FRAMEWORK DOCUMENTATION WITH PATTERNS: CHARACTERISTICS AND OPPORTUNITIES FROM FIVE SERIES OF CONTROLLED EXPERIMENTS

Ho Sin Ban¹*, Ian Chai² and Tan Chuie Hong¹
¹ Faculty of Information Technology, Multimedia University, 63100 Cyberjaya, Malaysia.
² Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Malaysia.
Phone: +603 8312 5408, +603 8312 5379; Fax: +603 8312 5264; Email: {sbho, ianchai, chtan}@mmu.edu.my

ABSTRACT

As Information and Communication Technology (ICT) provides industry with sharing of knowledge and information in emerging product and service opportunities, object-oriented frameworks have been in the increasing demands to develop applications with components-reuse features. However, the size and complexity of frameworks makes understanding how to use them difficult. The mission of pedagogy still remains the same: to educate novices and overcome the steep learning curve of large and complicated interrelations among objects therein. With the rapid progress and development of ICT technologies such as web pages and the Internet, it has become feasible and affordable to integrate framework documentation into teaching and learning environments for more effective knowledge transfer. This makes mass learning possible.

As such, the research work in this paper empirically investigated different documentation philosophies for effective transfer of knowledge in teaching new framework users, especially where patterns-style documentation was used as the means of knowledge organization, interaction and representation. Five controlled experiments, which served as case studies, were set up to compare the various documentation philosophies, namely patterns-style, minimalist and extended javadoc (Jdoc) documentation. These three knowledge transfer strategies and tests were evaluated within two different frameworks. Subjects’ exercises were used to gather data, the results from which were used to formulate guidelines for effective framework documentation. The discoveries in this paper were two fold. First, the statistical analyses such as multivariate analyses of variance show that the effect and impact of the various documentation strategies are different. Second, we discovered different effectiveness of the patterns-style when applied to different frameworks. The formulated guidelines will be used to serve as an important roadmap for instructors who wish to integrate framework documentation into their teaching and learning environments. Different documentation philosophies are better for different goals. For a simple task, use minimalist documentation. For a much more complex problem, our empirical results suggest using patterns-style documentation.

Keywords: Documentation; Empirical; Minimalist; Frameworks; Patterns

*Corresponding author. sbho@mmu.edu.my
1. Introduction

Although object-oriented programming itself was touted to promote large-scale reuse of source code, reusable object-oriented software is usually based on, or is part of, a framework. Frameworks are reusable designs expressed in code. This allows the reuse of both design and code. The right framework can decrease the time to develop a piece of software by an order of magnitude. The framework is defined as a complex type of software. Framework formation must support adaptation and evolution. Adapting a framework involves knowing its design and how to use it to fit into a particular purpose. When evolution happens, the abstract design would become more concrete and understandable, since many more examples are demonstrated. The framework lifecycle can be viewed as in Figure 1. Within the framework, new demands in the domain, such as new functionality requirements, motivate re-development of an existing product. On the other hand, development of the prototype is based on feedback from framework adaptation. Advances in the application framework industry are moving towards higher complexity with much more functions and lengthier code. A large effort is required to understand any new software system.

With this rapid development in technology and the increasing shift from tutor-centered to learner-centered teaching and learning, different pedagogical documenting models are being developed to exemplify the aspiration to create environments for effective student learning. However, at present, there is a lack of research that shows which ways of documenting philosophies actually are better. Research should be carried out to identify the definite characteristics of effective teaching and learning, and the types of documenting models that would allow the most productive knowledge transfer.

This paper proposes an approach for evaluating pedagogical framework documentation using a series of controlled experiments. This type of evaluation is very useful for formulating sound guidelines, because documentation is an important part in educating users how to use an object-oriented framework to build technical applications. The types of documentation styles selected play important roles to aid software engineers in understanding a framework more precisely.
2. The problem to be solved

Accurate and comprehensible documentation is crucial to the success of large-scale frameworks. However, documenting frameworks is a costly activity and contemporary tools often focus on low-level method-oriented documentation, which fails to capture the strategic roles and collaborations among framework components. We expect that the advent of tools for reverse-engineering the structure of classes and objects in complex frameworks will help to improve the accuracy and utility of framework documentation. Likewise, we expect to see an increase in the current trend (Johnson et al. 1995, Buschmann et al. 1996, Schmidt 1997, Fayad 2000, Nino and Hosch 2005) of using design patterns to provide higher-level descriptions of frameworks.

Framework documentation in general is a widely recognized problem. It is hard to document mature frameworks, especially white-box frameworks because these frameworks tend to be large and contain complicated interrelations among objects therein. Developers new to a framework are often overwhelmed by the steep learning curve. However, once the framework is learnt, developers can do things much more efficiently. The official Extreme Programming (XP) concept is that customers should prioritize the stories for any release, without any interference from developers. Pawson and Matthews (2002) in their Chapter 4 of *Naked Objects* book showed seven examples of XP-style stories for the Executive Car Services (ECS) System. At the Naked Objects framework, Pawson and Matthews (2002) demonstrated the ability to generate the HTML user documentation (similar to the minimalist style) automatically from an executable user acceptance test, which was associated with the second story of the ECS system. These auto-generated English-language user instructions represent a significant proportion of the user training manual for the system under development. The training manual would have explicit instructions on how to cope with different scenarios on the system, such as the scenarios that users can expect to encounter, some of them routine and some of them exceptional.

