling Metric:

Coupling between objects (CBO) CBO is a count of the number of other classes to which a class is coupled. The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult.

5. Conclusion

Nowadays, FCA has been used in wide range of areas. In software engineering, many tasks become very simple and less manual by the use of FCA. In this paper, an attempt is to present the refactoring of class hierarchy design by using FCA and RCA in reverse engineering and to

apply different inheritance metrics for impact assessment of the class hierarchy. This study also focuses on object oriented design metrics (cohesion and coupling) that can be worn to measure the quality of an object oriented design. In this work, the goal is to find the efficient class hierarchy design for maintainability of code in reverse engineering. We will analyze several java Open Source Projects. Nevertheless, in the one hand, the FCA and RCA technique is built upon the metrics one results for choosing the classes to analyze. In the other hand, the FCA and RCA techniques give some information like the cohesive and low coupling subsets of a large class that the metrics technique do not. Using the results obtained with the FCA and RCA techniques, we are able to resolve design problems and able to assess the impact with object-oriented design metrics.

References

[1] G. Arévalo, *"High-Level Views in Object-Oriented Systems Using Formal Concept Analysis*", 2004.

[2] G. Arévalo, S.Ducasse, O. Nierstrasz, "*Discovering Unanticipated Dependency Schemas in Class Hierarchies*", 2010.

[3] S. Azhar, S. Samia, A. Anam, A. Moein, "*Detection of Problems in Class Hierarchies and Their Correction through Formal Concept*", in ICSM, 2010.

[4] G. E. Boussaidi, D. Huynh, N. Moha, *"Detection of Design Defects:Formal Concept Analysis and Metrics",* 2005.

[5] J. Bansiya, C. G. Davis, *"A Hierarchical Model for Object-Oriented Design quality Assessment",* IEEE, 2002.

[6] X. Dolques, M. Huchard, C. Nebut, and P. Reitz, "*Fixing generalization defects in uml use case diagrams*", 2010.

[7] J. Falleri, G. Arévalo, M. Huchard, and C. Nebut, "*Use of Model Driven Engineering in Building Generic FCA/RCA Tools."*

[8] J. Falleri, M. Huchard, C. Nebut, "*A generic approach for class model normalization*."

[9] M. Genero, M. Piattini and C. Calero, *"Early measures for UML class diagrams", L' Objet. Volume 6, No. 4, 2000.*

[10] R. Godin, P. Valtchev, "*Formal concept analysis-based class hierarchy design in object-oriented software development."*

[11] M. L. Lee, *"Change Impact Analysis of Object-oriented Software", 1998.*

[12] N. Moha, A. Napoli, M. Rouane-Hacene, P. Valtchev, and Y. E. Gueheneuc, "*Refactorings of Design Defects using Relational Concept Analysis."*

[13] N. Moha, J. Rezgui, Y. Gueheneuc, P. Valtchev, and G. E. Boussaidi, "*Using FCA to Suggest Refactorings to Correct Design Defects."*

[14] C. Neelamegam, Dr. M. Punithavalli, *"A Survey - Object Oriented Quality Metrics*", in Global Journal of Computer Science and Technology, p 183.

[15] P. S. Patil, *"Applying Formal Concept Analysis to Object Oriented Design and Refactoring."*

[16] L. H. Rosenberg, L. E. Hyatt, "*Software Quality Metrics for Object-Oriented Environments."*

[17] K. Rajnish, V. Bhattacherjee "*Class Inheritance Metrics- An Analytical and Empirical Approach"*, 2007.

[18] K. Rajnish, A. K. Choudhary and A. M. Agrawal, "*Inheritence Metrics for Object-oriented Design"*, in International Journal of Computer Science & Information Technology (IJCSIT), Vol 2, No 6, 2010.

[19] M. Rouane-Hacene, M. Huchard, A. Napoli, and P. Valtchev, "*Using Formal Concept Analysis for discovering knowledge patterns*", in CLA'10, 2010.

[20] A. Shaik, C. R. K. Reddy, B. Manda, Prakashini. C, Deepthi. K, *"An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems"*, in Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 1 (2), 2010.

[21] H. A. Sahraoui, R. Godin, T. Miceli, "*Can Metrics Help Bridging the Gap between the Improvement of OO Design Quality and Its Automation."*

[22] G. Snelting, "*Concept Lattices in Software Analysis."*

[23] M. Streckenbach, G. Snelting," *Refactoring Class Hierarchies with KABA."*