Applying Design Patterns to Redesigning of an Existing Software and its Evaluation

Masuda, Gou
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University : Graduate Student

Sakamoto, Norihiro
Department of Medical Informatics, Kyushu University Hospital

Ushijima, Kazuo
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.15017/1508398

出版情報：九州大学大学院システム情報科学紀要. 4 (2), pp.113-118, 1999-09-24. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学研究科報告
第4巻 第2号 平成11年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.4, No.2, September 1999

Applying Design Patterns to Redesigning of an Existing Software and its Evaluation

Gou MASUDA*, Norihiro SAKAMOTO** and Kazuo USHIJIMA***

(Received June 21, 1999)

Abstract: Since design patterns are now well known as one of the most effective techniques for an object-oriented software design and implementation, a lot of programs have been developed using design patterns. However, there has been few researches on quantitative evaluation of the effectiveness of applying design patterns to software development. In this paper, we describe a case study on redesigning of an existing decision tree learning system which is a data mining tools based on machine learning technology. Moreover we aim to quantitatively evaluate the effectiveness of applying design patterns to the redesigning. The C&K metrics suite is used for the evaluation. We collect C&K metrics values for two releases of the decision tree learning system. One is a prototype release designed without using design patterns while the other redesigned using design patterns. We conduct the Mann-Whitny U-test, one of the nonparametric statistics for testing hypotheses about whether two sample values differ. As a result, we find significant differences between the C&K metrics values of the two releases. Finally we discuss the relationship between the design patterns and the C&K metrics suite. The result of the discussion suggests that new metrics should be devised for the evaluation of the effectiveness of applying design patterns.

Keywords: Object-oriented design, Design patterns, Object-oriented software metrics

1. Introduction

Software developments have become increasingly large, diverse and complex in recent years. Extending the life-span of such software has become a challenging problem. Object-oriented technology offers one solution. Improvement of the modularity and readability of software helps us to understand the structure of the software and to maintain, modify and extend it. Well-defined object-oriented designs make software flexible and extensible. However, achieving well-defined object-oriented designs is a skilled task that requires both time and experience.

Design patterns[8] present one solution to this problem. They are a good collection of successful object-oriented designs which appear repeatedly and have worked well as solutions to past design problems. Since the design patterns are based on the designs succeeded in the past, applying design patterns to redesigning of existing software provides a good foresight of the flexibility and future extensibility of them. A large number of researchers have reported results of applying design patterns to object-oriented software designs[12),18),19)]. However only a small amount of work on evaluation of the quantitative effectiveness has been reported. In this paper, we aim to redesign an existing software using design patterns and aim to evaluate a quantitative effectiveness of them.

The remainder of this paper is organized as follows. Section 2 describes a case study on redesigning of an existing decision tree learning system[3),14),16),17)] using design patterns. Section 3 experimentally evaluates the quantitative effectiveness of applying design patterns to redesigning of object-oriented software using the C&K metrics[6)]. The C&K metrics suite is one of the most popular object-oriented software metrics proposed by Chidamber and Kemerer. Section 4 discusses the relationship between C&K metrics and design patterns in detail. Section 5 concludes this paper.

2. A Case Study — Applying Design Patterns to Redesigning of a Decision Tree Learning System

2.1 Decision Tree Learning System

Decision tree learning systems construct a classifier from given cases as a form of decision tree. The systems are required to have great flexibility and extensibility in the context of KDD(Knowledge Discovery in Databases)[7)]. KDD aims to automatically analyze large real world databases and extract nov-

* Department of Computer Science and Communication Engineering, Graduate Student

** Department of Medical Informatics, Kyushu University Hospital

*** Department of Computer Science and Communication Engineering

el, useful and interesting knowledge from the data. Decision tree learning systems are one of the data mining methods applied in the KDD process.

A number of heuristic methods and strategies have been proposed for efficiency and accuracy in decision tree learning. In general there is no single best method or strategy for all knowledge discovery tasks. Users therefore have to select an appropriate method for their specific task. However there are no clear theoretical metrics for selecting an appropriate method under a given circumstance. Consequently users have to apply a range of methods to their own data and compare results to determine which provides the best fit. Moreover, users often have to modify or extend these methods. In existing decision tree learning systems this is a difficult task because current systems emphasize neither flexibility nor extensibility. Because of this general requirement that decision tree learning systems be flexible and extensible, we redesign an existing decision tree learning systems using design patterns.

