A quantitative approach for evaluating the quality of design patterns

Nien-Lin Hsueh a,*, Peng-Hua Chu a, William Chu b

a Department of Information Engineering and Computer Science, Feng-Chia University, Taichung, Taiwan
b Department of Computer Science and Information Engineering, Tunghai University, Taichung, Taiwan

Received 1 May 2007; received in revised form 21 November 2007; accepted 25 November 2007
Available online 4 December 2007

Abstract

In recent years, the influence of design patterns on software quality has attracted an increasing attention in the area of software engineering, as design patterns encapsulate valuable knowledge to resolve design problems, and more importantly to improve design quality. As the paradigm continues to increase in popularity, a systematic and objective approach to verify the design of a pattern is increasingly important. The intent session in a design pattern indicates the problem the design pattern wants to resolve, and the solution session describes the structural model for the problem. When the problem in the intent is a quality problem, the structure model should provide a solution to improve the relevant quality. In this work we provide an approach, based on object-oriented quality model, to validate if a design pattern is well-designed, i.e., it answers the question of the proposed structural model really resolves the quality problems described in the intent. We propose a validation approach to help pattern developers check if a design pattern is well-designed. In addition, a quantitative method is proposed to measure the effectiveness of the quality improvement of a design pattern that pattern users can determine which design patterns are applicable to meet their functional and quality requirements.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Design pattern; Object-oriented metrics; Software quality

1. Introduction

Software quality has been recognized as an important topic since the early days of software engineering. In this paper, we describe our research and why we focus on evaluating a design pattern. In the past, researchers and practitioners have various approaches to examine how systems can meet specific software quality requirements. Recently, a growing number of practitioners have shown great interest in using design patterns for high-quality software, since design patterns represent high-level abstractions that reflect their own experiences (Chung et al., 2003; Graves and Czarnecki, 2000; Gross and Yu, 2001; Tahvildari and Kontogiannis, 2002). Design patterns have become a popular means to encapsulate object-oriented designs. They capture successful solutions to recurring problems that arise when building software systems (Vliet, 2000).

As the paradigm continues to increase in popularity, analyzing design patterns in order to verify their correctness is becoming increasingly important. In general, a design pattern consists of four essential sessions (Gamma et al., 1994): the pattern name session which describes the design problem, its solutions, and consequences in short sentences, the intent session which describes situations in which the pattern may be applied, the solution session which provides an abstract description of the solution, including the elements of the solution as well as their relationships and collaborations, and the consequences session which describes the benefits and drawbacks of using this pattern. In a well-designed design pattern, the sessions should be consistent with each other. For example, the structural model in the solution session should resolve the problem indicated in the intent session. In this paper, we focus on the consistency validation between the intent and solution sessions.

* Corresponding author. Tel.: +886 4 24517250x3773.
E-mail addresses: nlhsueh@fcu.edu.tw (N.-L. Hsueh), cph@soft.iecs.fcu.edu.tw (P.-H. Chu), cchu@thu.edu.tw (W. Chu).

Available online at www.sciencedirect.com
Inspired by the work of Huston (2001), which provides an analysis method to examine the compatibility between design patterns and design metrics, we propose a quantitative approach in this paper to measure the effectiveness of a pattern to improve the quality properties of a design. Our approach provides the following benefits:

- An evaluation approach is proposed to help pattern developers check if a design pattern is well-designed.
- A quantitative method is proposed to measure the effectiveness of the quality improvement of a design pattern. Based on this information, pattern users can determine which design patterns are applicable to meet their functional and quality requirements.

The remainder of this paper is structured as follows: Section 2 describes some background techniques used in our approach. Section 3 introduces our approach to measure the effectiveness of the quality improvement of a design pattern. In Section 4 we demonstrate a case study, in which a software tool is developed, to support our approach. Section 5 summarizes our approach.

2. Background work

Two important research areas are related to our work, i.e., the role-based pattern representation (Gamma et al., 1994) and the quality model of object-oriented design (Bansiya and Davis, 2002).

