AN EMPIRICAL INVESTIGATION OF METHODS FOR TEACHING DESIGN PATTERNS WITHIN OBJECT-ORIENTED FRAMEWORKS

SIN BAN HO†, IAN CHAI‡ and CHUIE HONG TAN†§
† Faculty of Information Technology, Multimedia University, Jalan Multimedia, 63100 Cyberjaya, Selangor, Malaysia
sbho@mmu.edu.my
‡ Faculty of Engineering, Multimedia University, Jalan Multimedia, 63100 Cyberjaya, Selangor, Malaysia
ianchai@mmu.edu.my
§ Faculty of Information Technology, Multimedia University, Jalan Multimedia, 63100 Cyberjaya, Selangor, Malaysia
chtan@mmu.edu.my

Received 10 May 2005
Revised 15 August 2006
Accepted (Day Month Year)

Design patterns have become an important part of modern object-oriented design. It is necessary to evaluate and compare the effectiveness of different methods of pedagogical documentation in teaching design patterns. We compared patterns-style documentation with minimalist documentation. Patterns style disagrees with minimalism in that contextual information is not minimal - it usually contains information other than what needs to be done. The discoveries in this empirical study were two fold. First, the MANOVA (multivariate analyses of variance) shows that the effect of patterns-style documentation and minimalism are different. Second, we proposed a prediction model using a regression modeling technique. Considering these results, we formulate some guidelines for effective framework documentation by applying five of the design patterns within the Swing environment as our case study. Different documentation philosophies are better for different goals. For a simple task, use minimalist documentation. For a much more complex problem such involving five of the design patterns, our empirical results suggest using patterns-style documentation.

Keywords: Framework documentation; decision making; pedagogy practice; patterns; minimalist.

1. Introduction

Design patterns are usually related to the small and medium-scale design of objects and frameworks. Craig Larman explained that these design patterns are also known as micro-architectural patterns. They are applicable to designing a solution for connecting the large scale elements and among detailed design work for any logical aspect. A pattern is a proven successful solution to a recurring problem in a context. Typically, a framework uses multiple design patterns (DP) to govern the rules to form various components, which subsequently collaborate within the framework. Design patterns are specific patterns of object-oriented design that have been documented. The canonical collections of these patterns were published in the GoF book. Minimalist documentation is to minimize the amount of reading and other passive forms of training by allowing
learners to fill in the gaps themselves. This style was developed by John Carroll\(^{10}\) to make all learning activities self-directed. Patterns-style documentation, on the other hand, is a style of documentation that disagrees with the minimalist style in that contextual information is not minimal – it usually contains information other than what needs to be done. Design patterns are usually expressed using a patterns style, but pattern-style documentation can document other things besides design patterns.

While frameworks allow the reuse of both design and code, familiarity with one framework does not mean triviality in learning how to use another.\(^{13}\) Learning a framework means learning a set of parts, learning the kinds of things one might build with them, learning how they interact, and learning the properties that are most useful. The best way to learn a framework is to sit with an expert. However, handholding by an expert will not be practical for most framework users. This is a reason why we need good framework documentation to make mass learning possible. To achieve good documentation, we need to find well-supported rules for documenting frameworks. The major aim of this paper is to aid in this decision-making process.

Recognizing these issues, we investigate the feasibility of a practical solution to the large general problem of how to document frameworks. An empirical approach is proposed to tackle new user documentation or tutorials. This is a very important part because once past the new user stage, one often has the familiarity to figure the details out. Instead of just exposing plain text and sample screen output to the subjects, we intend to investigate the effects of exposing the design diagrams involved in building the code as well. If the subjects were told about the design diagrams before instructing them to code, the documentation may be much more effective. Thus, we intend to discover whether or not patterns-style documentation\(^{24}\) augmented with UML (Unified Modeling Language) class diagrams\(^{32}\) would work better than Dr John Carroll’s\(^{10}\) minimalist approach for new users. This will help us create more effective documentation.

2. Decision Making Process in Design Patterns Pedagogy

This experiment focused on five commonly used design patterns (DP): Composite, Decorator, Observer, Iterator and Strategy.\(^{22}\) The Composite pattern teaches how to bundle several objects that have the same behavior. The Decorator pattern forms a class that adds to the functionality of another class while keeping its interface. The Observer pattern pervades user interface elements, e.g. buttons, menus, checkboxes and sliders in order to tell the observer objects about their attached events. The Iterator pattern overcomes the pitfalls in changing internal implementation by attaching items to a collection of another data structure or in a relational database table. The Strategy pattern allows pluggable algorithms which vary independently from clients that use it.

Fig. 1 presents the independent (uncontrollable) and dependent variables in each category to be investigated in our experiment. This novice behavior model evolved from the consumer behavior model proposed by Turban \textit{et al.}\(^{37}\) in representing the E-Commerce environment. The purpose of having this novice behavior model is to guide
the empirical investigation on how effective a particular documentation philosophy is in teaching a beginner learning a framework. A software engineer who understands these needs could determine which documentation style is suitable in a certain situation, given parameters such as the characteristics of the novices’ demography and the object-oriented framework system. In this study, the novices’ demographic variables include CGPA (Cumulative Grade Point Average), gender and their three prior programming grades, i.e. Data Structure and Algorithm, C++ and C programming language. These demographic coefficients are considered in the regression, which are highlighted in the later section 4.2. For the second set of parameters, we chose the five DP within the Swing framework system due to their relative availability and convenience. The experiment was carried out during tutorials of an Object Oriented Programming course in August 2004, where the design patterns with Swing components were taught.

