International Journal of Information Science 2013, 3(1): 13-23
DOI: 10.5923/j.ijis.20130301.03

Leveraging Framework Documentation Solutions for Intermediate Users in Knowledge Acquisition

Sin-Ban Ho[1,*], Ian Chai[2], Chuie-Hong Tan[1]

[1]Faculty of Computing and Informatics, Multimedia University, Jalan Multimedia, 63100, Cyberjaya, Selangor, Malaysia
[2]Faculty of Engineering, Multimedia University, Jalan Multimedia, 63100, Cyberjaya, Selangor, Malaysia

Abstract Frameworks are increasingly employed as a useful way to enable object-oriented reuse. However, understanding frameworks is not easy due to their size and complexity. Previous work concentrated on different ways to document frameworks, but it was unclear which ones actually were better. This paper presents a novel way of investigating the different philosophies for framework documentation. The philosophies include minimalist, patterns-style and extended javadoc (Jdoc) documentation. Using a survey of 90 intermediate users engaged in Command and Adaptor design patterns coding work, this exploratory study discovered that minimalist documentation has positive impacts in encouraging knowledge acquisition, significantly in terms of the framework functional workings. This concludes that documentation solutions with the minimalist principle can lead intermediate users to faster growth in learning two of the design patterns.

Keywords Framework Documentation, Knowledge Acquisition, Patterns

1. Introduction

One of the key challenges to object-oriented frameworks is introducing the design patterns to intermediate users. Intermediate users are those who have already had some experience with the framework in question but not yet experts, i.e. they are between the novice and advanced levels. The subjects would perform the coding details of a particular portion of the code while the instructor ensures that the coding exercise is being followed with the help from the check-point time made available in the documentation.

This paper reports and discusses results from an empirical study on framework documentation. This practice populates a documentation model with the necessary technical and development how-to's to get the task done[1]. The general problem of how to document framework is large. The scope of this research work is to tackle intermediate user documentation or tutorials.

2. Motivations of the Study

One of the earliest works on empirical study in software environments is the goal/question/metric (GQM) goal template proposed by Basili and Rombach[2]. There are tendencies where instantiated goals show certain similarities. The purpose of using GQM paradigm is to refine the goals into quantifiable, reducing complexity and putting in knowledge learned from previous experiments. Basili et al. [3] provide the following five parameters in a GQM goal template:

(a) Object of study: a process, product or any other experience model.

(b) Purpose: to characterize (what is it?), evaluate (is it good?), predict (can one estimate something in the future?), control (can one manipulate events?), improve (can one improve events?)

(c) Focus: model aimed at viewing the aspect of the object of study that is of interest, such as reliability of the product, defect detection/prevention of the process, accuracy of the cost model.

(d) Point of view: the perspective of the person needing the information, e.g. in theory testing the point of view is usually the researcher trying to gain some knowledge.

(e) Context: models aimed at describing the environment in which the measurement is taken.

Selecting a particular type of process for study, the GQM template then becomes: *Analyse framework documenting techniques* to *evaluate* their *effectiveness on a product* from the point of view of the *knowledge builder* in the context of a particular *domain*. For some widely-used frameworks like Swing[4], educators may write more easily-understood documentation to teach less-experienced programmers.

Studies in pedagogical documentation show that the behaviour in organising a programming guide is a domain that has been used to describe the manner how beginners learn how to use a framework. For some time, studies have reported behaviour differences in pedagogical framework documentation. The three philosophies being evaluated in

* Corresponding author:
sbho@mmu.edu.my (Sin-Ban Ho)
Published online at http://journal.sapub.org/ijis

this study include minimalist[5], patterns-style[6,7] and extended javadoc documentation[8,9]. Each is compatible with the idea of mixing texts, examples and diagrams.

John Carroll's innovation, minimalist documentation, is based on the idea that people do not want information irrelevant to the task at hand. This idea attempts to give the reader the minimal amount of information to get the task done, and arrange it in short pages or index cards of information so that users can read in whatever order suits them[5]. As each minimalist page or card contains little information, they often refer to other pages or cards. Hence, they lend themselves well to hypertext presentations like the Web. Carroll gives these guidelines for minimalist documentation:

- Training on real tasks: people are more motivated to do an exercise when it relates directly to something useful they want to do.
- Getting started fast: if there is too much to read before readers get to typing something on the computer, they will lose interest and miss things.
- Reading in any order: topics are brief and allow readers to choose whatever order seems best to them.
- Coordinating system and training: instead of giving all the detailed steps, let the learner interact with the system.
- Supporting error recognition and recovery: instead of giving step-by-step instructions that assume readers will repeat flawlessly, expect them to fail and give them the resources to understand how to recover.
- Exploiting prior knowledge: instead of using insider jargon, use analogies to readers' prior experience to help them understand.
- Using the situation: take advantage of the expectations learners bring to the situation.

Most documentation focuses on specific computer software. These sources typically tell how people are supposed to perform tasks, not what they actually do[10]. In conjunction with this purpose, one of the objectives of a pattern is to get readers to understand some of the rationale for the solution, so that they can decide when to apply the pattern. A definition commonly used at Pattern Languages of Programs (PLoP) conferences for patterns is:

A pattern is a proven successful solution to a recurring problem in a context.

Patterns lend themselves well to hypertext presentations such as those found on the Web, since they refer to other patterns when a problem or its solution is too big to discuss in one sitting. Meszaros and Doble[11] said that patterns should have these elements:

- Pattern Name: so that people can refer to the pattern.
- Problem: a lengthy description of the problem it solves.
- Solution: there may be different solutions to the same problem depending on the context. A prescription for how it works.
- Context: the circumstances of the problem impose constraints on the solution.
- Forces: often contradictory considerations that must be taken into account when choosing a solution to a problem.