2.1. Role-based pattern representation

In general, a design pattern consists of four essential sessions (Gamma et al., 1994): pattern name, intent, solution, and consequences. The solution is informally presented by using class and interaction diagrams in model level, i.e., the model is an example model. For example, the structure of the solution of the design pattern Abstract Factory is presented in Fig. 1a, showing an example of two products families (family A and family B). The limitation of this concept is, though Abstract Factory can be applied to multiple product families, the structure cannot convey this concept in a systematic way.

France et al. (2004) proposed a role-based model to formally specify pattern solution in a meta-model level. A role in the meta-model representation specifies the properties a model element must have if it is to be part of the pattern solution model. Referring to the Abstract Factory example, the roles defined in this pattern include AbstractFactory, ConcreteFactory, AbstractProduct, ConcreteProduct and Client. Fig. 1b represents the structural solution based on the role-based approach. Please note that the model elements ProductA and ProductB groups in Fig. 1a are now modeled as a role ConcreteProduct.

2.2. Quality model of object-oriented design

In order to provide a quantitative approach to relate measurable object-oriented characteristics to the higher-level desirable software quality attributes, Bansiya and Davis extended the Dromey’s generic quality model (Dromey, 1995) to propose a hierarchical model for an object-oriented design quality assessment approach, called QMOOD (Bansiya and Davis, 2002). As shown in Fig. 2, there are four levels and three mappings between these levels in QMOOD. While defining the levels involves identifying design quality attributes, object-oriented design properties, object-oriented design metrics, and object-oriented design components, defining the mapping involves connecting adjacent levels by linking a lower level to the next higher level. Reusability, flexibility, effectiveness, understandability, functionality, and extendibility are selected as a typical set of quality attributes in the QMOOD level one.

Design properties defined in level two are used to assess the quality attributes in level one. For example, coupling is used to assess reusability. The linkage $l_{12}$ indicates that a model with lower coupling possesses the attribute of high reusability. To measure the coupling degree, a metric $cou$-
pling factor (COF) is defined in level three for counting the different numbers of classes that a class is directly related to. The linkage \( l_{23} \) indicates that COF is related to the coupling property. Level four elements consist of a set of primitive elements in an object model, including objects, classes, relationships, generalization, etc. The linkage \( l_{34} \) indicates that class and relationship are level four elements that are related to the COF metric. For simplicity, the QMOOD model is referred to as \( \Phi \). The set of quality elements in the QMOOD level \( i \) is denoted as \( \Phi_i \), and the set of link \( ij \) is denoted as \( \Phi_{ij} \). If an element \((x, y) \in \Phi_{23}\), then \( y \) is an object-oriented design metric defined in \( \Phi_3 \), used to evaluate the object-oriented property \( x \) defined in \( \Phi_2 \).

### 3. Pattern evaluation

In this section, we describe why design patterns are related to quality, our proposed representation for design patterns and the proposed quality evaluation model. The result shows the Mediator is a well-defined coupling-improver pattern, which conforms to our expectation to Mediator.

#### Table 1: Comparison of different design patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>FR-intent ((I_F))</th>
<th>NFR-intent ((I_N))</th>
<th>Quality focus ((Q))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observer</td>
<td>A subject object can notify all related objects when it changes state</td>
<td>Without knowing types of the observers, a subject object can automatically notify all observers</td>
<td>(\text{(coupling, dec)})</td>
</tr>
<tr>
<td>Abstract Factory</td>
<td>A client object creates/uses a set of related objects (called products)</td>
<td>A client object creates or uses these product objects without specifying their concrete types</td>
<td>(\text{(coupling, dec)})</td>
</tr>
<tr>
<td>Strategy</td>
<td>An object uses an algorithm to resolve a specific problem</td>
<td>Easier to replace the algorithm with a new one</td>
<td>(\text{(poly, inc)})</td>
</tr>
<tr>
<td>Iterator</td>
<td>A client object navigates an aggregate object</td>
<td>Traverses the aggregate object without knowing its internal structure</td>
<td>(\text{(encap, inc)})</td>
</tr>
</tbody>
</table>