The guiding rule from this paper is a statistical decision making model similar to that proposed by Ming Li et al.\textsuperscript{38} in their study to detect denial-of-service flooding attacks. These models form a hierarchy of many levels in making decisions. For instance, Norita Ahmad et al.\textsuperscript{39} applied this approach to develop a model that can be used to assess the performance of Small to Medium-Sized Manufacturing Enterprises (SMEs). The projection method proposed by Xu et al.\textsuperscript{40,41} can help to reflect the decision maker’s subjective preferences as to which documentation style would be more suitable for a framework.

On the other hand, a different view is proposed by Prechelt et al.\textsuperscript{30}, which suggests that design patterns should be documented explicitly in the program source code. Their empirical work showed that Pattern Comment Lines (PCL) may considerably reduce the time required for a program change or improve the quality of the change. Another controlled experiment compares design using patterns versus simpler solutions, which are alternative straight-forward designs in the context of program maintenance. Prechelt et
conclude that in most of the maintenance tasks, design patterns help to reduce maintenance time. It is wise to choose the flexibility provided by the DP as unexpected new requirements often appear. In addition, Antoniol et al.\textsuperscript{2} and Penta et al.\textsuperscript{29} focus on the traceability links between code and documenting frameworks, but not on the impact in exposing different philosophies or models of documentation.

In order to converge towards credible answers, a series of experiments with different designs are usually necessary.\textsuperscript{7} This paper complements the earlier findings in the Habanero framework\textsuperscript{11} and the Visual Basic (VB) framework,\textsuperscript{20,21} which presented experimental results on those respective frameworks to determine which documentation styles are better for various goals. The earlier findings reaffirmed that different documentation type have different effects. For a simple task such involving Habanero and VB, use minimalist documentation for faster compilation. However, we were still missing an investigation on the impact of exposing design diagrams in documentation to the new users. The experimental design and results presented in this paper were carried out to address this need: whether or not class diagrams work better in helping software developers to use the framework with DP. We inserted the class diagrams on each web page of the patterns-style documentation. In this paper, we compared patterns-style with minimalist documentation. The patterns-style documentation consists of class diagrams for design patterns with their explanation as part of the contextual information, while the minimalist documentation only contains what needs to be done. The sample size of data in this investigation is 79 subjects.

3. The Empirical Investigation of DP practices

In this empirical work, students were asked to record effort spent under each work task within a framework. Germain and Robillard\textsuperscript{16} supported this approach for its ability to provide direct measurement of the task itself. To be able to get a better understanding of applying DP in the Swing framework, we now present a short description of our experiment design and conduct. The experimental materials include the DP work tasks, questionnaires, programs, and documentation.\textsuperscript{35,36}

3.1. Hypothesis

Standard significance testing was used to clearly specify the effects in learning design patterns with Swing components using the different documentation philosophies. The null hypothesis is stated as follows.

H\textsubscript{0} - There is no difference between the patterns-style documentation with UML class diagrams and minimalist documentation, in terms of completion time, ease of comprehension, accuracy, workings and numbers of difficulties faced.

The interpretations of the experiment are derived from the rejection or non-rejection of this hypothesis. Furthermore, Miller,\textsuperscript{37} Endres and Rombach\textsuperscript{12} affirm that statistical significance testing is still the most preferred remedy for conducting empirical software engineering studies.
3.2. **Setup of the experiment**

In order to test the hypothesis of our experiment, two different groups of the Swing documentation were required. The two sets of on-line programming documents were developed according to the different documentation philosophies. There were 30 students using the minimalism, and 49 students using the patterns-style documentation.

Our experimental materials resembled the pedagogy of information which one would typically find in learning design patterns within the Swing framework. We called this pedagogical documentation the *simple invoice application*. The content was structured as five work tasks, so that the completion time for each checkpoint could be recorded. The five work tasks guide the novices to implement the five chosen design patterns, namely, *Composite, Decorator, Observer, Iterator* and *Strategy* design pattern. Figures 2 and 3 show an example of the pedagogical framework documentation. The background information section is added to the top of each piece of the minimalist documentation in order to form the patterns-style documentation.

### How to implement the **COMPOSITE** design pattern

**Table of Contents**

- Background Information
- How does this work?
- What next?

**Background Information**

In this application, stores may sell **bundles** of related items, such as a bundle of a stereo system, which consisting of a tuner, amplifier, CD player and speakers. It is possible to add a bundle to an invoice. In other words, a bundle contains line items and is again a line item. This is precisely the situation of the **COMPOSITE** pattern, as shown in the following figure.

![COMPOSITE pattern diagram](image)

The **COMPOSITE** pattern teaches us that the **Bundle** class should implement the **LineItem** interface type.

**How does this work?**

/* ... continued with the subsequent steps in implementing the Composite design pattern (DP) */

### How to implement the **OBSERVER** design pattern

**Table of Contents**

- Background Information
- How does this work?
- What next?
Background Information

This simple invoice program will have a graphical user interface in which we show the invoice text in a text area. At the end of this section, you will make the “Add” button responsible for updating the text area. In the next sections, the real strength of this solution will be shown to decouple adding items from the invoice display. Whenever new items are added to the invoice, the program could simply refresh the text area at that time.