Clements et al.[12] said that the patterns style should consist of partial design solutions found repeatedly in actual practice. As mentioned above, patterns also provide background information (the context) and not just the raw solution. We present this background information first, before the *how does it work* section, where the readers may follow to apply the solution to an actual system. Patterns style addresses the application-specific problem in a specific context. It proposes a development solution that can serve as the basis for teaching intermediate users how to reuse components available within a framework.

Jdoc incorporates the HTML documentation generated by the javadoc tool. The class information, such as inheritance and subclasses are provided at the top of the Jdoc. This is followed by a textual description of the class, constructors and methods, which contain their pseudo code with hypertext links to the particular steps. Erik Berglund[8] centred much of his library communication work on the Java programming language domain and javadoc tool that provides automatic generation of reference documentation from Java source files. The javadoc tool represents the state of the art in automated documentation generation and online reference documentation.

The minimalist documentation only provides the information directly relevant to the task at hand. The patterns documentation in addition provides background information that explains the context in which the solution should be applied. The Jdoc documentation provides classes, inheritance and methods information, which are not found in both minimalist and patterns documentation.

Table 1 summarizes some key points of the documentation philosophies. Each of these philosophies sounds reasonable from their description. Each documentation philosophy is compatible with the idea of testing one's documentation on end-users and making changes based on their reactions. Each of them is open to the idea of mixing text, examples, and diagrams.

Table 1. The overview of documentation philosophies

Doc. Philosophies	Background information description	Level of background information
Minimalist doc.	Let user figure out	Low
Patterns documentation	Give problem context	Medium
Extended javadoc (Jdoc)	Class information	High

What is data in guiding decision making? Data is raw information where collections of fact must be gathered and processed to be meaningful. Associating facts in a given context derives information. Knowledge associates information obtained within one context with other information obtained within a different context[1]. As such, wisdom takes place when disparate knowledge derives generalized principles. The wisdom view to acquire knowledge can be represented schematically in Figure 1, which gives an overview of the work performed in Visual Basic (VB) and Swing experiments[13]. Software

practitioners are concerned with systems that process knowledge. Information gathered from experiments is connected to build a body of fact that is referred as knowledge. The ability to associate information from various sources forms the key to provide one with some distinct advantages.

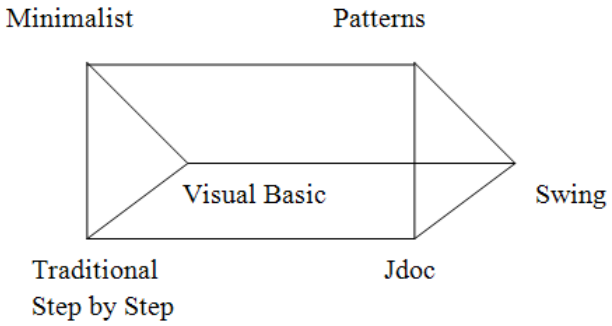


Figure 1. Information spectrum of a knowledge acquisition model

Andrew Forward[14] did a case study using the IBM Eclipse project architecture[15] and found that, to achieve good documentation relevance, one needs to find ways to increase its power, simplicity or preferably both. Lethbridge et al.[16] conducted three studies to see how software engineers use and update documentation. They found that out-of-date software documentation remains useful in many circumstances. Andrew Forward and Timothy C. Lethbridge[17] also compiled evidence that software engineers value technologies such as the javadoc. This is because the automation of the documentation process facilitates its maintenance. In short, Forward and Lethbridge surveyed the usefulness of documentation during maintenance by software engineers.

This paper takes a different approach. Our main research question is to empirically test whether minimalist, patterns documentation or Jdoc presentation would give better performance in teaching intermediate users how to use design patterns. This question is indeed the main concern that is being challenged. In this paper, we use the Command and Adaptor design patterns[18] as the basis of study on the impact of the documentation philosophies.

3. Experiment Description

This research work used an exercise-based research typically used in empirical software engineering. One of the main components of the research methodology is exercise-based investigation, which was preceded with the presentation of a certain documentation set. Overall, it consists of the following four activities:

- **Activity I**: Forming the research question and formulating hypotheses based on the relevant literature.
- **Activity II**: Developing the documentation sets to test the hypotheses.
- **Activity III**: Building the exercise instrument and the collection of data through exercises.
- **Activity IV**: Analysis of the data collected as part of Activity III.

The formulated hypotheses were used to design the documentation sets and the respective exercise, which were pre-tested for usability, soundness, and readability before it was rolled out for collecting data from the field. The data collected were then statistically analysed using suitable data analysis techniques.

3.1. Documentation Procedure

The documentation procedure of Figure 2 is composed of five steps. In the first step, we manually identify which tutorial that would correspond to the work task discussed in the textbook[19]. The textbook provides the rationale for the work task, supporting the scenario in demonstrating the Command and Adaptor (CmdAdp) design patterns.