The wide variety of documentation techniques (see Fayad et al. 1999) for documenting, specifying, and reasoning about frameworks can be categorized into a number of variations, e.g. examples, recipes and cookbooks (the most basic form is regarded as minimalism), contracts, patterns-style, framework overview, traditional manual, design notebooks, and the javadoc style, which was made popular by Sun Microsystems (2005). All these methods show that there is a growing body of work available for teaching how to write programs. Unfortunately, Fayad (2000) emphasized there is often little, or only anecdotal evidence on the impact of the documentation style in the actual reuse of a framework. The research in this empirical work is mainly focused on the styles shown in bold face in Figure 2: patterns-style, traditional instructions, minimalist documentation, and the extended javadoc style. The various documentation styles are summarized below as a reference.

- **Examples** – demonstrated *example applications* via source code using the framework. These example applications are usually created during the development process of the framework. A framework is usually a result of the evolution of an application. Other applications are developed to confirm the reusability of the framework before it is rolled out for more general use. Most of these revolve around a small number of simple applications.
- **Cookbook** – a collection of recipes. A *recipe* describes how to perform a typical example of reuse during application development. The most basic form of a cookbook corresponds to the
minimalist innovations from Dr John Carroll (1998). Firstly, people do not want information irrelevant to the task at hand. Secondly, he suggested giving the reader the minimal amount of information to get the task done. The idea of arranging information in short pages or index cards of information fits well to hypertext and web presentations. The cookbook may start with a guide to the contents of the recipes, either as a table of contents or by the first recipe acting as an overview for the cookbook.

- Contracts – a specification of obligations and collaborations. Contracts were originally a mechanism to compose behavioural descriptions from customers. The contracts were then refined into a more specialized behavioural composition, such as extending participant’s actions, or deriving a new invariant that implies the old. A contract is able to support a cookbook recipe, since a developer can consult this available specification of the collaborative behaviour of classes.

![Diagram showing various styles of documentation](image)

**FIGURE 2** The various styles of documentation (Fayad et al. 1999) used to describe how to adapt the functionality of a framework.

- Patterns-style – An object-model pattern is a grouping of objects with stereo typical responsibilities and scenario interactions (Coad 1997). Patterns could be organized into various pattern families in such a way that one will first encounter the *Fundamental* pattern. This may then lead on to Transaction patterns, Aggregate patterns, Plan patterns, and Interaction patterns. This contributes to the idea of having four to five subtasks, where each subtask could be
grouped into a pattern. Furthermore, Johnson (1992) introduced the concept of using patterns as a format for each recipe and an organization for documenting a framework. The elements include the purposes, background information, how to use it and some examples. This approach is the main style to be evaluated together with some other documentation styles, in the series of experiments for this research.

- **Framework overview** – often served as the first recipe in a cookbook. Setting the context of a framework is the first step in encouraging novices to reuse the framework. The overview includes defining the jargon of the domain, the scope and the flexibility of the framework.

- **Traditional instructions manual** – based on the idea of the *traditional step-by-step instructions*, i.e. if we give clear steps of what needs to be done, the novices can follow them and accomplish the task. Interspersing explanations along the way will let the novices learn things that are not obvious from the steps. This is the traditional form that most documentation takes.

- **Design notebooks** – based on issue-driven design to capture the design rationale of software systems, as well as hardware systems, and combined hardware / software systems. The information in this style includes the domain theory, analyses of situations, and a discussion of engineering trade-offs in the approach being taught.

- **Extended javadoc** – makes use of the HTML documentation generated by the javadoc tool (Sun Microsystems 2005). It first presents class information, such as inheritance and subclasses. This is followed by a textual description of the class, constructors and methods, which contain their pseudocode with hypertext links to the particular steps.

The method-of-choice resulting from this empirical study on pedagogical framework documentation is patterns. The patterns technique has come a long way since the landmark book by Alexander et al. (1977). The Alexander’s patterns idea, as it is usually recalled in the literature, is well known for its timeless reusability and applicability. The method organizes architectural patterns into a systematic arrangement, like the everyday structures used in towns, buildings and conclusions. This method is suitable for laying down sound rules for having a proven successful solution to a recurring problem. Initially, the patterns concept is used to embody decisions the designer made in physical world, building houses from reusable parts are much faster than building them from scratch. For a true analogy, consider one is given a problem of long corridors with straight and narrow design with doors, which are close to each other. To have a proven successful solution, the designer can lay down rules of short passages with carpets or wood on the floor with plenty of light and windows along the entire wall.