\(\text{inc: increased; dec: decreased; poly: polymorphism; encap: encapsulation.}\)

### 3.1. Analyzing design patterns from a quality perspective

Design patterns usually provide a possible way to deal with non-functional requirements since they provide solutions to satisfy functional requirements as well as “better” solutions to meet non-functional requirements (Winn and Calder, 2002). For example, considering the original intent described in the Observer design pattern (Gamma et al., 1994):

Define an one-to-many dependency between objects so that when one object changes state, all its dependents are automatically notified and updated.

The Observer design pattern is designed to address the communication problem between subject objects and their related observer objects. Viewing from a functional aspect, it requires the subject to notify all observers when it changes its state. Viewing from a non-functional aspect, it requires the notification to work automatically without knowing the types of observers. Based on this observation, we were inspired to explore how a design pattern can enhance non-functional requirements. A design pattern from this perspective is defined as a tuple \((I_F, I_N, Q, S_F, S_N, T)\):

- \(I_F\): Functional Requirement Intent (FR-intent), describing what does the pattern do.
- \(I_N\): Nonfunctional requirement Intent (NFR-intent), describing how well can this pattern contribute to quality attributes, such as reusability, maintenance or extensibility.
- \(Q\): Quality Focus, representing the quality focus from \(I_F\) to \(I_N\).
- \(S_F\): FR-structure, representing the structure that can realize the functional requirement intent \((I_F)\).
- \(S_N\): NFR-structure, representing the structural model that can enhance the non-functional requirement intent \((I_N)\).
- \(T\): Transformation, representing the transformation function from \(S_F\) to \(S_N\).

Both the FR- and the NFR-intents are text descriptions to specify the functional and non-functional intent of a design pattern. For example, the FR-intent of the Abstract Factory design pattern requires the client object to create or use a set of related objects (called products), and the NFR-
intent requires the client object to create or use these product objects without specifying their concrete types. Table 1 illustrates our analysis of some design patterns that are described by Gamma et al. (1994). The FR-intent and the NFR-intent of a design pattern \( dp \) are denoted as \( I_F(dp) \) and \( I_N(dp) \), respectively.

The FR-structure and the NFR-structure specify the structure for fulfilling the FR-intent and the NFR-intent, respectively. Fig. 3 illustrates the FR-structure and the NFR-structure of the Abstract Factory design pattern. The NFR-structure can enhance the NFR-intent in the sense that it can satisfy the NFR-intent to a higher degree than compared to its associated FR-structure. Note that the FR- and the NFR-structures are meta-models and represented by a role-based approach introduced in Section 2. The FR-structure and the NFR-structure of a design pattern \( dp \) are denoted as \( S_F(dp) \) and \( S_N(dp) \), respectively.

The transformation represents the essence of a design pattern. The transformation \( T \) maps a FR-structure to a NFR-structure. That is,

\[
\forall s \in S_F(dp), \quad T(s) \in S_N(dp),
\]

where the notation \( \in \) denotes the instantiation relationship between model and meta-model. \( s \in S \) means that the model \( s \) is an instance model of the meta-model \( S \). We take advantage of Fig. 4 to illustrate this transformation. First, we design the \( S_F \) and \( S_N \) of \( dp \) at the metamodel level, and randomly generate \( s \) by \( S_F(dp) \). In order to execute such a transformation, we have to design the Pattern Transformation Specification at the metamodel level, so that \( T \) can transform \( s \) into \( T(s) \), which corresponds to \( S_N(dp) \) and automatically validate the correctness of the model. This will be explained in more detail in step 3 of the case study in Section 4.