2.2 Design Patterns

Design patterns provide one solution for designing reusable object-oriented software. Each pattern systematically names, explains and evaluates an important and recurring design in object-oriented software designs. Design patterns have the following advantages:

- The patterns provide reusable, well-defined object-oriented designs.
- The patterns define the responsibilities and relationships among objects in a design problem. They represent good documentation of the software easing comprehension and future maintenance.
- Each pattern name constitutes a vocabulary for designers to describe design problems and solutions.

2.3 Redesign using Design Patterns

2.3.1 Hot-spot based approach

We redesign an existing decision tree learning system developed by our research group using object-oriented technology[11]. The following approach is employed in redesigning the system.

First, we identify the parts of the system which we expected to require future modification or extension. We use *hot-spot*[15] to represent such parts of the system. In the book 15), *hot-spot* is used in the context of an application framework. We use this term to refer to the parts of the system that we expect to be modified or extended. We examine several decision tree learning systems[3),14),16),17)] and several methods for decision tree learning[4),10),16)] in order to identify the hot-spots of the system.

Once we identify the hot-spots, we choose appropriate design patterns which can be applied to each hot-spot. When choosing design patterns, we carefully consider their applicability and consequences described in each design pattern.

2.3.2 Hot-spots of the system

In this study we identify the following eight hot-spots of the system.

(1) "Data set creation" handling: A variety of data sources such as relational databases and formatted text files are anticipated in the practical application. The system needs to support various ways to create a data set.

(2) "Attribute type" handling: The system needs to support any types of attributes in addition to the ordinary continuous and discrete attribute.

(3) "Decision tree structure" handling: A decision tree forms a hierarchical structure. The system needs to deal with the elements in a decision tree uniformly without making any distinction between primitive elements and groups of elements.

(4) "Test method" handling: The system needs to support any kind of test that can be applied to any type of attribute.

(5) "Test selection method" handling: The system needs to handle various methods or criteria and make them exchangeable without affecting the rest of the system.

(6) "Noise data handling method" handling: The system needs to support a variety of noise data handling methods.

(7) "Decision tree pruning method" handling: The system needs to handle various tree pruning methods and make them exchangeable without affecting the rest of the system.

(8) "Decision tree evaluation method" handling: Induced decision tree need to be evaluated from various points of view. The system needs to provide such a variety of evaluation methods.

2.3.3 Applying design patterns

We decide design patterns that we apply to each hot-spot described above. In coming to this decision we consider the requirements for each hot-spot and the applicability and consequences of the selected design patterns. We give a solution and the consequences for only the hot-spot (1) as an example of how we solve the requirements using design patterns. In 12), we have described our solutions and

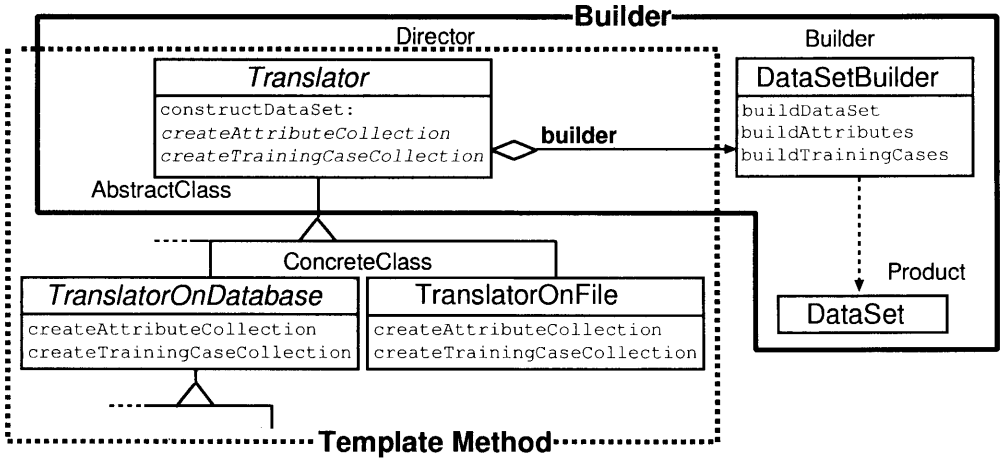

Fig.1 Application of the **Builder** and **Template Method** patterns to the data set construction.

the consequences for all the hot-spots.

(1) "Data set creation" handling

Requirement:

- Create a data set independent of the data source.

Solutions: In order to encapsulate the creation process of a data set, we apply the **Builder** pattern to data set creation. The **Builder** pattern separates the construction of a complex object from its representation. **Fig. 1** illustrates how the **Builder** pattern is applied to this hot-spot. In this case, class Translator behaves as a Director object. It constructs a data set object gradually using interfaces defined in class DataSetBuilder. A concrete DataSetBuilder is the only one which knows the internal structure of the data set. This allows the different representation of the data set.