The OBSERVER pattern helps us to proceed.
  a. Define an observer interface type. Observer classes must implement this interface type.
  b. The subject maintains a collection of observer objects. In this scenario, the line items are the observer objects.
  c. The subject class supplies methods for attaching observers.
  d. Whenever an event occurs, the subject notifies all observers.

The OBSERVER pattern is to be used because the program needs to update the text area automatically whenever the invoice changes. The above figure shows how this goal can be easily achieved.

How does this work?
/* ... continued with the subsequent steps in implementing the Observer DP with Swing components */

Writing code to implement Composite design pattern (DP)

10. In the Bundle class, create a private reference “items” to class ArrayList, which imported from package java.util.*. You may copy and paste only the boldfaced code to the respective class.

   Class: ‘Bundle’
   import java.util.*; // to provide ArrayList
   public class Bundle implements LineItem
   {
     private ArrayList items;
     // .. continued with subsequent constructor & methods
   }

11. Within the Bundle class, create Bundle class constructor to initialize reference items with an instance of the ArrayList class.

   Class: ‘Bundle’ – Class Constructor: ‘Bundle’
   public class Bundle implements LineItem
   { // .. continued from previous declaration code
     public Bundle() { items = new ArrayList(); } }

Writing code to use Observer design pattern with Swing components

*4. The subject of the Observer design pattern is the Invoice class. In order to attach the observers, supply a
collection of observers. Similar to “items” variable, make use of a private variable “listeners” and the class constructor to initialize an instance of the ArrayList class.

**Tip** (Optional) Details on how to create private variable and class constructor to initialize an instance of the ArrayList class. You may skip this tip if you know how to create and initialize a collection of line items.

5. Next, create a method called `addChangeListener` with parameter `listener`, the change listener to add.

```java
import java.util.*;
import javax.swing.event.*; // to provide ChangeListener

public class Invoice
{
    // .. continued from previous declaration and constructor code

    // Adds a change listener to the invoice.
    // Parameters: listener - the change listener to add
    public void addChangeListener(ChangeListener listener) {
        listeners.add(listener);
    }
    // .. continued with subsequent methods
}
```

Fig. 3. Examples of the documentation fragment, which were presented in both patterns-style and minimalist documentation.

Since the documentation contains multiple files and figures, we adjusted each document set such that the subjects using patterns-style documentation may be given additional background information. The class diagram for the DP is given at the top of the patterns-style documentation. The documentation size is measured in kilobytes, as proposed by Beizer. The following Table 1 gives a picture of relative total length of the documentation, and quantitative characterization in several other dimensions. The key difference between minimalist and patterns-style documentation is the presence of background (context) information, including class diagrams in the patterns-style documentation.

<table>
<thead>
<tr>
<th>Quantitative characterization</th>
<th>Minimalist</th>
<th>Patterns-style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Relative total length (kilobytes)</td>
<td>368 KB</td>
<td>660 KB</td>
</tr>
<tr>
<td>2. The advantage that is missing</td>
<td>Background information</td>
<td>Self-directed reasoning</td>
</tr>
<tr>
<td>3. Number of document files</td>
<td>11 files</td>
<td>16 files</td>
</tr>
<tr>
<td>4. Total sections</td>
<td>15 sections</td>
<td>21 sections</td>
</tr>
<tr>
<td>5. Total paragraphs</td>
<td>17 paragraphs</td>
<td>30 paragraphs</td>
</tr>
</tbody>
</table>

The practical exercise session for the experiment took approximately two hours. Prior to the two-hour session, the students were taught basic object oriented programming principles during the lectures. A digital clock was displayed on the projector for the subjects’ common reference before our experiment began. Our method of observation consisted of a questionnaire with two sections: The first section requested the subjects to record the time for each task, while the second section contains eight questions of various types including multiple-choice, ratings, and free-form questions. Questions raised by more than two subjects were recorded as qualitative findings.
3.3. Experimental Design

Our experimental design uses nine dependent variables and seven independent variables (factors). The main independent variable is the documentation type; while other factors include gender, the three programming grades (see fig. 4), CGPA and background information. The dependent variables are the completion time, specific DP semi-completion time (i.e. Composite, Decorator, Observer and Iterator time), number of difficulties faced, accuracy, workings and comprehension.

- “Documentation Type”: We use two different documentation types, each with a similar purpose: to complete the given work task.
- Dependent variable “completion time”: The time taken to finish the entire exercise. At the end of the fifth work task, the subject was to record this end time.
- Dependent variable “specific DP semi-completion time”: Time taken for the subjects to complete the particular DP, i.e. Composite, Decorator, Observer and Iterator.
- Dependent variable “number of difficulties faced”: Total accumulated problems that the subject encountered. Instead of giving all the detailed steps, some parts of the documentation let the learner interact with the system.
- Dependent variable “comprehension”: The subjects had to identify the method, procedure, line of the code and constants that perform the given task. There were a number of questions to test their understanding of the code.
- Dependent variable “accuracy”: This indicates whether the participant’s solutions fulfilled the requirements of the task or not.
- Dependent variable “workings”: This is to test how well the subjects are able to follow the instructions for creating the default methods or constructors with their modifiers and arguments.