To more explicitly highlight the quality issue that a design pattern addresses, the extension from the FR-intent to the NFR-intent is defined as a quality focus \( Q(dp) = \{property, constraint\} \). The property refers to an object-oriented design property defined in \( \Phi_2 \), which may be design size, hierarchy, abstraction, encapsulation, coupling, cohesion, composition, inheritance, polymorphism, message, or complexity. Constraint refers to the expected constraint on the subject's property, which may be increased, decreased or maintained. The constraint increased means that the NFR-intent expects to maximize the design property. For example, the quality focus of the NFR-intent of Abstract Factory is defined as (coupling, decreased) to indicate the intent to decrease the coupling degree. A design pattern's quality focus \( Q \) is also denoted as \( Q_{dp}(I_F, I_N) \) to explicitly indicate that the quality focus is derived from \( I_F \) to \( I_N \). Table 1 illustrates more examples of the concept of quality focus. For example, the quality focus of the Observer design pattern can be formulated as (coupling, decreased), meaning that it intends to decrease the coupling. The quality focus of the Strategy design pattern is defined as (polymorphism, increased), meaning that using the Strategy design pattern can increase the number of polymorphic operations. Note that the properties of coupling and polymorphism are not only a general notion in the object orientation, but also object oriented properties defined in level 2 of QMDOOD (that is, \( \Phi_2 \)).

Fig. 5 illustrates the relationship between the elements in the Abstract Factory design pattern. The FR-structure is a meta-model to realize the FR-intent, whereas the NFR-structure is a meta-model to enhance the NFR-intent. The quality form is defined on the extension between the FR-intent and the NFR-intent to explicitly indicate the quality focus. The transformation is defined between the FR-structure and NFR-structure to indicate the mapping of their instance models.

3.2. Quality improvement effectiveness

Let \( \mu \) be an object-oriented metric, \( \mu(m) \) denotes the metric value of the object model \( m \). \( (P, \mu) \in \Phi_2 \) if \( \mu \) can be used to evaluate the object-oriented design property \( P \).
(since $\Phi_{23}$ defines the link between property and metric). $\mu(m_1) < \mu(m_2)$ means that the model $m_2$ possesses a higher property $P$ than $m_1$. Given a model $m$, $\mu(m)$ can be calculated directly based on the definition of the metric $\mu$. For example, COF (COupling Factor) is a metric to assess the property coupling, which is defined as (Brito and Abreu, 1995):

$$\frac{\sum_{j=1}^{TC} |\sum_{i=1}^{TC} is\_client(C_i, C_j)|}{TC^2 - TC},$$

where $TC$ is the total number of classes and

$$is\_client(C_i, C_j) = \begin{cases} 1, & \text{if } C_i \Rightarrow C_j \cap C_j \neq C_j \\ 0, & \text{otherwise}. \end{cases}$$

The $is\_client$ function returns 1 if class $C_i$ contains at least one non-inheritance reference to class $C_j$, where a reference may be an argument type, a returned value, or a call to methods of server classes.

The notion of object metric is applied in the meta-level to evaluate a transformation, $\mu(M, T)$ is a metric to evaluate the effectiveness of a transformation on a meta-model:

$$\mu(M, T) = G_{m \in M} \left( \frac{\mu(T(m)) - \mu(m)}{\mu(m)} \right), \tag{1}$$

where $G$ is an aggregation function such as an average function. $M$ is a typical model set whose elements are instantiated from $M$ and represent a typical case in a context. $\mu(M, T) > 0$ means that the transformation $T$ enhances the property $P$, i.e., $T$ can transform instances of $M$ into another model with a higher $P$. The transformation $T$ is effective on the meta-model $M$ with respect to the property $P$ if $\mu(M, T) < 0$. The transformation $T$ impairs the property $P$ if $\mu(M, T) < 0$. Based on $\mu(M, T)$, we define the quality improvement effectiveness on property $P$ of a design pattern (denoted as $QIEP(dp)$). Assume a design pattern $dp = (I_F, I_N, (P, C), S_F, S_N, T)$, its $QIEP$ is measured on the basis of the FR-structure $S_F$, the transformation $T$, and the metric $\mu$:

$$QIEP(dp) = \mu(S_F, T) = G_{s \in S_F}(QIEP_{s,F,s,N}),$$

$$= \begin{cases} 1, & \text{if } \mu(s) = 0 \\ G_{s \in S_F} \frac{\mu(T(s)) - \mu(s)}{\mu(s)}, & \text{otherwise.} \end{cases} \tag{2}$$