Next, we apply the **Template Method** pattern in order to deal with data set creation from the different data sources. The **Template Method** pattern is composed of an AbstractClass and several ConcreteClasses. The AbstractClass has a template method that describes the skeleton of an algorithm. The ConcreteClasses implement the hook methods that describe the variable parts of the algorithm depending on the situation. The class Translator in **Fig. 1** has a template method, `constructDataSet:` which defines a skeleton of an algorithm for the data set translation operation from each data source. The `createAttributeCollection` method is one of the hook methods which defines a translation operation dependent on a particular data source.

The class TranslatorOnFile which supports data creation from ASCII text files has a DataFileParser object in order to parse the data files. We apply the **Strategy** pattern in order to deal with a variety of file formats. This pattern encapsulates the algorithms and makes them interchangeable. In this case the **Strategy** pattern encapsulates a parsing algorithm for a data file in a particular format.

Consequences:

- The **Builder** pattern makes it possible to change the internal representation of a data set object.
- The **Template Method** pattern allows reuse of the skeleton of the data creation algorithm. All the developer has to do on changing the data source is describe the minimum required hook methods.
- The **Strategy** pattern offers a way to interchange a variety of file parsing algorithm.

We implemented a decision tree learning system with the class design described above. We used Smalltalk for a programming language. The fact it is a pure object-oriented programming language allowed us to reflect the design using patterns directly in the implementation. We applied a total number of 15 patterns to the hot-spots and used over 80 classes in order to implement the system. **Table 1** summarizes each hot-spot and the design patterns applied to that.

3. Evaluation of Applying Design Patterns

In this section we quantitatively evaluate the effectiveness of applying design patterns from the point of view of the complexity of software. The C&K metrics suite is used for the evaluation. It defines six metrics for object-oriented design. It has been used in various application domains and its effectiveness has been proven[2),9)].

Table 1 Design patterns applied to each hot-spot

Hot-spot	Design patterns
(1) "Data set creation" handling	**Builder, Template Method, Strategy**
(2) "Attribute type" handling	**Factory Method, Abstract Factory**
(3) "Decision tree structure" handling	**Composite, Visitor, Abstract Factory**
(4) "Test method" handling	**Command**
(5) "Test selection method" handling	**Template Method**
(6) "Noise data handling method" handling	**Strategy, Abstract Factory**
(7) "Decision tree pruning method" handling	**Visitor, Template Method**
(8) "Decision tree evaluation method" handling	**Strategy**

Weighted Methods per Class(WMC) is defined as the sum of the complexities of all methods of a class. The larger the WMC value for a class, the more complicated and expensive it is to maintain the class. In the C&K metrics, the complexity of a method is not defined specifically in order to allow for the most general application of this metric. In this study, we adopt the approach described in 1). That is, the complexity of a method is defined as the ratio of the weight of the method to 128 bytes, which is the standard weight of a method given in 1). "Weight of a method" is defined as the byte size of all objects included in the method.

Depth of the Inheritance Tree(DIT) is defined as the maximum length from the node to the root of the inheritance tree of a class. The larger the DIT value for a class, the greater the number of variables and methods it is likely to inherit, and therefore the more difficult it is to predict its behavior.

Number Of Children(NOC) is defined as the number of immediate subclasses. The larger the NOC value for a class, the greater influence the class has, and therefore the more testing of the methods in the class likely to be required.

Coupling Between Objects(CBO)] is defined as the number of classes to which a class is coupled via a method or attribute use. The larger the number of couplings, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.

Response For a Class(RFC) is defined as the number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class. The larger the number of RFC value for a class, the greater the complexity of the class, and therefore the more difficult maintenance.

Lack of COhesion in Methods(LCOM) is the number of pairs of methods without shared instance variables, minus the number of pairs of methods with shared instance variables. The metric is set to 0 whenever the above subtraction is negative. Low cohesion increases complexity, and therefore maintenance is more difficult.

Table 2 Results of C&K metrics

Metric	Release 1			Release 2		
	Min	Median	Max	Min	Median	Max
WMC	0.56	9.53	30.56	0	3.75	44.59
DIT	1	1	5	1	2	5
NOC	0	0	3	0	0	5
CBO	0	1	8	0	2	20
RFC	2	28	83	0	19	143
LCOM	0	21	187	0	1	1469

We collect the C&K metrics values from the decision tree learning systems before redesigning (Release1) and after redesigning (Release2). We use a metrics measurement tool developed by our research group. It is based on OOM[1)] which is a metrics measurement tool for Smalltalk. The results of collecting the metrics values appear in **Table 2**.