Appendix A presents the test items. The data collected from this experiment is in a discrete manner, either right or wrong for a particular question. This evaluation approach is a kind of examination or exercise, not a survey of opinions. Thus, factor analysis for convergent validity is not required for these direct observable variables. The validation of these variables is well supported with their propositional discrete nature, i.e. the exact total of correct answers and factual nature, such as the exact completion time of various tasks.

3.4. Threats to validity

Internal validity is the degree to which conclusions can be drawn about the casual effect of independent variables on the dependent variable. It is important to point out that weaknesses imposed by these threats to one empirical study can be addressed by the strengths of another. A possible threat is COMPREHENSION (understanding of the exercise), which is a difficult concept to measure. In a single controlled experiment, it is unlikely that different dimensions of a concept can be captured. The researchers must focus on what can be realistically achieved.
External validity is the degree to which the results of the research can be generalized to the population under study and other research settings. The undergraduates knew that analysis would be performed on this data, but were unaware of the experimental hypothesis that was being tested. In this study, the previous programming course grades of the students are categorized into grade ‘A’ (four points) for highest achievement, ‘B’ (three points) for average, ‘C’ (two points) for poor, F (one point) for fail and ‘None’ (0 point), if the student has not taken the course at all. In addition, the grade intervals were considered as well, i.e. ‘A-’ (3.67 points), ‘B+’ (3.33 points), ‘B-’ (2.67 points) and ‘C+’ (2.33 points). From Fig. 4, the subjects in this experiment were a good representative sample of the total population for DP beginners, since most likely beginners are usually in the average grade group. This reaffirms the homogeneity of the subjects being compared.

Table 2. Pearson Chi-square tests on programming language grades and CGPA for the documentation groups.

<table>
<thead>
<tr>
<th>Programming language: Data Struct.</th>
<th>C++ grade</th>
<th>C grade</th>
<th>CGPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-values:</td>
<td>0.148</td>
<td>0.267</td>
<td>0.138</td>
</tr>
<tr>
<td></td>
<td>0.736</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Furthermore, the Pearson Chi-Square tests in Table 2 show that there is no significant difference detected, with all the p-values > 0.05. Consequently, it can be concluded that the threats to internal and external validity were not significant in this experiment. The students had similar knowledge and were required to fulfill certain prerequisites before they were able to undergo the course. Thus, the threats regarding the random
heterogeneity of the subjects are limited.

4. Data Analysis and Findings

The proposed novice behavior model in section 2 is being evaluated to see if one of the documentation sets let the subjects compile a particular design pattern (specific DP TIME) and finish the fastest (COMPLETION) with the number of difficulties recorded by the subject at these intervals (NUMBER OF DIFFICULTIES), as well as understand the most (COMPREHENSION). We also checked for the correlation to the self-tested scores of how well the documentation taught them to use DP to build an application (ACCURACY) and test scores on how well their knowledge in the inner workings the framework (WORKINGS) was. Since we did not want to rely on the assumption of normal distribution, we tested for the normality of the dependent variables.

Insert table showing results of normality test HERE

From the normality test, we discovered that all dependent variables except NUMBER OF DIFFICULTIES are normally distributed for each subject groups. For this dependent variable, medians will be used as the expected values, rather than the means, as shown in Table 3.

<table>
<thead>
<tr>
<th>Category (Dependent variable)</th>
<th>Mean</th>
<th>Std.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Composite time (hh:mm:ss)</td>
<td>0:18:51</td>
<td>0:13:18</td>
</tr>
<tr>
<td>2. Decorator time (hh:mm:ss)</td>
<td>0:40:27</td>
<td>0:18:48</td>
</tr>
<tr>
<td>3. Observer time (hh:mm:ss)</td>
<td>1:08:13</td>
<td>0:21:03</td>
</tr>
<tr>
<td>4. Iterator time (hh:mm:ss)</td>
<td>1:14:35</td>
<td>0:18:19</td>
</tr>
<tr>
<td>5. Completion (hh:mm:ss)</td>
<td>1:22:24</td>
<td>0:11:58</td>
</tr>
<tr>
<td>6. Comprehension (scale: 0 – 6)</td>
<td>2.83</td>
<td>2.135</td>
</tr>
<tr>
<td>7. Accuracy (scale: 0, 1, 2, 3)</td>
<td>1.07</td>
<td>1.172</td>
</tr>
<tr>
<td>8. Workings (scale: 0 – 11)</td>
<td>3.13</td>
<td>4.447</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category (Dependent variable)</th>
<th>Median</th>
<th>Std.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9. Number of difficulties</td>
<td>1.00</td>
<td>7.067</td>
</tr>
</tbody>
</table>

4.1. Correlations and multivariate analyses of variance (MANOVA)

The results of MANOVA are presented in the following Table 4 and 5. When the correlations were computed separately for each dependent variable, the inter-correlations were significant across nearly all categories, as shown in Appendix B to save space.

Table 4. Wilks’ Lambda and F test for the multivariate effect (documentation type).
In order to determine whether any of the categories differed on any of the scales for the dependent variables, mean scores (and standard deviations) were computed for each category on each scale. Using the documentation type as the independent variable and the eight dependent measures, the data were subjected to a multivariate analysis of variance (MANOVA). Table 4 shows the multivariate effect for documentation type on patterns-style and minimalist performance of DP pedagogy practice. These results, using Wilkes’ Lambda = 0.492, $F(8, 70) = 9.033$ ($p < .001$), indicated highly significant differences among the mean scores.