$QIEP_{s,F,s,N}$ is the quality improvement effectiveness measurement from FR-structure to NFR-structure. $QIEP(dp) > 0$ means that applying the design pattern $dp$ will enhance the property $P$, whereas $QIEP(dp) < 0$ means that $dp$ will impair the property $P$. Furthermore, the value $|QIEP(dp)|$ is larger, the $dp$ impacts the property $P$ more. If $QIEP(dp) = 0$, the design pattern is inefliccious to $P$. For example, we can apply COF metric to the Abstract Factory design pattern to measure its effectiveness on coupling:

$$QIE\_coupling(AF) = \text{AVG}_{s \in S_F(AF)} \left( \frac{\text{COF}(T(s)) - \text{COF}(s)}{\text{COF}(s)} \right),$$

where AVG is an average function. If $QIE\_coupling(AF) < 0$, this means that applying the $AF$ design pattern will impair the coupling property.

### 3.3 Design pattern for quality improvement

We define a design pattern as a quality-improver if its quality intent is consistent with its structure. To validate consistency, we check the $QIEP$ with the constraint defined in the quality focus. A design pattern $dp = (I_F, I_N, (P, C), S_F, S_N, T)$ is a property $P$ improver if

$$QIEP(dp) = \begin{cases} > 0, & \text{if } C = \text{“increased”}, \\ < 0, & \text{if } C = \text{“decreased”}, \\ 0, & \text{if } C = \text{“maintained”}. \end{cases} \tag{3}$$

We use $P$-improver($dp$) to denote $dp$ is a design pattern improver on property $P$. When a design pattern $dp$ is a quality-improver, it implies its NFR-structure provides a better solution for enhancing the NFR-intent, i.e., $P$-improver($dp$) $\Rightarrow d(I_F, S_F) < d(I_N, S_N)$, (4) where $d(intent, structure)$ is a satisfaction function, representing the degree to which the intent can be realized in the structure. The degree is called satisfaction degree in our approach. If the intent is a functional intent, the degree is either 1 (satisfied) or 0 (not satisfied). If the intent is non-functional, the degree ranges between 0 and 1. The satisfaction degree is just a concept which cannot be directly measured. The predicate $d(I_N, S_F) < d(I_N, S_N)$ in (4) is indirectly derived by evaluating whether or not the design pattern is a $P$-improver. If the design pattern is a $P$-improver, $d(I_N, S_F) < d(I_N, S_N)$ is true, which means the NFR-intent can be satisfied to a greater degree in the non-functional than in the functional structure.

When we apply the NFR-structure to enhance the satisfaction degree of the NFR-intent, we also have to make sure the NFR-structure will not impair the original FR-intent of the pattern. We thus define a well-designed $P$-improver as follows:

$$A \text{ design pattern } dp \text{ is a well-designed } P\text{-improver if it is a } P\text{-improver and its NFR-structure does not impair its FR-intent}, \text{ i.e.,}$$

$$P\text{-improver}(dp) \iff P\text{-improver}(dp) \land (d(I_F, S_F) = \text{satisfied}) \tag{5}$$

$d(I_F, S_N) = \text{satisfied}$ means the FR-intent is not impaired by the NFR-structure. The judgment of the condition is made by the pattern designer, by examining if the classes in $S_N$ have enough operations and proper interactions to implement the functionality of $I_F$.

Fig. 3 represents the FR-structure ($S_F(AF)$) and the NFR-structure ($S_N(AF)$) for fulfilling $I_F(AF)$ and $I_N(AF)$. Compared to $S_F(AF)$, $S_N(AF)$ introduces a surrogate object
**Factory** for bridging the connection between the client and the product objects. Thus $I_{N}(AF)$ is enhanced since the coupling degree between the client and the product objects is reduced.

