A Methodology for Constructing a Design Handbook for Object Oriented Systems

D Janaki Ram, K N Anantha Raman, K N Guruprasad and Suchitra Raman
Distributed and Object Systems Group
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
India
djram@iitm.ernet.in

Abstract

Design reuse in mature engineering disciplines is helped by the presence of Design Handbooks. These handbooks help in organizing design knowledge in a format suitable for systematic reuse. This paper attempts to provide a methodology to construct a design handbook to help reuse in building object oriented systems. The handbook attempts to provide a formal basis for design decisions in the context of object oriented software development. A specific case of choosing a design pattern is illustrated to demonstrate the practical utility of the methodology proposed.

Keywords - Design problem, design pattern, design handbook, metrics, coding effort, static adaptability, dynamic adaptability, performance, correlation table, selection function

1 Introduction

Most engineering disciplines provide mechanisms to capture and organize existing design knowledge which can be used while building new systems. Handbooks and manuals are often the desired form of organization of this knowledge. Systematic presentation of design in the form of handbooks enables designers to make effective design decisions for a large number of routine design problems.

The construction of a handbook requires identification of often used components and codification of properties of these components in terms of a few key parameters. The values of these parameters form the basis of choosing appropriate design components in a given context. For example, in electronics engineering, a transistor amplifier handbook gives the general characteristics of a transistor viz. current gain, voltage gain, power gain, input resistance, output resistance etc. in numbers. While designing a transistor amplifier, an electronics engineer consults this handbook for selecting a suitable transistor.

Design patterns can be thought of as components of an object oriented design. However, they fail to provide a suitable representation for handbooks. To develop a handbook on design patterns, it is essential to identify the important characteristics of design patterns and provide a method to quantify these characteristics.

This paper identifies four important characteristics of a design pattern viz. Coding effort, Static adaptability, Dynamic adaptability and Performance. We propose a set of metrics which can be correlated to these characteristics. This correlation helps in quantifying these characteristics and constructing the design handbook. This design handbook is useful when there exists more than one design pattern applicable in a given context and the design decision involves choosing one among the applicable set. The application of the design handbook for the case of Adapter patterns has been demonstrated.

The rest of the paper is organized as follows. Section 2 defines some important terms. Section 3 describes the proposed metrics. Section 4 illustrates how these metrics aid the decision making process and demonstrates the construction of a handbook. Section 5 concludes and discusses the contribution of this paper.

2 Terminology

A few terms are introduced in this section.

Coding effort of a pattern is the amount of coding that must be done for implementing the pattern in a programming language.

Static adaptability is a measure of easiness with which a pattern can be adapted to a particular context at the time of coding.

Dynamic adaptability reflects the ease with which the behavior of a pattern can be modified/adapted at runtime.

Performance is a measure of the speed with which a pattern delivers services expected of it.

Hook method is one which is declared in a class and defined in the subclass.

Template method is that which calls at least one hook method.

Rigid method is one which is declared and defined in the same class.

Root class is one whose task is to define a common interface for its subclasses. It has at least one hook method and it cannot be instantiated.

Concrete class is that which has only rigid methods.

Indirection link exists between two classes when a method of one class calls a method of another class through a reference variable.

Inheritance link exists between two classes when one class is subclass of another.

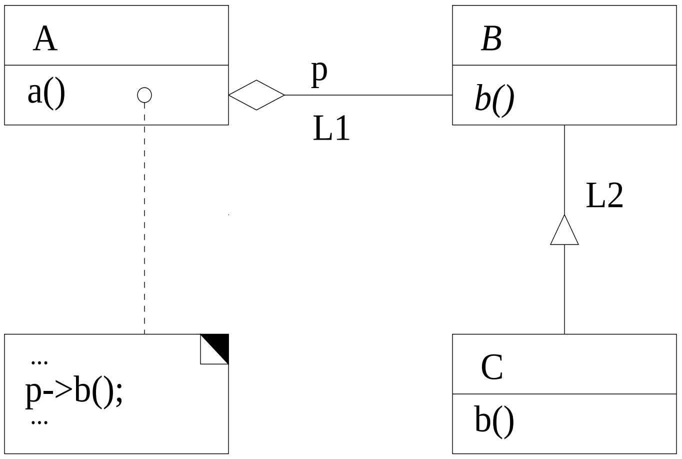
Number of activations of a link for a method is number of times the link is "used" when the method is called. It is applicable to indirection and inheritance links. An example is given in Figure 1.

3 Metrics

In the following paragraphs, a set of metrics is proposed to provide quantitative measure of the four important characteristics of a pattern viz. coding effort, static adaptability, dynamic adaptability and performance. While applying the metrics, the children classes are not considered as part of the pattern. The value of each metric as applied to Class Adapter(Figure 2) is also given. The correlation between the metrics and the characteristics is summarized in Table I.