In order to ascertain whether the results have statistically significant difference, that is, whether the design patterns affect the metrics values, we conduct a two-tailed Mann-Whitney U test with significance at the 5% levels. It is one of the non-parametric statistics for testing hypotheses whether two samples differ. The result of this statistical test shows that 5% significant differences exist in WMC and RFC. It may be not enough to discuss generalities of the effectiveness of applying design patterns from only this case study. However we believe that we can show a quantitative evidence of the effectiveness that redesigning using design patterns improves the complexity of the software.

4. Discussion

4.1 Analysis on Relationship between Design Patterns and C&K Metrics

The experimental evaluation described in Section 3.2 shows statistically significant differences are observed in metrics values of WMC and RFC. However the experiment also shows that specific design patterns tend to make a particular metric value worse.

For example, the max values of RFC and WMC in Release2 are higher than those in Release1 though statistical differences are observed. We therefore analyze the measurement of individual metrics and the relationship between design patterns and C&K metrics in detail. In this paper we give only the analysis on WMC, CBO, RFC and LCOM. Complete analysis was described in 13).

4.1.1 Analysis on WMC

Some design patterns make object granularity small. In particular, class ConcreteCommand in the **Command** patterns, class ConcreteStrategy in the **Strategy** pattern and class ConcreteFactory in the **Factory Method** pattern tend to become a small class with a low WMC value. Since we applied these design patterns to a large number of places, the statistical difference was observed between the two releases.

Further analysis on WMC shows that the WMC values for the classes which play a central role tend to be high. Class DTLSDecisionTreeManager (WMC=44.59) is one of the example. It corresponds to class ConcreteMediator in the **Mediator** pattern. The **Mediator** pattern makes coupling between colleagues loose. However on the other hand, it centralizes control in class ConcreteMediator. As a result, the complexity of class ConcreteMediator increases.

4.1.2 Analysis on CBO

The following tendency on CBO value was found in our experiment: class ConcreteBuilder in the **Builder** pattern, class ConcreteCreator in the **Factory Method** pattern, class ConcreteFactory in the **Abstract Factory** pattern and class Mediator in the **Mediator** pattern have high CBO value. In the creational patterns such as **Builder**, **Factory Method** and **Abstract Factory**, objects taking the responsibility for object creation need to know the class name of products. Because of this, those classes tend to have high CBO values. Since design patterns indicate which classes have such responsibility for object creation, it is not so difficult to maintain those classes, contrary to C&K's predictions. Furthermore, the class playing a central role in the application such as class Mediator in the **Mediator** pattern need to know about other colleague classes. The CBO value therefore tends to be high.

4.1.3 Analysis on RFC

The RFC value of a class tends to be high if the class invokes a large number of methods of other classes. As in the case of the WMC metric and CBO metric, class DTLSDecisionTreeManager (RFC=143), (which is correspondent to class ConcreteMediator in the **Mediator** pattern), have high RFC values. We believe the reason is that the ConcreteMediator tends to communicate with many other colleague objects via message passing.

Class DTLSTreePruningVisitor (RFC=71) and class DTLSTreeGeneratingVisitor (RFC=54) also have high RFC values. They are correspondent to class ConcreteVistor in the **Visitor** pattern. In the **Visitor** pattern, class ConcreteVisitor often invokes methods defined in the ConcreteElement object. The RFC value of class ConcreteVisitor tends to be high as a result.

4.1.4 Analysis on LCOM

A high LCOM value indicates disparateness in the functionality provided by the class. Class DTLSDecisionTreeManager (LCOM=1469) has high LCOM values. It is correspondent to class ConcreteMediator in the **Mediator** pattern. Class ConcreteMediator receives a lot of different requests in place of its colleague objects. As a result, it has various methods that are unrelated to each other. The LCOM values therefore tend to be high.

4.2 Future Directions of Devising Metrics for Applying Design Patterns

Our detailed analysis of individual measurements on the C&K metrics revealed that particular design patterns tend to make the specific metric values worse. However as discussed above, worsening metric values in some design patterns is due to the characteristics of those design patterns. It therefore does not necessarily mean that those design patterns have disadvantages as object-oriented design. Instead it suggests that several C&K metrics are not appropriate measures of the quality of designs using design patterns. In this section we describe our future directions of devising new metrics for evaluating the effectiveness of applying design patterns.