### 4. Case study and system design

In this section, we design some experiments as examples to demonstrate our approach. These experiments were carried out by the tool **Pattern Verification System (PVS)**, developed by us, and the commercial UML modeling tool by IBM Rational Rose. With the IBM Rational Rose, the FR-structure, the NFR-structure and the transformation were modeled as UML profiles (UML Semantics, 2007) and exported as XML files. These files were imported into PVS and the quality evaluation task performed.

#### 4.1. A validation example: Mediator pattern

To validate our approach, we conduct a step-by-step experiment on the **Mediator** pattern.

**Step 1: Extraction the nonfunctional intent and the quality focus definition**

In the first step, we investigate the original definition described in the **Mediator’s** intent (France et al., 2004):

> Define an object that encapsulates how a set of objects interacts. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

The functional intent is to let a set of objects communicate with each other, and its non-functional intent is to decrease the coupling between these objects. Therefore, we have

- $I_{F}(\text{Mediator})$: A colleague object can communicate with other colleague objects.
- $I_{N}(\text{Mediator})$: A colleague object can communicate with other colleague objects in a loose coupling manner by explicitly keeping objects from referring to each other.

To determine the quality focus, we check the design property in QMOOD to see which design properties relate to coupling. We finally define the quality focus of this design pattern as \(\langle \text{coupling, decreased} \rangle\). Therefore, we have $Q(\text{Mediator}) = \langle \text{coupling, decreased} \rangle$.

**Step 2: Designing the pattern structures**

In the second step, we build the FR- and the NFR-structure for the **Mediator**. Fig. 6a and b illustrates the FR-structure ($S_{F}(\text{Mediator})$) and the NFR-structure ($S_{N}(\text{Mediator})$) in a role-based approach, respectively. The major difference between the two structures is that $S_{N}(\text{Mediator})$ introduces a Mediator object to serve as a broker among Colleagues, thus avoiding their direct communication between them.

To make the transformation between the FR- and the NFR-structure automatically, we apply France’s approach (France et al., 2004) to build the pattern transformation specifications for the design patterns. A pattern transformation specification is built in a meta-model level in which the source and the target of the transformation and the transformation itself are all represented in the meta-model level. In this step we first describe how to present the FR- and the NFR-structure in the meta-model level.

To avoid inventing new notations for new concepts, UML supports an extension mechanism called “Profile” to allow developers to create new meta-model elements under the UML semantics (Heaven and Finkelstein, 2004; UML Semantics, 2007). Fig. 7c illustrates a part of the profile representation of the $S_{N}(\text{Mediator})$. For example, Colleague and their inter-relationship, are marked as $$\text{Class}$$ and $$\text{Association}$$, respectively. The link point between the Colleague and the relationship is marked with the stereotype $$\text{AssociationEnd}$$.

**Step 3: Designing the transformation**

We adopt the approach of Dong and Yang (2003) to construct the pattern transformation specification in which the transformation is a mapping from the meta-model of the FR-structure (as the source model) to the NFR-structure (as the target model). In Fig. 7, four tags are defined to describe the mapping:

- $$\langle \text{removed} \rangle$$: if the element is only in the source model but not in the target model, if the element is marked as $$\langle \text{removed} \rangle$$ (we omitted the tag to keep it concise);
- $$\langle \text{new} \rangle$$: if an element only appears in the target model but not in the source model is marked as $$\langle \text{new} \rangle$$;
- $$\langle \text{unmodified} \rangle$$: if after transformation the element is the same as before, we use a transformation association which is denoted $$\langle \text{unmodified} \rangle$$ stereotype;
- $$\langle \text{modified} \rangle$$: if after transformation the element is modified, we use an association class which is denoted $$\langle \text{modified} \rangle$$ stereotype to indicate the attributes or operations that have been added or changed. The transformation specification $T$, as well as $S_{F}$ and $S_{N}$, are created using the Rational Rose tool, exported to an XMI, and imported to PVS for further pattern validation.