1. Total Methods : TM

   The Total number of Methods that have been declared in the pattern.



Indirection link = L1

Inheritance link = L2

For method a() of class A :

No. of activations of indirection link = 1

No. of activations of inheritance link = 1

Figure 1: Example

As the value of TM goes up, more number of methods must be declared and defined during the implementation phase. This increases the coding effort.
Class Adapter pattern has two methods(Request() and SpecificRequest()). TM for Class Adapter pattern is 2.

2. Percent Hook Method : PHM

   Percentage of Hook Methods in TM.

   Since a hook method is declared in the root class and defined only in the subclass, behavior of a pattern with a high percentage of hook methods can be easily adapted to our requirements by suitable definitions during sub classing of root classes. Thus, static adaptability increases with increase in PHM.
   Class Adapter pattern has a hook method(Request()). PHM for the Class Adapter pattern is 50.

3. Percent Template Method : PTM

   Percentage of Template Methods in TM.

When a template method is called, it in turn calls hook method(s). In a pattern with high PTM, we may have many methods that call hook method(s). This results in decreased performance. Thus, performance comes down with increase in PTM.
Class Adapter pattern has no template methods. PTM for the Class Adapter pattern is 0.

4. Percent Rigid Method : PRM

   Percentage of Rigid Methods in TM.

   The declaration and definition of a rigid method is done in the same class and it is not changed. High PRM for a pattern implies that a high percentage of behavior of the pattern is fixed and cannot be adapted. Since the rigid methods do not call other methods, a pattern with high PRM gives good performance.
   Class Adapter pattern has a rigid method(SpecificRequest()). PRM for the Class Adapter pattern is 50.

5. Percent Client Called Methods : PCCM

   Percentage of the total number of methods which the client calls directly.

   When PCCM of a pattern is low, most of the methods are hidden from the client. This improves static adaptability. For example, two methods which are not called directly by the client may be merged without the client being aware of it. Similarly, a method not called directly by the client can be split into two methods under suitable circumstances. When PCCM is low, there exists a high percentage of methods which are called by the client indirectly, through client called methods. This results in poor performance. Thus, high PCCM implies low static adaptability and high performance.
   In the Class Adapter pattern, Request() is the only method which is called by the client. PCCM for the Class Adapter pattern is 50.

6. Percentage of Methods with Dynamically Determined Number of Link Activations : PMDDNLA

   Percentage of methods called directly by the client for which the number of activations of an inheritance or indirection link can be determined only at runtime.

   A nonzero value for PMDDNLA exists when the pattern has at least one root class with a reference to itself or a subclass which has a reference to its root class. Every

method which contributes to PMDDNLA may result in a number of indirection link activations or indirection and inheritance link activations at runtime. Each of these activations is an opportunity to change/adapt the behavior at runtime. However, every activation increases the response time and performance comes down. Thus, high PMDDNLA implies high dynamic adaptability and low performance.
In the Class Adapter pattern, there is no class which has either a reference to itself or to its root class. PMDDNLA for this pattern is 0.

Table I
Correlation table

No.	Metric	Coding Effort	Static Adaptability	Dynamic Adaptability	Performance
1	TM	+	o	o	o
2	PHM	o	+	o	o
3	PTM	o	o	o	-
4	PRM	o	-	o	+
5	PCCM	o	-	o	+
6	PMDDNLA	o	o	+	-
7	ASDIDAPM	o	o	+	-
8	ASDINAPM	o	+	o	o
9	TC	+	o	o	o
10	PRC	o	+	o	o
11	NROP	+	o	o	o

+ : Positive Correlation
- : Negative Correlation
o : Little or no Correlation

7. Average Statically Determined Indirection Activations per Method : ASDIDAPM

   It is the sum of total number of activations of all indirection links for each method in the client called methods divided by number of client called methods. Methods for which number of activations of some indirection link can be determined only at runtime are not considered.

   An indirection link activation is an opportunity to change the behavior of the pattern at runtime. But every indirection link activation increases the response time.

Thus, high ASDIDAPM means high dynamic adaptability and low performance. There is no indirection link in the Class Adapter pattern. Hence ASDIDAPM for this pattern is 0.

8. Average Statically Determined Inheritance Activations Per Method : ASDINAPM

   It is the sum of total number of activations of all inheritance links for each method in the client called methods divided by number of client called methods. Methods for which number of activations of some inheritance link can be determined only at runtime are not considered.

   Every inheritance link activation corresponds to an opportunity to statically adapt (by sub classing a root class) the pattern during implementation. Thus, high ASDINAPM implies high static adaptability.
   There are two inheritance links which are active for each invocation of Request() method in the Class Adapter pattern. ASDINAPM for this pattern is 2.