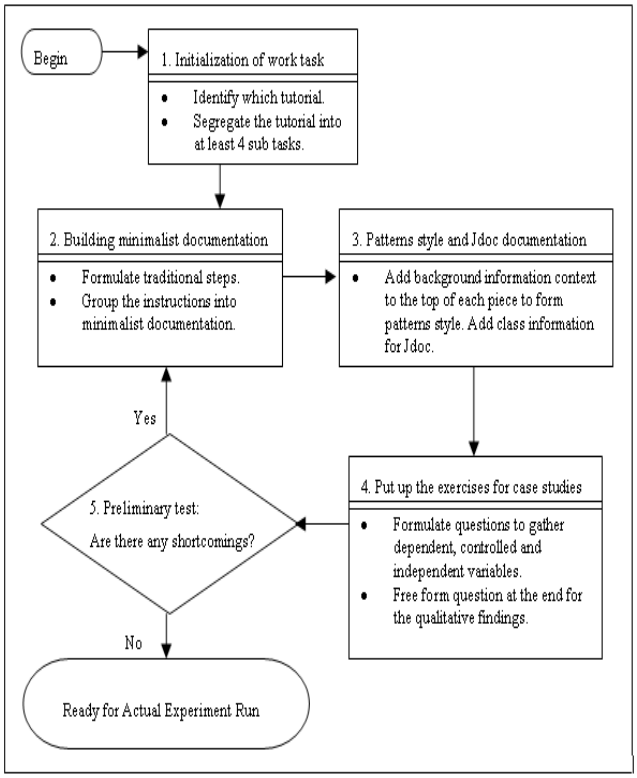


Figure 2. An overview of the documentation procedure

The second step of the documentation is to build minimalist documentation by formulating step-by-step instructions. The instructions are grouped into minimalist documentation. There are two important observations from this step. Firstly, it is less chaotic in writing the traditional steps in one web page before segregating the steps into multiple web pages. Next, instead of merely instructions, some of the steps provide short exercises. The subjects would need to figure out some parameters and code. With this, the subjects have a chance to apply what they learn thus far based on some of the similar previous steps. The content is structured into four work tasks, so that the completion time for each check point could be recorded. The four work tasks include *What components to be put into the content pane first*

for your simple Command DP program, How to implement commands to demonstrate action objects, How to create and show the MapAdaptorTest object, and Writing an adaptor to adapt a Map and populating a SortedMap with the key/value pairs.

The third step of the documentation is to formulate the patterns style and Jdoc documentation. By adding the background information context to the top of each piece, we could include the respective class diagram, with its description, into the patterns style. Meanwhile, the classes, inheritance and methods information, which are obtained from running the javadoc tool, are added on top of the Jdoc documentation. Four work tasks focus on the *Intermediate topics* where these web pages provide links to the related steps in the previous experiment documents. The patterns style and Jdoc examples are therefore deemed sound for testing when putting up the exercise for the case study at hand and it is sufficient to account for gathering dependent, controlled and independent variables through questions on the tutorial identified in step 1. The questionnaire consists of mainly close-ended multiple-choice questions, with an open-ended question at the end. For the close-ended multiple-choice questions, statistical analysis is performed on the available data. Meanwhile, the open-ended questions provide a further dimension in capturing some additional information from the respondents which is not captured in the close-ended questions.

The last step is to conduct a test on the documentation before the actual experiment runs. This includes verifying the correctness of the identified steps by undergoing human trials with at least two testers, at a high level of abstraction, against the exercise on the work task. This step involves the presence of both the author and the tester, as it requires one to understand why the changes were made and verify that the tester can follow the instructional steps. Should there be any shortcomings, the steps are to be noted and changed accordingly prior to the actual experiment. This conforms to the intent of the identified tutorial.

So, what exactly are the participants to do? The participants should follow the documentation and create java source code that import the main Swing package i.e. javax.swing.* and two AWT packages i.e. java.awt.event.* and java.awt.*. The expected result from these tasks is to have an outcome of running Command and Adaptor (CmdAdp) programs. Figure 3 shows an example of the CmdAdp documentation, which is organised into pieces to formulate the minimalist documentation. The background information section is added to the top of each piece in order to form the patterns style (see Figure 4). For Jdoc, the background information is replaced by the output of the javadoc tool, which comprises of the extracted information from the source code about interfaces, methods and data-fields, as shown in Figure 5.

To provide a picture of the relative total length of the documentation, the documentation size is measured in kilobytes, as proposed by Beizer[20]. Through this approach, we can quantitatively characterize the documents. Table 2 gives quantitative information about the character of the documents used in this experiment.

Writing code to implement COMMAND design pattern (DP)

1. Create a java source code file with the name of "**GreetingAction.java**". Notice your filename must start with **uppercase** '**G**'. This is to correspond to the class name, which begins with uppercase letter.

Class: '**GreetingAction**'

```
//GreetingAction.java
import javax.swing.*; // provide JTextArea
import java.awt.event.*; // provide AbstractAction

public class GreetingAction extends AbstractAction
{
    //To continue with subsequent declaration & methods
} // end class
```

The **AbstractAction** class **implements** the **Action interface** type. This **GreetingAction** class **extends** the **AbstractAction class**, rather than implement the Action interface type.

2. At the first line in the *GreetingAction* class, declare two private instance variables to store the string and text area for the greeting. You may copy and paste **only the boldfaced code** to the respective class.

Class: '**GreetingAction**'

```
public class GreetingAction extends AbstractAction
{
    private String greeting; // string for the text area
    private JTextArea textArea;        // text area for the greeting
}
/* ..continued with the subsequent steps in the Swing Intermediate
Topics documentation*/
```

5. Within the *GreetingAction* class, include two methods to set the state of an action. The *Action* interface type extends the *ActionListener* interface type. Thus, you specify the command action in an *actionPerformed* method.

Class: '**GreetingAction**' – Method Summary

```
//Task1: Set the opposite action
public void setOpposite(Action action)
{ // action to be enabled after this action was carried out
    oppositeAction = action;
}
//Task2: Specify the command action in an actionPerformed method
public void actionPerformed(ActionEvent event)
{
    textArea.append(greeting);
    textArea.append("\n");
    if (oppositeAction != null)
    { //setEnabled method to enable or disable an action
        setEnabled(false);
```

```
            oppositeAction.setEnabled(true);
        } //end if
    } //end method actionPerformed
```

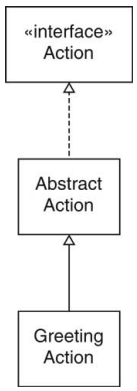
Figure 3. Examples of the documentation fragment which was presented in all the three documentation groups

How to use Swing library to implement *COMMANDS* that can be enabled / disabled

Table of Contents

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

Background Information



This action places a greeting into a text field, and afterwards disables itself and enables its opposite action.
The *AbstractAction* class implement the *Action* interface type. The *AbstractAction* class would be extended rather than implement the *Action* interface type. The above figure shows how this goal can be achieved.