**Step 4. Performing pattern validation**

Before performing the validation, we first set the sampling parameters for the pattern. Remember, $S_{F}$ may have an unlimited number of instance models $s_{F}$. Therefore, we have to set a reasonable number of instance models in each pattern verification. In our experiment, as Table 2, we set the number to be 20. PVS randomly generates 20 sample $s_{F}$ based on the Mediator pattern $S_{F}$ specifications. Then, we set the context in each $s_{F}$ – including the number of participating classes and their connections. We assume that

---

1 In the Mediator design pattern, these objects are called Colleague object.
the numbers of Colleague classes in each sample model ranges from 3 to 10, and that the associations are randomly constructed. After the setting, each $s_F$ is transformed to $s_N$ using the pattern transformation $T(Mediator)$. Fig. 8 is one of the samples where (a) part is a $s_F$ model and (b) part is the transformed $s_N$ model.

Since $Q(Mediator) = \langle coupling, decreased \rangle$ and the object metric in $\Phi_3$ to measure coupling is COF, we apply
COF to each $s_F$ and $s_N$. $QIE_{s_F,s_N}$ and $QIE_{s_F}(Mediator)$ are therefore calculated by Eq. (2). By using average as the aggregation function, $QIE_{s_F}(Mediator) = -0.72$ is negative, consistent with the constraint “decreased” in $Q(Mediator)$ (based on Eq. (3)), therefore we conclude the Mediator is a Coupling-Improver.

4.2. Discussion

To validate our approach, we apply object-oriented metrics of QMOOD to Gamma’s design patterns. Some interesting results are discussed in this section.

Polymorphism is one of the design properties in QMOOD, defined as the ability to substitute objects whose interfaces match for one another at run-time. A system with polymorphism possesses better flexibility, extensibility and effectiveness. When we evaluate the polymorphism property of design patterns, we found most of them are polymorphism-improver. This result makes sense since most design patterns apply polymorphism mechanism to improve their design flexibility, such as Abstract Factory, Builder, Observer, Strategy and State. Though these patterns have different intent and different applicability, they have similar transformation style from FR-structure to NFR-structure, as shown in Fig. 9. Before the transformation, Role1 objects call Role2 object directly. After the transformation, a hierarchy is created in which we can extend the Role2 without modifying the Role1. Polymorphic methods are added in ConcreteRole2 and/or ConcreteRole1. Obviously, the number of polymorphic methods is increased by at least 1. Assume we have $n$ polymorphic

![Fig. 8. An Example of Mediator design pattern.](image1)

![Fig. 9. Similar structure of most design patterns.](image2)
method in the FR-structure, we will have \( n + 1 \) polymorphic methods in the NFR-structure, therefore the QIE_{polymorphism}(pattern) is close to \( \frac{(n+1)-n}{n} = \frac{1}{n} > 0 \), compatible with the quality focus of the design patterns.

Most polymorphism-improver design patterns are also hierarchy-improver and abstraction-improver. Hierarchy represents the generalization-specialization concept in a design, when properly applied, can enhance a system’s functionality. Abstraction also represent the generalization–specification concept, but focus on the average number of classes from which a class inherits information. A system with abstraction may have better extensibility and effectiveness. When we apply polymorphism to a pattern’s transformation, a new hierarchy is established and the abstraction increases consequentially. Fig. 10 is an example of Observer design pattern. In its FR-structure, the Subject object calls the Observer object directly. After the transformation, two hierarchies are created, concrete observers are abstracted as a more general class Observer, and the polymorphic methods update is added. The QIE value for hierarchy, abstraction and polymorphism are all positive, indicating the Observer pattern is an improver of the three design properties.