Table 5. Multivariate effects of the documentation type on four specific DP time, Completion, Comprehension, Accuracy, and Workings.

<table>
<thead>
<tr>
<th>No.</th>
<th>Categories</th>
<th>$F$</th>
<th>$\eta^2$</th>
<th>No.</th>
<th>Categories</th>
<th>$F$</th>
<th>$\eta^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Composite</td>
<td>04.973</td>
<td>0.061</td>
<td>5</td>
<td>Completion</td>
<td>01.346</td>
<td>0.017</td>
</tr>
<tr>
<td>2</td>
<td>Decorator</td>
<td>16.993 **</td>
<td>0.181</td>
<td>6</td>
<td>Comprehension</td>
<td>33.869 **</td>
<td>0.305</td>
</tr>
<tr>
<td>3</td>
<td>Observer</td>
<td>11.074 **</td>
<td>0.126</td>
<td>7</td>
<td>Accuracy</td>
<td>63.295 **</td>
<td>0.451</td>
</tr>
<tr>
<td>4</td>
<td>Iterator</td>
<td>06.325 *</td>
<td>0.076</td>
<td>8</td>
<td>Workings</td>
<td>19.680 **</td>
<td>0.204</td>
</tr>
</tbody>
</table>

Note: ** $p < 0.01$ (2-tailed); * $p < 0.05$ (2-tailed)

Table 5 displays the results of the separate multivariate tests. Multivariate F-tests were conducted to determine which of the dependent variables differed across the various categories. Post hoc tests are not performed for the documentation type because the treatments in this experiment are fewer than three groups. From these results, we observed that six out of eight independent variables are significant.

Among the dependent variables on specific DP TIME and COMPLETION in Table 5, we see that the treatments in DECORATOR and OBSERVER TIME make strong significant differences at the 0.01 level; while ITERATOR TIME exhibits a significant difference at the 0.05 level. The patterns group subjects completed each interval of the experiment faster than the minimalist group. Therefore, we conclude that there is a significant difference between the two documentation types, especially as to how long it takes for the subjects to complete the decorator, observer and iterator DP. The patterns documentation significantly helps novices to perform faster in the middle stage of the work tasks.

In terms of COMPREHENSION, ACCURACY and WORKINGS, the subjects in the patterns group exhibited significantly higher scores than the minimalist group. Interestingly, there are strong significant differences in these three dependent variables at 0.01 level, which mean that the $H_0$ hypothesis in section 3.1 is rejected. These rejections show that the two documentation types are not the same in teaching the subjects about DP, the inner workings, as well as completing the work tasks correctly.
Pertaining to **NUMBER OF DIFFICULTIES**, we use the non-parametric Mann-Whitney test since this dependent variable was not normally distributed. Since the two-sided asymptotic significant value in Table 6 for the experiment is more than 0.05, this means that there was no significant difference in the number of difficulties faced by the subjects between the two groups. This could be due to the experiment being conducted near the end of the semester. The subjects’ learning might have reached the maturation effect since they had undergone ten weeks of tutorials and lectures on object-oriented programming prior to this design patterns experiment.

### 4.2. Regression model for future prediction

In order to validate further the various points thus far, let us build the regression model. We extract the models to predict future data trends for continuous valued functions with the assumption that the determinant is linearly related to the factors. The proposed model for regression testing is explained by Gujarati, which can be denoted by the following basic form:

\[ M = c + a_1x_1 + a_2x_2 + \ldots + a_ix_i + e \quad (1) \]

where \( M \) is the determinant, \( x_i \) denotes factor, \( c \) and \( a_i \) are parameters to be estimated, and \( e \) is the error term.