This application demonstrates action objects where two actions insert greetings into a text area. Each action can be triggered by a menu item or toolbar button. When an action is carried out, the opposite action becomes enabled. This *GreetingAction* class would contain two sub-tasks, namely constructing a greeting action, and setting the opposite action.

How does this work?

/* … continued with the subsequent steps in implementing the Command design pattern (DP) */

Figure 4. Example of the documentation fragment that is available in the patterns style documentation, but not available in the minimalist and Jdoc documentation

Package	**Class**	Tree	Deprecated	Index	Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Class GreetingAction

java.lang.Object
  |
  +--javax.swing.AbstractAction
        |
        +--GreetingAction

All Implemented Interfaces:

javax.swing.Action, java.awt.event.ActionListener, java.lang.Cloneable, java.util.EventListener, java.io.Serializable

public class **GreetingAction**
extends javax.swing.AbstractAction

About this program: This action places a greeting into a text field and afterwards disables itself and enables its opposite action.

/* *..continued with the field summary information in the Jdoc documentation* */

Constructor Summary

GreetingAction(java.lang.String greeting, javax.swing.JTextArea textArea)
Constructs a greeting action.

Method Summary

void actionPerformed (java.awt.event.ActionEvent event)
3-2-5: Specify the command action.

void setOpposite(javax.swing.Action action)
3-2-5: Sets the opposite action.

/* *..continued with the methods inheritance detail information* */

Constructor Detail

GreetingAction

public **GreetingAction**(java.lang.String greeting, javax.swing.JTextArea textArea)
Constructs a greeting action.
3-2-3: Provide two arguments with data type **String** and **JTextArea**.

Parameters:

greeting - the string to add to the text area
textArea - the text area to which to add the greeting

Method Detail

setOpposite

public void **setOpposite**(javax.swing.Action action)
3-2-5: Sets the opposite action.

actionPerformed

public void **actionPerformed** (java.awt.event.ActionEvent event)
3-2-5: Specify the command action.

/* *..continued with the subsequent steps in the documentation* */

What next?

1. Proceed to the next program to implement the Adaptor design pattern. The goal is to write an adapter that adapts a Map to an AbstractTableModel.
2. Populate a SortedMap with key/value pairs and show the map inside a JTable.

Package	**Class**	Tree	Deprecated	Index	Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Figure 5. Example of the documentation fragment that is available in the Jdoc documentation, but not available in the minimalist and patterns style documentation

Table 2. Characterize the relative documentation quantitatively

Quantitative characterization	Minimalist	Patterns	Jdoc
1. Relative total length (in kilobytes)	244 KB	293 KB	340 KB
2. Information that is relatively available	Short overview list of work tasks	Background information	Classes, method and interface information
3. Number of document files	10 files	13 files	22 files
4. Total sections in the documentation	9 sections	14 sections	11 sections
5. Total paragraphs in the documentation	17 paragraphs	27 paragraphs	24 paragraphs

In summary, the ultimate objective is to put the theoretical inference to the test: the hypothesis may be proven true or may be different from the actual result gathered during the observation process. As supported by Kallakuri and Elbaum[21], the subsequent empirical experiment run is characterized by investigation through gathering data and performing analyses. This is to determine the meaning of the data, and encompasses the following case study strategies:

- The experiment would provide the investigator with control over some of the conditions in which the study takes place by manipulating independent factors to elicit responses from the dependent factors.
- The observation through case study, which investigates real-life phenomena in the context of a current model.
- A demonstration of documentation strategy on the subjects.

3.2. Hypotheses

Standard significance testing is used to clearly specify the effects of the three documentation philosophies. The null hypotheses are stated as follows.

$E1H_0$ - There will be no difference between patterns and minimalist documentation for the intermediate users in doing the same exercise.

$E2H_0$ – There will be no difference between patterns and Jdoc documentation for the intermediate users in doing the same exercise.

$E3H_0$ – There will be no difference between minimalist and Jdoc documentation for the intermediate users in doing the same exercise.

The interpretations of the experiment are derived from the rejection or non-rejection of these hypotheses for each expectation.

3.3. Participants

There are 90 participants in this study. 33 (36.7%) are female and 57 (63.3%) are male, with the mean years in the university of 2.97, and SD of 0.436, a minimum 2 years and maximum 4 years in the university. Participants are all information technology undergraduates who undergo the object-oriented programming course at the university. The normal age of the students at this level is 22 years old.

To be able to test the hypotheses of our experiment, three different groups of the CmdAdp documentation are required. We arrange the participants into three different groups, according to their tutorial sections. Table 3 shows more detailed information about the groups.

Table 3. The detailed information of years in the university (year) and previous achievement of C Language course (CLang), C++ (CPP), Data Structures and Algorithms (DataStruct) grades, and CGPA

Documentation philosophies	Minimalist	Patterns	Jdoc
N (participants)	26	26	38
Mean (year)	3.08	2.92	2.92
Std. dev. (year)	0.077	0.110	0.058
Mean (CLang)	3.16	2.95	3.16
Mean (CPP)	3.08	3.08	3.17
Mean (DataStruct)	3.04	1.58	1.74
Mean (CGPA)	3.12	3.08	3.20

During the lectures, the students are taught basic object-oriented programming (OOP) principles. The lectures are supplemented by practical tutorial sessions where the students have the opportunity to make use of what they have learned through the completion of various java coding exercises using the assigned on-line documentation. Prior to this experiment, the preliminary stage of the on-line documentation presents the Swing framework. The second stage discusses five of the design patterns[13]. This experiment focuses on the third stage of the intermediate users learning, which is on the CmdAdp design patterns, as summarized in Figure 6. The participants in this experiment are regarded as intermediate users since they have attempted the prior two stages. They are not advanced users since they have not completed the OOP course yet.