9. Total Classes : TC

   Total number of Classes in the pattern.

   If TC is high, many classes have to be declared and defined. This increases the coding effort.
   There are two classes(Target and Adaptee) in the Class Adapter pattern. TC for Class Adapter pattern is 2.

10. Percent Root Classes : PRC

    Percentage of Root Classes in TC.

    During implementation every root class can be adapted to the requirements by proper sub classing. Thus, high PRC implies high static adaptability.
    Target is the only root class in the Class Adapter pattern. PRC for Class Adapter pattern is 50.

11. Number of Root Classes Outside Pattern : NROP

    Number of Root classes Outside the Pattern which must be declared and defined for using the pattern.

    When NROP is high, many classes have to be declared and defined, though they are not part of the pattern. This implies high coding effort.
    No root classes outside the pattern are required for the working of the Class Adapter pattern. NROP for Class Adapter pattern is 0.

4 Design Handbook

This section explains in detail, the construction and usage of the design handbook.

4.1 Handbook Construction

The construction of the handbook is demonstrated with the following example:

Assume a handbook which has information about only two patterns - Class Adapter(Figure 2) and Object Adapter(Figure 3). The results of applying the metrics to these patterns are given in Table II.

Table II
Metric values for Adapter patterns

No.	Metric	Class Adapter	Object Adapter
1	TM	2	2
2	PHM	50	50
3	PTM	0	0
4	PRM	50	50
5	PCCM	50	50
6	PMDDNLA	0	0
7	ASDIDAPM	0	1
8	ASDINAPM	2	1
9	TC	2	2
10	PRC	50	50
11	NROP	0	0

Normalization of these values is done by dividing each element of a row by the maximum value in the row, and multiplying the result by 100. The results after normalization are shown in Table III.

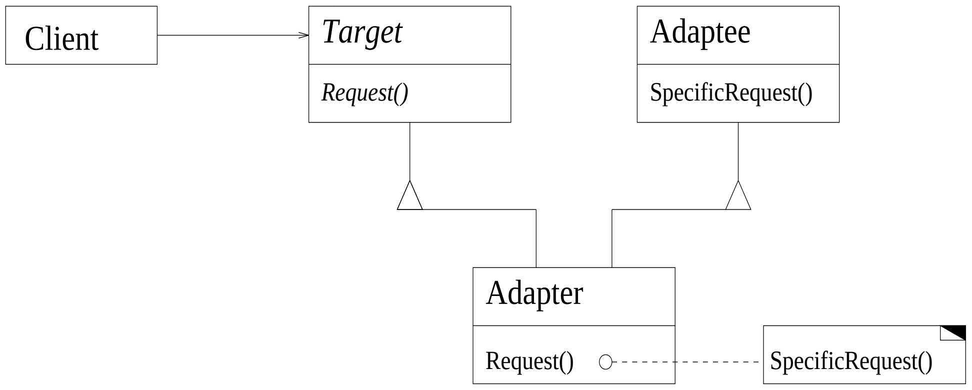


Figure 2: Class Adapter pattern

Table III

Normalized metric values for Adapter patterns

No.	Metric	Class Adapter	Object Adapter
1	TM	100	100
2	PHM	100	100
3	PTM	0	0
4	PRM	100	100
5	PCCM	100	100
6	PMDDNLA	0	0
7	ASDIDAPM	0	100
8	ASDINAPM	100	50
9	TC	100	100
10	PRC	100	100
11	NROP	0	0

Consider the "+" entries in the correlation table to be 1 and the "-" entries to be -1. For each characteristic, the metric values are multiplied with the values in the corresponding entries of the correlation table and summed. This gives a quantitative measure for the characteristic.

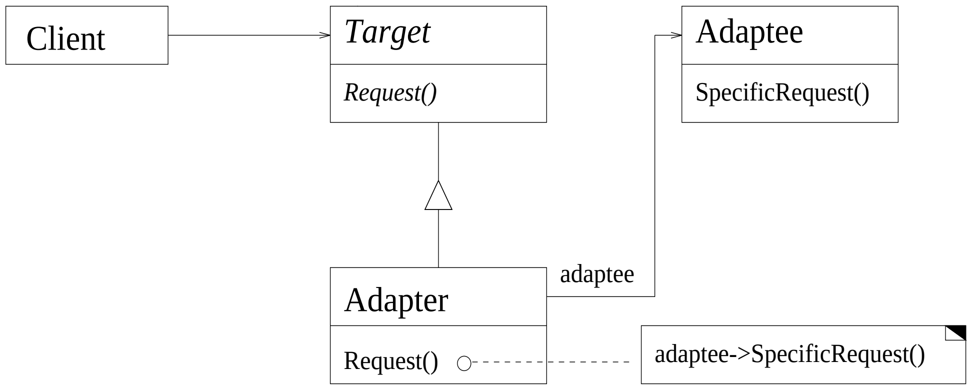


Figure 3: Object Adapter pattern

For Class Adapter :