<table>
<thead>
<tr>
<th>Category / Beta (Std. Error)</th>
<th>Compr.</th>
<th>Accuracy</th>
<th>Workings</th>
<th>Decorator Time</th>
<th>Completion Time</th>
<th>(Number of Difficulties)**</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-1.327</td>
<td>-0.681</td>
<td>*** -9.415</td>
<td>*** 5027.60</td>
<td>*** 6671.58</td>
<td>*** 15.942</td>
</tr>
<tr>
<td>(0.809)</td>
<td>(1.524)</td>
<td>(3.852)</td>
<td>(696.615)</td>
<td>(726.546)</td>
<td>(4.923)</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>-0.252</td>
<td>-0.734</td>
<td>0.327</td>
<td>214.098</td>
<td>286.283</td>
<td>* 2.916</td>
</tr>
<tr>
<td>(0.242)</td>
<td>(0.457)</td>
<td>(1.154)</td>
<td>(208.680)</td>
<td>(217.646)</td>
<td>(1.475)</td>
<td></td>
</tr>
<tr>
<td>Data Structure &amp; Algo.(DataStruct)</td>
<td>-0.071</td>
<td>0.038</td>
<td>0.528</td>
<td>12.607</td>
<td>19.871</td>
<td>1.015</td>
</tr>
<tr>
<td>C++ Comp. Prog.</td>
<td>-0.250</td>
<td>-0.140</td>
<td>0.440</td>
<td>-0.849</td>
<td>* -406.196</td>
<td>-2.056</td>
</tr>
<tr>
<td>(0.262)</td>
<td>(0.494)</td>
<td>(1.249)</td>
<td>(225.836)</td>
<td>(235.539)</td>
<td>(1.596)</td>
<td></td>
</tr>
<tr>
<td>C++ Comp. Prog.</td>
<td>0.334</td>
<td>0.206</td>
<td>0.349</td>
<td>-168.262</td>
<td>119.192</td>
<td>1.728</td>
</tr>
<tr>
<td>(0.212)</td>
<td>(0.399)</td>
<td>(1.010)</td>
<td>(182.614)</td>
<td>(190.460)</td>
<td>(1.291)</td>
<td></td>
</tr>
<tr>
<td>CGPA</td>
<td>* 0.477</td>
<td>* 0.890</td>
<td>* 2.399</td>
<td>*** -640.19</td>
<td>* -447.069</td>
<td>** -3.654</td>
</tr>
<tr>
<td>(0.254)</td>
<td>(0.478)</td>
<td>(1.209)</td>
<td>(218.544)</td>
<td>(227.934)</td>
<td>(1.544)</td>
<td></td>
</tr>
<tr>
<td>Background Info.</td>
<td>***1.349</td>
<td>*** 1.810</td>
<td>*** 3.265</td>
<td>*** -531.21</td>
<td>-1.534</td>
<td>** -3.243</td>
</tr>
<tr>
<td>Level (BkgInfo)</td>
<td><strong>0.215</strong></td>
<td><strong>0.405</strong></td>
<td><strong>1.024</strong></td>
<td><strong>185.117</strong></td>
<td><strong>193.071</strong></td>
<td><strong>1.308</strong></td>
</tr>
<tr>
<td>R-square</td>
<td>0.480</td>
<td>0.375</td>
<td>0.288</td>
<td>0.391</td>
<td>0.309</td>
<td>0.321</td>
</tr>
</tbody>
</table>

Note:

*** significant at 0.01 level; ** significant at 0.05 level; * significant at 0.10 level
In Table 7, we reproduce the data of Table 4 and 5 to analyze many more factors with the various dependent variables. Based on Table 7, we obtain the following regressions.

**COMPREHENSION**
\[
-1.327 - 0.252 \text{(Gender)} - 0.071 \text{(DataStruct)} \\
- 0.250 \text{(CPP)} + 0.334 \text{(CLang)} + 0.447 \text{(CGPA)} + 1.349 \text{(BgInfo)}
\] (2)

**ACCURACY**
\[
-0.681 - 0.734 \text{(Gender)} - 0.038 \text{(DataStruct)} \\
- 0.140 \text{(CPP)} + 0.206 \text{(CLang)} + 0.890 \text{(CGPA)} + 1.810 \text{(BgInfo)}
\] (3)

**WORKINGS**
\[
-9.415 + 0.327 \text{(Gender)} + 0.528 \text{(DataStruct)} + 0.440 \text{(CPP)} \\
- 0.349 \text{(CLang)} + 2.399 \text{(CGPA)} + 3.265 \text{(BgInfo)}
\] (4)

\[
\text{(NUMBER_OF_DIFFICULTIES)}^{0.5} = 15.942 + 2.916 \text{(Gender)} \\
+ 1.015 \text{(DataStruct)} - 2.056 \text{(CPP)} + 1.728 \text{(CLang)} \\
- 3.654 \text{(CGPA)} - 3.243 \text{(BgInfo)}
\] (5)

The regressions in Table 7 show that all the mentioned factors, i.e. gender, the three programming grades, CGPA and the level of background information reasonably explain the variations, e.g. 48% in comprehension, 37% in accuracy, 28% in workings and more than 30% for Decorator time, completion and number of difficulties. To obtain the level of background information, we assigned level one for minimalist and level two for patterns-style. The higher the level, the more background information is provided. In Eq. (2), (3) and (4), not only is the background information variable statistically strongly significant at a 0.01 level, but it also has encouraging positive coefficients. Thus, the amount of comprehension, accuracy and workings increased with the level of background information. Next, we transform the number of difficulties into the format of square root so that the normality assumption is valid. For Eq. (5), we discovered that both the background information and CGPA variables are significant at the standard level of 0.05 with negative coefficients. All these overwhelming results support that the background information increases the scores of comprehension, accuracy and workings, while it decreases number of difficulties faced by the neophytes. This is consistent with the earlier findings in section 4 that patterns-style mean scores are higher than minimalist group, as depicted in Table 3. The complexity of using five design patterns requires the availability of background information with UML class diagrams. Telling merely the instructional steps to the novices is not sufficient in completing the five DP task. The novices need to know the background of the DP before able to perform well in accomplishing the task.

5. Conclusions and Discussions

This empirical investigation focuses on teaching design patterns within framework documentation models. Our motivation comes from the fact that each documentation style has different constraints and limitations as well as philosophies on how people will learn DP. We conclude that the patterns-style is more suitable for situations like this.

In this study, we collected empirical data about the subjects, storing them in SPSS (Statistical Package for Social Science) repositories. The repositories equipped us with
the needed information about the subjects’ performance – both on those who failed to complete the given work task using a particular documentation type and on those who were able to complete the task. From this experiment, many elements could be considered to construct a classification tree.\textsuperscript{3} The classification tree would be able to help one to make decisions on which type of documentation style should be used for the most effective learning effect. At each level of the tree, nodes will split data into groups until they reach the ending node, where the documentation style is determined.