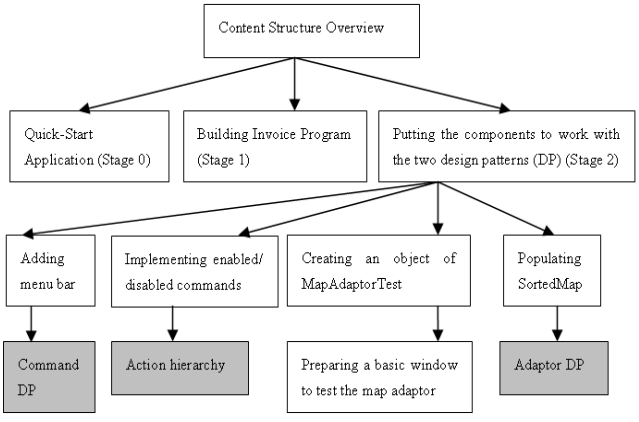


Figure 6. Overview of the content structure. The grey boxes are UML diagrams

3.4. Procedure (Tasks)

The pedagogical documents are developed on a workstation using an html editor such as Dreamweaver. These documents are subsequently uploaded to a web server so that the users can access the on-line documentation. Before the experiment begins, a digital clock is displayed on the projector for the subjects' common reference.

Our method of observation consists of a survey with two sections. The subjects receive this survey printed on paper. The first section requests the subjects to record their completion time after each task, while the second section includes eleven post mortem questions of various types including multiple-choice, ratings and free-form question. The responses to this free-form question raised by more than two subjects are recorded as qualitative findings. These responses are delivered in handwriting. However, the overall amount of the text written is small, so handwriting speed is not a limiting factor.

3.5. Experimental Design

Our experimental design uses one independent variable (factor) and six dependent variables. The independent variable consists of the documentation group. The dependent variables are the completion time, number of difficulties faced, semi completion time, workings and comprehension (understanding of the exercise).

Independent variables:

Documentation type: We use three documentation philosophies, as described in section 1, each with a similar purpose: to complete the given work task.

Dependent variables:

Semi Completion time: Time taken for the subjects to do their first compilation.

Completion time: The time taken to finish the entire exercise.

Comprehension: The subjects have to identify the method, procedure, line of the code, and constants that perform the given task. There are a number of questions to test their understanding of the code.

Workings: This is to test how well the subjects are able to follow the instructions for assigning default settings to the CmdAdp components.

Number of difficulties faced: Instead of giving all the detailed steps, some parts of the documentation let the learners interact with the system. The subjects are to record and accumulate the number of problems they encounter.

Appendix A provides the exercise of the experiment. This gives a more specific description on what the exercise ask for the dependent variables and how they are measured. The data collected from this experiment is discrete, 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[22,23]. The validation of these variables is well supported with their propositional discrete nature[24], i.e. the exact total of correct answers and factual nature, such as the exact completion time of various tasks.

3.6. Validity

Internal validity is the degree to which conclusions can be drawn about the casual effect of independent variables on the dependent variable. To see whether the groups differ significantly, we perform ANOVA tests on the three groups of participants. In Table 4, with all the p-values > 0.05, except for the Data Structures and Algorithms course that they took in the prior semester, there is no major significant difference detected. The random assignments of the three tutorial groups are balanced in terms of their years in the university, the courses like C and C++ language, and Cumulative Grade Point Average (CGPA).

Table 4. The ANOVA tests results on years in the university, C language (CLang), C++ (CPP), Data Structures and Algorithms (DataStruct) grades, and CGPA for the three documentation groups

Categories:	F-values	p-values
Years in the university	1.175	0.314
CLang	1.015	0.367
CPP	1.101	0.337
DataStruct	7.843	0.001**
CGPA	0.499	0.609

Note: ** Statistically significant at 0.05 level

Furthermore, the total completion time of the participants shows an almost perfectly symmetric distribution. Thus, there is no evidence that slower participants hurried because of others having finished before them, in spite of the particular participant group working in the same laboratory at the same time. A final consideration is the precision and accuracy of time stamps recorded by the participants. Although the participants are informed that they have at most two hours to complete the work task, by cross checking, we discover that their responses in the time stamp to be highly accurate and reliable.

External validity is the degree to which the results of the research can be generalised to the population under study and other situation settings. We have identified two possible external threats. Firstly, the participants who take part in the experiment may be not the full representatives of software learners. Due to the constraint in tutorial sections arrangement, all the participants are FIT (Faculty of Information Technology) undergraduates. None are from other faculties, such as management, psychology etc.

4. Data Analysis, Results and Discussion

4.1. Statistical Analysis on the Results

Statistical analyses are conducted using Statistical Package for Social Science (SPSS). The results are based on the sample of 90 responses. The data is analysed to see if one of the documentation sets let the participants compile **(Semi-Completion)** 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 check for test scores on how well their knowledge in the inner workings of the framework **(Workings)**. Since we do not want to rely on the assumption of normal distribution, we test for the normality of the dependent variables. From the normality test in Table 5, we discover that all dependent variables except **Number of difficulties** are normally distributed for each participant group. Thus, for this dependent variable, medians will be used as the expected values, rather than the means, as shown in Table 6 and Table 7. There exists a small outlier and we have checked that the results are resilient to the removal of the outlier.

Table 5. Results of normality test

Category (Dependent variable)	Kolmogorov-Smirnov Z
1. Semi-Completion time	1.100
2. Completion time	1.047
3. Comprehension	1.328
4. Workings	1.270
5. Number of difficulties	2.070**

Note: ** Significant at 0.01 level

Table 6. The means and standard deviations of all categories

Category (Dependent variable)	Mean Min.	Pat.	Jdoc
1. Semi-Completion (hh:mm:ss)	**0:31:06**	0:33:59	0:36:56
2. Completion Time (hh:mm:ss)	**0:58:11**	1:04:29	1:07:03
3. Comprehension (Scale: 0-18)	**14.69**	13.31	14.08
4. Workings (Scale: 0-4)	**3.42**	2.81	2.87

Table 7. The mean rank, medians and standard deviations of the number of difficulties

Documentation type	Minimalist	Patterns	Jdoc
Sample size, n	26	26	38
Median	1.00	1.00	0.00
Mean rank	46.44	39.21	49.16
Standard deviation	2.765	1.531	3.184
Removal of invalid cases*	7	5	12

Note: Those subjects who did not answer the number of difficulties they faced are considered invalid cases.