A pattern with hierarchy structure does not imply it is a hierarchy or abstraction improver. For example, the Decorator pattern in Fig. 11 looks like a hierarchy-improver, but its QIE value is negative in our experiment. The result is comprehensible. In its FR-structure, static inheritance is applied to extend functionality, which may produce an explosion of subclasses to support every combination. Such an design may have high degree of abstraction (ANA\(^2\) of model (a) is 3.14). On the other hand, the NFR-structure utilizes an object composition approach to extend responsibilities in a flexible way, by which the responsibility can be attached to the Decorator object at run-time dynamically and get lower ANA value (ANA of model (b) is 2.23). Therefore, QIE_{abstraction}(Decorator) is negative, representing it is not an abstraction-improver. The benefits of Decorator is to provide a flexible structure for extending responsibilities without defining too many subclasses. Decorator is actually a complexity-improver since it can reduce the number of classes and methods.

Not all design patterns are quality improvers. For example, Singleton design pattern is proposed to resolve a specific design or programming problem, not for quality improvement. As creating an object is easy, but constraining a class hierarchy to create just one object is not easy. Singleton is proposed to resolve this problem by declaring a protected reference on the superclass and making the constructors of these classes private. In this case, we will
not treat it as a quality-improver and will not define its quality model. In fact, in our previous work (Hsueh et al., 2007), design patterns are classified as activity-facilitator for facilitating design activities; quality-improver for handling non-functional requirements and improving software quality; problem-solver for solving design problems; and conflict-resolver to resolving design conflicts. Singleton is treated as a problem-solver in our classification.

5. Conclusion

As indicated by Winn and Calder (2002), a design pattern should be able to deal with non-functional requirements. In this work we propose an approach to analyze and verify design patterns from the perspective quality. A quality-improver design pattern is a design pattern which intends to address quality requirements and has a better structure for addressing them. To verify the consistency between the intent and the structure of a design pattern, an object-oriented quality model is used. The basic idea is "if the pattern’s intent maps to an object-oriented property (for example, coupling), its structure should support that property; and the object-oriented property of a structure can be evaluated by object-oriented metrics (for example, COF metric)". Verification of a design pattern can help control the quality of the design pattern as it is widely used in object-oriented development. We also propose a quality improvement effectiveness measurement to measure the effectiveness of improving the quality of a design pattern.

In the future, we plan to develop a tool for our verification method. A XMI-based representation will be developed for representing the metamodel functional and non-functional structures of a design pattern. To define the transformation in the design pattern, we will explore the possibility of applying the technique of Model-Driven Architecture (MDA Miller and Mukerji, 2003; Meservy and Fenstermacher, 2005).

Acknowledgment

This work was support by the National Science Council, Taiwan, under Grant NSC94-2213-E-035-041, and Department of Industrial Technology, Ministry of Economic Affairs, Taiwan, under the Grant 96-EC-17-A-02-S1-029.

References


Nien-Lin Hsueh is an assistant professor in the Department of Information Engineering and Computer Science at Feng Chia University, Taiwan. His research interests include software engineering, object-oriented methodologies, software process improvement, and data hiding. He was the technology consultant in the CMMI-based process improvement project of Office of Information Technology at Feng Chia University. He received his Ph.D in Computer Science from National Central University in Taiwan.

Peng-Hua Chu is a Ph.D student in the Department of Information Engineering and Computer Science at Feng Chia University, Taiwan. His research interests include design pattern, object-oriented methodologies, software engineering education, software process improvement and software engineering. He received his MS and BS from Feng Chia University.

William Cheng-Chung Chu is a professor of the Department of Computer Science and Information Engineering and the Director of Software Engineering and Technologies Center. He had served as the Dean of Research and Development office at Tunghai University from 2004 to 2007, Taiwan. From 1994 to 1998, he was an associate professor at the Department of Information Engineering and Computer Science at Feng Chia University. He was a research scientist at the Software Technology Center of the Lockheed Missiles and Space Company, Inc., where he received special contribution awards in both 1992 and 1993 and a PIP award in 1993. In 1992, he was also a visiting scholar at Stanford University. His current research interests include software engineering, embedded systems, and E-learning. Dr Chu received his MS and Ph.D degrees from Northwestern University in Evanston Illinois, in 1987 and 1989, respectively, both in computer science. He has edited several books and published over 100 referred papers and book chapters, as well as participating in many international activities, including organizing international conferences.