Coding Effort = (1 * 100) + (1 * 100) + (1 * 0) = 200
Static Adaptability = (1 * 100) + (-1 * 100) + (-1 * 100) + (1 * 100) + (1 * 100) = 100
Dynamic Adaptability = (1 * 0) + (1 * 0) = 0
Performance = (-1 * 0) + (1 * 100) + (1 * 100) + (-1 * 0) + (-1 * 0) = 200

For Object Adapter :

Coding Effort = (1 * 100) + (1 * 100) + (1 * 0) = 200
Static Adaptability = (1 * 100) + (-1 * 100) + (-1 * 100) + (1 * 50) + (1 * 100) = 50
Dynamic Adaptability = (1 * 0) + (1 * 100) = 100
Performance = (-1 * 0) + (1 * 100) + (1 * 100) + (-1 * 0) + (-1 * 100) = 100

The handbook will have entries as shown in Table IV.

Table IV
Handbook entries for Adapter patterns

Pattern	Coding Effort	Static Adaptability	Dynamic Adaptability	Performance
Class Adapter	200	100	0	200
Object Adapter	200	50	100	100

4.2 Handbook Usage

A decision situation is said to exist if it is necessary to choose one course of action from among several. During the design of an object oriented system using design patterns, decision situations may occur when there is more than one pattern which is applicable to the given problem. The designer has to select one which is best suited to his requirements. The handbook helps in this decision making.

This is illustrated with an example.

Consider a design problem in which a client requires a service which is provided by a server, whose interface is different from that which is expected by the client. This problem can be solved by using an Adapter pattern. There exists two types of Adapter patterns viz. Class Adapter and Object Adapter. The designer has to choose one of these.

To select one of these patterns, the value of the Selection Function(SF) for each pattern is calculated. The pattern with highest SF is selected.

Selection Function (SF) $= -(x_1 * a) + (x_2 * b) + (x_3 * c) + (x_4 * d)$

where $x_1..x_4$ are values obtained from the handbook and a, b, c, d are user specified weights. Negative value of the first term in the SF expression is due to the fact that the coding effort is to be minimized.

When performance is the most important criterion of the system being designed, the set of values for a, b, c and d can be 0, 0, 0 and 1 respectively. By consulting the handbook, the following values for SF are obtained.

Class Adapter	: SF = 200
Object Adapter	: SF = 100

Therefore, Class Adapter pattern will be chosen for the above design problem.

5 Conclusion

A set of eleven metrics was proposed for the selection of patterns in the context of object oriented system design. The metrics were correlated with four important attributes of the pattern viz. coding effort, static adaptability, dynamic adaptability and performance. Steps involved in the construction and usage of the handbook of design patterns were demonstrated taking the specific case of selection of an Adapter pattern. We are presently working on a detailed software process model which uses design handbooks. We expect this approach to go a long way in providing a formal basis for systematic reuse of design knowledge in building object oriented software systems.

References

[1] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns – Elements of Reusable Object–Oriented Software*, Addison-Wesley Publishing Company, 1995

[2] Pree, W., *Design Patterns for Object–Oriented Software Development*, ACM Press, 1995

[3] Booch, G., *Object Oriented Analysis And Design With Applications*, Benjamin/Cummings Publishing Company Inc, 2nd edition

[4] Harrison, W., "Software Measurement : A Decision-Process Approach," *Advances in Computers*, 1994

[5] Chidamber, S.R. and Kemerer, C.F., "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, June 1994

[6] Coad, P., "Object Oriented Patterns," *Communications of the ACM*, Sept 1992

[7] Anderson, B., "OOPSLA '93 Workshop Report, Patterns: Building Blocks for Object-Oriented Architectures," *ACM SIGSOFT*, Jan 1994

[8] Lea, D., "Christopher Alexander: An Introduction for Object-Oriented Designers," *ACM SIGSOFT*, Jan 1994

[9] Schmidt, D.C., "Using Design Patterns to Develop Reusable Object-Oriented Communication Software," *Communications of the ACM*, Oct 1995

[10] Marciniak, J.J., *Encyclopedia of Software Engineering - Volume 2*, Wiley-InterScience Publication, 1994

[11] Hicks, T.G., *Standard Handbook of Engineering Calculations*, McGraw-Hill Publications, 3rd Edition