The first guideline that formulates our conclusion is having a prediction paradigm. Using the means in Table 3, the following guiding rule could be formulated to govern a systematic classification for construction of a decision tree.

If \textit{target-completion\_time} < 1 hour 20 minutes \& \textit{Comprehension\_required} $\geq$ 5/6 \& \textit{Accuracy\_required} $\geq$ (2.5)/(3.0) and \textit{Workings\_required} $\geq$ 7/11, then use patterns-style documentation; otherwise go for minimalist documentation to save time in developing the documentation materials.

\textbf{(5.1)}

In such a simple guiding rule in model (5.1), the two classes of documentation type for a framework are clearly separated. In addition to the rule governing the classification, it is easy to deduce what would happen in more complex situations, when a systematic technique for tree construction in decision making is required. The experiments have shown that the patterns-style documentation is suited for teaching design patterns. The following model (5.2) generates rules for decision making.

If the task is of high complexity \& the framework is coding-intensive like Swing, then use patterns-style documentation;

// supported by observation in this paper

If the task is of medium complexity,

if the speed in completing task is emphasized, then use minimalist documentation;

else use patterns-style documentation to help novices to understand better.

// supported by VB\textsuperscript{20} and Habanero\textsuperscript{11} observations

\textbf{(5.2)}

Pressman\textsuperscript{33} and Fowler\textsuperscript{14} emphasize that patterns are not always applicable. One would need to adapt patterns to one’s own environment. Through empirical work, one would be able to surmise whether it contains the right information presented in the right way. In a much more complex problem, such as this one involving five DP within the Swing framework, our empirical results suggest that patterns-style documentation exhibits better effectiveness than minimalist documentation. Meanwhile, in the earlier VB experiment\textsuperscript{20} that involved an easier task without DP, we found that the subjects who used minimalist documentation completed their first compilation significantly faster than the ones using patterns documentation. This is also compatible with the Habanero experiment.\textsuperscript{11} It was of medium complexity and found that minimalist group faster and patterns-style helps novices to understand better. Minimalism helped in a simpler task, such as creating a prototype basic payroll program. However, patterns-style documentation helps novices to get the complicated task such involving the five DP, done
faster and more accurate. As such, we conclude different documentation methods are better for different goals.

The application and analysis of practicing the five DP were demonstrated through documentation philosophies based on the regression modeling technique. We have illustrated the effectiveness of the proposed novice behavior model with a wide range of possible measures, e.g. comprehension, accuracy, workings, number of difficulties faced and completion time. A software development team can utilize these documentation guidelines in conjunction with any existing design patterns, to build a model that is best suited for achieving their pedagogy practices. Another related future work is to investigate other alternatives to the UML class diagrams. How would other kind of diagrams, e.g. use case maps⁹ and entity relationship diagram²³,²⁴ affect things? Would they work better in helping application programmers understand the framework in order to use it? We intend to investigate these matters further.

Acknowledgments

The authors are grateful to many people for making this empirical work possible, particularly the course tutors and their experimental subjects. The inputs of the following individuals in the various stages of the study are gratefully acknowledged: Goon Y-Kong, Nathar Shah Packier, Neoh Kee Lin (Multimedia University, Malaysia) and especially Ho Chin Kuan for discussions on the models that were proposed in this article.

Appendix A. Items from the exercise on five DP Pedagogy Practice

Note: (Qn) refers to the original question number in the exercise. The dependent variables are numbered with prefix ‘Y’, while the demographic characteristics are numbered with prefix ‘X’.

Section 1: Documentation on the Invoice Program

Y1 to Y5, Y9: Specific DP end time, Completion time and the Number of Difficulties

Please record time as ‘hh:mm:ss’:

⚠️ Record Start Time: ______________

Y9.1: Number of difficulties faced: ________

⚠️ Y1: 1st-Composite end time: __________

Y9.2: Number of difficulties faced: ________

⚠️ Y2: 2nd-Decorator end time: __________

Y9.3: Number of difficulties faced: ________

⚠️ Y3: 3rd-Observer end time: __________
Section 2: Tutorial Exercise on the Invoice Program

Y6: Comprehension (Understanding of the exercise)

Y6.1: Which class implements the ‘LineItem’ interface? You may select more than one answer.
- A. Bundle
- B. DiscountedItem
- C. Invoice
- D. InvoiceTest
- E. SimpleFormatter
- F. Product

Y6.2: Which class implements the ‘InvoiceFormatter’ interface?
- A. Bundle
- B. DiscountedItem
- C. Invoice
- D. InvoiceTest
- E. SimpleFormatter
- F. Product

Y6.3: The lowest total price for 4 hammers & 2 boxes of assorted nails: RM ________

Y6.4: With discount of 25% for Bundle and other prices fixed, total cost for 2 hammers & 1 box of assorted nails: RM ________ (Hint: Change a value at InvoiceTest.java and compile)

<table>
<thead>
<tr>
<th>No.</th>
<th>Variables</th>
<th>Correct solutions</th>
<th>Discrete scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Y6.1 (Q3)</td>
<td>A. Bundle, B. Discounted Item, F. Product</td>
<td>(0 – 3)</td>
</tr>
<tr>
<td>2.</td>
<td>Y6.2 (Q4)</td>
<td>E. SimpleFormatter</td>
<td>(0 or 1)</td>
</tr>
<tr>
<td>3.</td>
<td>Y6.3 (Q6)</td>
<td>RM 75.10</td>
<td>(0 or 1)</td>
</tr>
<tr>
<td>4.</td>
<td>Y6.4 (Q7)</td>
<td>RM 32.95</td>
<td>(0 or 1)</td>
</tr>
</tbody>
</table>