Table 8. Multivariate effects of the documentation type on Semi-Completion time, Completion time, Comprehension, and Workings

Category	F	Significance
1. Semi-Completion time	1.657	0.197
2. Completion time	2.305	0.106
3. Comprehension	1.077	0.345
4. Workings	4.639	0.012**

Note: ** Statistically significant at 0.05 level;
* Statistically significant at 0.10 level.

In order to determine whether any of the categories differed on any of the scales for the dependent variables, mean scores (and standard deviations) are computed for each category on each scale. Using the documentation type as the independent variable and the four dependent measures, the data are subjected to an analysis of variance. Table 8 presents the results of the separate multivariate tests. Multivariate F-tests are conducted to determine which of the dependent variables differ across the various categories. These values are obtained via tests of between-subjects effects using Multivariate Analysis of Variance (MANOVA) with a Scheffe test adjustment[25]. We choose this test to examine the sample sizes, since the three documentation groups in this experiment are unequal. From these results, we observe that one out of four independent variables is significant.

In terms of **Semi-Completion** and **Completion** in Table 6, the subjects who use minimalist documentation complete their first compilation and complete the experiment faster than the ones using the other two documentation styles. When looking for the standard significance level of 0.05 (i.e. 95% probability) in Table 8, there is evidence that the patterns group are not significantly slower. Therefore, we conclude that there is no significant difference between patterns and the other two documentation styles as to how long it takes the subjects to complete the experiment. Subjects using minimalist are faster than both of the others perhaps because there is less text to read, while subjects using patterns style are faster than subjects using Jdoc perhaps because it is not cluttered with too much class information such as inheritance and subclasses.

As for **Comprehension**, there is no significant difference between how well the subjects understand the materials. This might be because the students are still able to understand the CmdAdp code in the end, irrespective of the document styles. Their learning may reach a maturation effect[26] after going through the four work tasks of documentation. Furthermore, this can be due to the experiment being conducted at the end of the semester. The participants learn enough from the prior eleven weeks of tutorials and lectures on object-oriented programming to bias their performance in the final stage of the experimental run.

Regarding **Workings**, the subjects in the minimalist documentation group exhibit significantly better workings scores than the other documentation styles at the 5 per cent level. Interestingly, this indicates that the $E1H_0$, $E2H_0$ and $E3H_0$ hypothesis in section 3.2 are rejected. These rejections show that the patterns documentation and the other two styles are not the same in teaching the subjects about completing the work tasks with the designated settings. Spending more time in directly instructing the coding of the CmdAdp can be more beneficial in having the default result rather than flooding the intermediate users with too much background information. Too much background information may motivate intermediate users to try something different. They are more confident to differ since they are equipped with the additional background.

Table 9. Kruskal-Wallis test on the number of difficulties

Chi-square	Degree of freedom (DF)	Asymptotic significance
2.502	2	0.286

Since the **Number of difficulties** is not normally distributed over the comparison of the three groups, we use the Kruskal Wallis test[27,28]. With the two-sided asymptotic significant value in Table 9 more than 0.05, the number of difficulties faced by the subjects has no significant difference among the three groups. The

participants might not record fully the number of difficulties they have solved the task. Looking at the results in Table 7, many participants have been removed because they do not answer the questions. In summary, among the strong proxies that confirm minimalist advantages include the fastest semi completion time, the fastest completion time, the highest comprehension and workings scores. Hence, we conclude that minimalist documentation is relatively superior to others in encouraging the positive knowledge transfer strategies of intermediate users.

4.2. Regression Model for Future Prediction

In order to further validate the various points, let us build the regression model[29]. We extract the models to predict future data trends for continuous valued functions[30] with the assumption that the determinant is linearly related to the factors. The proposed model for regression testing is explained by Greene[31] and Gujarati[32], which can be denoted by the following basic form:

$$M = c + a_1x_1 + a_2x_2 + \quad \ldots \quad + a_ix_i + e \qquad (1)$$

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

We further explore the data of Table 8 to analyse many more factors with the various dependent variables. Based on the regression analysis, we obtain the following regressions.

COMPREHENSION = 14.066 – 0.278 *(Gender)* – 0.037 *(DataStruct)* – 0.516 *(CPP)* + 0.507 *(CLang)* + 1.526 (CGPA) – 1.327 (Year) – **0.186 (BgInfo)** (2)

WORKINGS = 1.208 – 0.226 *(Gender)* – 0.012 *(DataStruct)* + 0.071 *(CPP)* – 0.064 (*CLang*) + 0.280 (*CGPA*) + 0.403 (Year) + **0.015 (BgInfo)** (3)

SEMI-COMPLETION TIME = 3596.17 + 163.93 (*Gender*) – 17.94 (*DataStruct*) – 15.48 (*CPP*) + 10.69 (*CLang*) – 293.97 (*CGPA*) – 185.08 (*Year*) – **120.92 *(BgInfo)*** (4)

COMPLETION TIME = 6064.24 + 398.32 (*Gender*) – 9.254 (*DataStruct*) – 96.889 (*CPP*) + 87.59 (*CLang*) – 584.10 (*CGPA*) – 211.62 (*Year*) – **136.31 *(BgInfo)*** (5)

In addition, the regressions indicate that all the mentioned factors, i.e. gender, the three programming grades, CGPA, years in the university and the level of background information reasonably explain the variations (refer to the R-square values), e.g. 10% in comprehension, 8% in workings, 7% in semi-completion time, and 15% in completion time. To obtain the level of background information, we assigned level one for minimalist, level two for patterns-style and level three for Jdoc. The higher the level, the more background information is provided. In Eq. (2), (4) and (5), the background information variable has negative coefficients. Thus, the amount of comprehension, semi-completion time and completion time inversely relate to the level of background information. For Eq. (3), we discover that the background information with positive coefficient. Therefore, the workings increase with the level of background information. These results support that less background information helps achieving faster semi-completion time, faster completion time, and increases the scores of comprehension. Thus, the use of minimalist documentation can be beneficial to the users for a simple task such involving only two of the design patterns.