(1) Modified C&K metrics: For some C&K metrics, we did not observe statistically significant difference in our evaluation. However it does not mean that these metrics are completely useless to the evaluation of applying design patterns. We can therefore devise some modified C&K metrics in order to better reflect the characteristics of design patterns. For example, it is a modified CBO metric that does not count couplings between objects which are part of a collaboration. Some design patterns encapsulate coupling by using dynamically bound references. Though using such design patterns increase the number of coupling, it is not a bad design but

good one.

(2) Run-time metrics: Static metrics such as C&K metrics can only capture the static aspects of objects. We think it is important to capture the dynamic aspects of objects in order to reflect the characteristics of design patterns. Take the case of the **Flyweight** pattern which uses sharing to support large numbers of fine-grained objects efficiently. When applying the **Flyweight** pattern, several run-time aspects are important such as the total number of objects which are actually created. By capturing the aspects using run-time metrics, we can provide some useful guidelines that indicate whether or not **Flyweight** pattern should be used in a specific situation.

(3) Metrics for design process: The C&K metrics measure the quality of software product rather than software design process. However the effectiveness of applying design patterns is also involved in the design process. For instance, using design patterns makes it easy to get an appropriate solution to a design problem. It therefore shortens the period of design process. However the software product would be almost the same, no matter whether design patterns are applied or not. In this case metrics for only the software product can not capture the effectiveness of applying design patterns in the design process.

5. Conclusion

In this paper we have described a case study on redesigning of existing software using design patterns. We have also tried making quantitative evaluation of effectiveness of applying design patterns. Through this measurement and analysis, we are convinced of the need to devise new object-oriented metrics for evaluating the effectiveness of applying design patterns. For this purpose we presented three directions of such new metrics. They are metrics for the goodness of applying design patterns rather than ones for evaluation of effectiveness of it. They can be used as a guideline for applying design patterns in a specific circumstance. For example, putting to proper use the patterns classified into creational patterns of GoF book[8] is difficult for developers who are unfamiliar with the design patterns. We believe these metrics make a contribution as a part of the guideline for use of design patterns such as a system of patterns[5].

References

1) Aoki, A. *Smalltalk Idiom*, SRC, 1997. (In Japanese)

2) Basili, V. R., Briand, L. and Melo, W. L."A Validation of Object-Oriented Design Metrics as Quality Indicators," Technical Report UMIACS-TR-95-40, Univ. of Maryland, 1995.

3) Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. "Classification and Regression Trees," Belmont, CA: Wadsworth, 1984.

4) Brodley, C. E. "Automatic Selection of Split Criterion during Tree Growing Based on Node Location," Proc. of the Twelfth International Machine Learning Conference, Tahoe Cith, CA, 1995.

5) Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Ltd, 1996.

6) Chidamber, S. R. and Kemerer, C. F. " A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476–493, 1994.

7) Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, H. *Advances in Knowledge discovery and data mining*, AAAI/MIT Press, 1996.

8) Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns* , Addison-Wesley Publishing Company, 1995.

9) Harrison, R. and Nithi, R. "An Empirical Evaluation Of Object-Oriented Design Metrics," OOPSLA'96 Workshop: OO Product Metrics, 1996.

10) Ho, T. and Nguyen, T. "Evaluation of Attribute Selection Measures in Decision Tree Induction," Proc. 9th Int. Conf. on IEA/AIE, pp. 413-418, 1996.

11) Masuda, G., Sakamoto, N. and Ushijima, K. "A Practical Object-Oriented Concept Learning System in Clinical Medicine," Proc. 9th Int. Conf. on IEA/AIE, pp. 449-454, 1996.

12) Masuda, G., Sakamoto, N. and Ushijima, K. "Applying Design Patterns to Decision Tree Learning System," Proc. of ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering, pp. 111-120, 1998.

13) Masuda, G., Sakamoto, N. and Ushijima, K. "Evaluation and Analysis of Applying Design Patterns," Proceedings of International Workshop on the Principles of Software Evolution, pp. 135-139, 1999.

14) Murthy, S. K. and Kasif, S., and Salzberg S. "A System for Induction of Oblique Decision Trees," Journal of Artificial Intelligence Research 2 1-32, 1994.

15) Pree, W. *Design Patterns for Object-Oriented Software Development*, ACM Press, 1995.

16) Quinlan, J.R. "Induction of Decision Trees," *Machine Learning, 1,* pp. 81-106, 1986.

17) Quinlan, J. R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

18) Schmidt, D. C. and Stephenson, P. "Experience Using Design Patterns to Evolve Communication Software Across Diverse OS Platforms," In Proc. of the 9th E-COOP, 1995.

19) Schmidt, D.C. "A Family of Design Patterns for Flexibly Configuring Network Services in Distributed Systems," In Proc. of the Int. Conf. on Configurable Distributed Systems, 1996.