Y7: Accuracy

Y7.1: From your code, the price of one hammer: RM ________________

and the price for one box of Assorted Nails only: RM ________________

Y7.2: In your program, with one hammer and one bundle of hammer plus assorted nails, Total Due/price: RM _____

<table>
<thead>
<tr>
<th>No.</th>
<th>Variables</th>
<th>Correct solutions</th>
<th>Discrete scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Y7.1 (Q5)</td>
<td>RM 15.70, RM 7.30</td>
<td>(0 – 2)</td>
</tr>
<tr>
<td>2.</td>
<td>Y7.2 (Section1)</td>
<td>RM 37.55</td>
<td>(0 or 1)</td>
</tr>
</tbody>
</table>

Y8: Workings

Y8: List the methods/constructors that you created in Bundle, Invoice and SimpleFormatter class with their modifier and arguments:
An Empirical Investigation of Methods for Teaching Design Patterns

<table>
<thead>
<tr>
<th>Number</th>
<th>Class</th>
<th>Method/Constructor</th>
<th>Modifier/State</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.g.0</td>
<td>InvoiceTest</td>
<td>main</td>
<td>public static void</td>
<td>String args[ ]</td>
</tr>
<tr>
<td>E.g.1.</td>
<td>Bundle</td>
<td>Bundle</td>
<td>public</td>
<td>none</td>
</tr>
<tr>
<td>1.</td>
<td>Bundle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th>Correct solutions</th>
<th>Discrete scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y8 (Q8)</td>
<td>01. Bundle; toString; public String; none</td>
<td>(0 – 11)</td>
</tr>
<tr>
<td>02. Bundle; getPrice; public double; none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03. Bundle; add; public void; LineItem item</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04. Invoice; addChangeListener; public void; ChangeListener listener</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05. Invoice; addItem; public void; LineItem item</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06. Invoice; getItems; public Iterator; none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07. Invoice; format; public String; InvoiceFormatter formatter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08. SimpleFormatter; formatHeader; public String; none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09. SimpleFormatter; formatFooter; public String; none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. SimpleFormatter; formatLineItem; public String; LineItem item</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Invoice; Invoice; public; none</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FX: Demographic Characteristics**

X1: Gender*: Male / Female

Legend: * Please circle one of the items above.

X2 to X4: What grade do you obtain for the following subjects: (state your answer as far you can recall)

X2: Data Structures and Algorithm

- A+/A  
- A-  
- B+  
- B  
- B-  
- C+  
- C  
- F  
- None

X3: Computer Programming II (C++)

- A+/A  
- A-  
- B+  
- B  
- B-  
- C+  
- C  
- F  
- None

X4: Computer Programming I (C)

- A+/A  
- A-  
- B+  
- B  
- B-  
- C+  
- C  
- F  
- None

X5: Indicate your CGPA thus far: ________________________________.

<table>
<thead>
<tr>
<th>Variables (Original question number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1 (Header); X2 (Q1-i); X3 (Q1-ii); X4 (Q1-iii); X5 (Q2)</td>
</tr>
</tbody>
</table>
Appendix B. Correlation Matrix

<table>
<thead>
<tr>
<th>Categories:</th>
<th>Y₁</th>
<th>Y₂</th>
<th>Y₃</th>
<th>Y₄</th>
<th>Y₅</th>
<th>Y₆</th>
<th>Y₇</th>
<th>Y₈</th>
<th>X₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y₁ Composite</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₂ Decorator</td>
<td>.744**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₃ Observer</td>
<td>.508**</td>
<td>.698**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₄ Iterator</td>
<td>.486**</td>
<td>.642**</td>
<td>.891**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₅ Completion</td>
<td>.424**</td>
<td>.554**</td>
<td>.750**</td>
<td>.869**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₆ Comprehension</td>
<td>- .329**</td>
<td>- .428**</td>
<td>- .403**</td>
<td>- .311**</td>
<td>- .184</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₇ Accuracy</td>
<td>- .390**</td>
<td>- .463**</td>
<td>- .475**</td>
<td>- .393**</td>
<td>- .239*</td>
<td>- .711**</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y₈ Workings</td>
<td>- .321**</td>
<td>- .443**</td>
<td>- .344**</td>
<td>- .365**</td>
<td>- .333**</td>
<td>.591**</td>
<td>.599**</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>X₀ Doc. Type</td>
<td>- .246*</td>
<td>- .425**</td>
<td>- .355**</td>
<td>- .276*</td>
<td>- .131</td>
<td>.553**</td>
<td>.672**</td>
<td>.451**</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Note: ** Correlation is significant at the 0.01 level (2-tailed);
* Correlation is significant at the 0.05 level (2-tailed).

References

An Empirical Investigation of Methods for Teaching Design Patterns

15. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software Elements of Reusable Object-Oriented Software* (Addison-Wesley, Reading, MA, 1994) (Commonly called the “Gang of Four” or “GoF” book).
35. Swing framework – Minimalism: how to develop program using five design patterns, Online available: http://pesona.mmu.edu.my/~sbho/Swing/Pattern/t10Pat/201composite.htm