5. Conclusions

In this work, a set of philosophies for organizing pedagogical textual and graphical information on the CmdAdp documentation has been proposed. This work reveals the missing salient variable in the recent individual differences study by Graff[33], i.e. the time users spent at each page of hypertext. From the results, we realize that the effects of the patterns style documentation are not supreme all the time. Perhaps, for intermediate users, patterns are not always the best. Furthermore, Pressman[1] suggested that patterns are not suitable for every situation. Interestingly, minimalist documentation shows an overwhelming advantage in terms of the intermediate users' completion speed and comprehension in fulfilling requirements.

The quantitative results show that minimalist documentation did not have a significant impact on the time and comprehension that it took to perform the programming tasks. Nevertheless, in terms of the functional workings of the framework, minimalist documentation had a practically and significantly positive impact, in spite of the fact that the participants were not experts in applying design patterns into programming tasks. The aim of using the most effective documentation is to provide intermediate users with a good process that will lead to faster growth in learning the CmdAdp design patterns. All these results demonstrate the behaviours of CmdAdp intermediate users in using pedagogical framework documentation.

ACKNOWLEDGEMENTS

The study described in this paper would not be possible without the cooperation and willingness of the experimental subjects and course tutors. The authors also thank the anonymous reviewers for their valuable suggestions that improved the paper.

APPENDIX

the exercise items

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 Command and Adaptor Program

Y0 to Y2, Y5: Check point time, Completion time and the Number of Difficulties

Please record time as ‘hh:mm:ss’:

⚠ Record Start Time: _____

Y5.1: Number of difficulties faced: _____

⚠ Y0a: Command *Quarter* end time: _____

Y5.2: Number of difficulties faced: _____

⚠ Y1: Command *Semi* end time: _____

Y5.3: Number of difficulties faced: _____

⚠ Y0b: Adaptor 2nd *Quarter* end time: _____

Y5.4: Number of difficulties faced: _____

⚠ Y2: Congratulations! Completion Time: _____

Section 2. Tutorial Exercise on the Command and Adaptor Program

Y3: Comprehension (Understanding of the exercise)

Y3.1: In **Command** DP code, indicate the **respective package** that provides the listed **class/interface**. Answer with numbering: **1**. javax.**swing**.*, **2**. java.**awt**.* or **3**. java.awt.**event**.*

No.	*Class*	*Package (Pkg.)*	*No.*	*Class*	*Pkg.*
Eg.	Action	**1.** javax.**swing.***	5.	JFrame	
1.	JTextArea		6.	Container	
2.	ActionEvent		7.	JMenu	
3.	JMenuBar		8.	JToolBar	
4.	AbstractAction		9.	ImageIcon	

Y3.2: In **Adaptor** DP code, indicate the **respective package** that provides the particular **class**. Answer with respective numbering: **1**. javax.**swing**.*, **2**. java.**awt**.* **3**. java.**util**.* or **4**. javax.**swing**.**table**.*

No.	*Class*	*Package (Pkg.)*
Eg0.	TableModel	**4.** javax.**swing.table.***
1.	TreeMap	
2.	Set	
3.	Map	
4.	JTable	
5.	BorderLayout	
6.	JScrollPane	
7.	SortedMap	
8.	AbstractTableModel	
9.	JPanel	

Variables	*Correct solutions*	*Discrete scale*
Y3.1 (Q5)	01. 1. javax.swing.*; 02. 3. java.awt.event.*; 03. 1. javax.swing.*; 04. 1. javax.swing.*; 05. 1. javax.swing.*; 06. 2. java.awt.*; 07. 1. javax.swing.*; 08. 1. javax.swing.*; 09. 1. javax.swing.*	(0 to 9)
Y3.2 (Q9)	01. 3. java.util.*; 02. 3. java.util.*; 03. 3. java.util.*; 04. 1. javax.swing.*; 05. 2. java.awt.*; 06. 1. javax.swing.*; 07. 3. java.util.*; 08. 4. javax.swing.table.*; 09. 1. javax.swing.*	(0 to 9)

Y4: Workings

Y4.1: In **CommandTest** code, the text appears when **goodbyeAction** is selected: _____

Y4.2: In your **CommandTest** program, indicate the **title name** of the window: _____

Y4.3: **Initially**, size set for **MapAdaptorTest frame**: **x:** _____ **pixels**; **y:** _____ **pixels**

Y4.4: The code to **suit MapAdaptorTest frame** into its **components' size**: _____

Variables	*Correct solutions*	*Discrete scale*
Y4.1 (Q2)	Goodbye, Departing…	(0 or 1)
Y4.2 (Q3)	A simple Command program	(0 or 1)
Y4.3 (Q6)	200 pixels; 150 pixels	(0 or 1)
Y4.4 (Q7)	test.pack( )	(0 or 1)

FX: Demographic Characteristics

X1: Gender*: Male / Female

Legend: * Please *circle one* of the items above.

X2 to X4: What grade did 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: _____

Variables (Original question number)
X1 (Header); X2 (Q1-i); X3 (Q1-ii); X4 (Q1-iii); X5 (Q2)

REFERENCES

[1] R.S. Pressman, Software Engineering: A Practitioner's Approach, 7th ed., McGraw Hill, New York, NY, USA, , p.347-354, p.835-837, 2010.

[2] V.R. Basili, and H.D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments", IEEE Transactions on Software Engineering, vol. 14, no. 6, pp.758-773, 1988.

[3] V.R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments", IEEE Transactions on Software Engineering, vol. 25, no. 4, pp.456-473, 1999.

[4] A. Goncalves, Beginning Java™ EE 6 Platform with GlassFish™ 3: From Novice to Professional, Springer-Verlag, New York, NY, USA, p.261-267, 2009.

[5] J.M. Carroll, "Minimalism beyond the Nurnberg Funnel", MIT Press, Cambridge, MA, USA, 1998.

[6] I. Chai, "Pedagogical framework documentation: how to document object-oriented frameworks: an empirical study", PhD dissertation, University of Illinois at Urbana-Champaign, IL, USA, http://www.cs.uiuc.edu/research/techreports.php?report=UIUCDCS-R-99-2077, 2000.

[7] R. Johnson, "Documenting frameworks using patterns", in: Proceedings of ACM Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'92), ACM

Press, Vancouver, British Columbia, Canada, pp.63-76, October, 1992.

[8] E. Berglund, "Designing electronic reference documentation for software component libraries", Elsevier, Journal of Systems and Software, vol. 68, no. 1, pp.65-75, 2003.

[9] A. Cockburn, "Supporting tailorable program visualisation through literate programming and fisheye views", Elsevier, Information and Software Technology, vol. 43, no. 13, pp.745-758, 2001.

[10] A. Dix, J. Finlay, G.D. Abowd, and R. Beale, Human computer interaction, 3rd ed., Pearson Prentice Hall, Essex, England, U.K., p.532-533, 2004.

[11] G. Meszaros, and J. Doble, "MetaPatterns: a pattern language for pattern writing", in: Proceedings of Int'l Conf. Pattern Languages of Programs (PLoP 96), The Hillside Group, Inc., Allerton Park, Illinois, USA, 4-6 September 1996, http://www.cs.wustl.edu/~schmidt/PLoP-96/meszaros.ps.gz.

[12] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyonds, Pearson Addison-Wesley, Boston, MA, USA, p. 24–29, 2003.

[13] S.B. Ho, "Framework documentation with patterns: an empirical study", PhD thesis, Multimedia University, Cyberjaya, Malaysia, 2008.

[14] A. Forward, "Software documentation: building and maintaining artifacts of communication", Master's thesis, School of Information Technology and Engineering, University of Ottawa, Canada, http://www.site.uottawa.ca/~tcl/gradtheses/ aforward/, 2002.

[15] Eclipse Website, The Eclipse project information, Online Available: http://www.eclipse.org/.

[16] T.C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: the state of the practice", IEEE Computer Society, Los Alamitos, CA, USA, IEEE Software vol. 20, vol. 6, pp.35-39, November-December, 2003.

[17] A. Forward, and T.C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey", in *Proceedings of the 2002 ACM Symposium on Document Engineering (DocEng'02),* ACM Press, McLean, Virginia, USA, pp.26-33, 8–9 November 2002.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Pearson Addison-Wesley, Reading, MA, USA, 1994. (Commonly called the "Gang of Four" or "GoF" book).

[19] R. Horstman, Object-Oriented Design and Patterns, 2nd ed., John Wiley and Sons, Hoboken, NJ, USA, 2006.

[20] B. Beizer, "Software is different", in Comparative Studies of Engineering Approaches for Software Engineering, eds. D. Patel, and Y. Wang, Baltzer Science Publishers, Norwell, MA, USA, Vol. 10, pp.293-310, 2000.

[21] P. Kallakuri, and S. Elbaum, Experimental studies in empirical software engineering, 2005, http://www.acm.org/crossroads/xrds7-4/empirical.html.

[22] C.D. Gray, and P.R. Kinnear, IBM SPSS Statistics 19 Made Simple, Psychology Press, Taylor and Francis Group, Hove and New York, USA, p.85-90, p.387-398, 2012.

[23] S.B. Green, N.J. Salkind, and T.M. Akey, Using SPSS for Windows: Analyzing and Understanding Data, 2nd ed., Prentice Hall, Upper Saddle River, NJ, USA, p.292-294, 2000.

[24] R. Johnsonbaugh, Discrete Mathematics, 6th ed., Pearson Prentice Hall, Upper Saddle River, NJ, USA, p.2-7, 2005.

[25] J. Neter, M.H. Kutner, C.J. Nachtsheim, and W. Wasserman, Applied Linear Statistical Models, McGraw Hill, Boston, MA,USA, 1996.

[26] L.C. Briand, C. Bunse, and J.W. Daly, "An experimental evaluation of quality guidelines on the maintainability of object-oriented design documents", in ACM Proceedings of 7th Workshop on Empirical Studies of Programmers, ACM Press, Alexandria, VA, USA, pp.1-19, 1–3 October, 1997.

[27] A. Field, Discovering Statistics Using SPSS, 3rd ed., SAGE Publications Ltd, London, U.K., p.560-567, 2011.

[28] A.G. Bluman, Elementary Statistics: A Step by Step Approach, McGraw-Hill International Edition, New York, USA, 2004.

[29] G. Antoniol, G. Canfora, and G. Casazza, Recovering traceability links between code and documentation, IEEE Transactions on Software Engineering, vol. 28, no. 10, pp.970-983, 2002.

[30] J. Han, and M. Kamber, Data Mining: Concepts and Techniques, Academic Press, Morgan Kaufmann Publishers, San Diego, CA, USA, p.284-296, p.319-326, 2001.

[31] W.H. Greene, Econometric Analysis, 6th ed., Pearson Prentice Hall, Upper Saddle River, New Jersey, p,8-18, 2008.

[32] D.N. Gujarati, Basic Econometrics, 3rd ed., McGraw-Hill, Singapore, p.332-335, 1995.

[33] M. Graff, "Individual differences in hypertext browsing strategies", Elsevier, Behaviour and Information Technology, vol. 24, no. 2, pp.93-99, 2005.