### 3. The proposal

Consider the following statements, which are proposed to be sound, after observations from the few series of controlled experiments that we have carried out thus far.

- All object-oriented (OO) frameworks reuse components.
- Some OO frameworks are suitable for patterns-style documentation, for example the Habanero framework as observed in Chai (2000).

∴ Therefore, some frameworks that reuse components are suitable to be documented with the patterns style.

Usually, we will use the general first order logic (FOL) to verify the above statements, as Taibi and Ngo (2003) have attempted to describe the structural aspect of patterns using a subset of FOL.
These first order formulas use typical connectors, i.e. \( \land \) (conjunction), \( \lor \) (disjunction), \( \neg \) (negation) and \( \to \) (implication), plus the quantifiers of \( \forall \) (universal), and \( \exists \) (existential). The conclusion is follow after the symbol of \( \Rightarrow \), which indicates the inference obtained from the premises

\[
\forall x \ [ \text{OO}(x) \rightarrow \text{RC}(x) ] \land \exists x \ [ \text{OO}(x) \land \text{PS}(x) ] \Rightarrow \exists x \ [ \text{RC}(x) \land \text{PS}(x) ]
\]

where the following denotation of predicates takes place.

- OO: Object-oriented framework
- RC: Reuse components
- PS: Patterns-style suitability

Inference Proof:
1. \( \forall x \ [ \text{OO}(x) \rightarrow \text{RC}(x) ] \) P (Premise)
2. \( \exists x \ [ \text{OO}(x) \land \text{PS}(x) ] \) P
3. \( \text{OO}(c) \land \text{PS}(c) \) 2, Existential Instantiation (EI) to instantiate an element, \( c \)
4. \( \text{OO}(c) \rightarrow \text{RC}(c) \) 1, Universal Instantiation (UI)
5. \( \text{OO}(c) \) 3, Simplification
6. \( \text{RC}(c) \) 4, 5, Modus Ponens
7. \( \text{PS}(c) \) 3, Simplification
8. \( \text{RC}(c) \land \text{PS}(c) \) 6, 7 Conjunction
9. \( \exists x \ [ \text{RC}(x) \land \text{PS}(x) ] \) 8 Existential Generalization (EG)

q.e.d. (what was to be proven: \textit{quat errm demonstrantion})

With the sound intuition that some frameworks that reuse components are suitable for patterns-style documentation, we explored further what frameworks (other than Habanero framework), would benefit from the use of patterns-style documentation. What features of a framework determine the suitability of documenting with patterns?

As such, we carried out five series of controlled experiments to understand better this phenomenon. Table 1 summarized the experiments that we have conducted in two frameworks, i.e. the Visual Basic (VB) and Swing (Swg.) frameworks. We chose VB and Swing due to their relative availability and convenience. The VB and Swing experiments were carried out during tutorials of a Software Engineering course in February 2004 and an Object Oriented Programming course in August 2004 respectively. These programming tools were taught in these courses. Furthermore, Raadt (2004) showed in his census that VB is the most commonly used language to introduce the RAD world, rather than other popular RAD tools such as Delphi and JBase.

\textbf{TABLE 1.} The five series of controlled experiments.

<table>
<thead>
<tr>
<th>Stages</th>
<th>Description of work tasks</th>
<th>Total valid cases evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB Stage 1</td>
<td>Enhanced Payroll Program</td>
<td>281 valid respondents</td>
</tr>
<tr>
<td>VB Stage 2</td>
<td>Menu and Text Program</td>
<td>86 valid respondents</td>
</tr>
<tr>
<td>Swg. Stage 0</td>
<td>Simple Drawing Program</td>
<td>140 valid respondents</td>
</tr>
<tr>
<td>Swg. Stage 1</td>
<td>Invoice Program (using five design patterns)</td>
<td>79 valid respondents</td>
</tr>
<tr>
<td>Swg. Stage 2</td>
<td>Command and Adaptor Program</td>
<td>48 valid respondents</td>
</tr>
<tr>
<td>Five studies</td>
<td>Grand total cases evaluated</td>
<td>634 cases</td>
</tr>
</tbody>
</table>
4. Conclusion

The experiments test the effect of different styles of documentation on learners trying to accomplish given tasks. The pedagogical framework documentation contains discrete components, i.e. task overview, background information, how does it work and what next sections. Related structures imply this model could be generalized to environments such as Graphical User Interface (GUI) and the coding-intensive framework itself. The authors draw upon previous contributions by past researchers and improved on a few key areas (notably comparison of patterns-style and its presentation order in certain sequences) to enable the patterns-style approach to effectively teach novices how to use a framework. Furthermore, the questions of effectiveness of the patterns-style, when applied to frameworks, containing inter-object behaviours and reuse-related tasks, such as description of usage, incremental learning and operational realization have been answered, at least in the context of two different frameworks. The main contributions of this empirical work include the following.

- For all the five evaluations on Visual Basic (VB) and Swing framework, an important discovery is that documentation cannot be expected to work without testing. This situation is similar to white-box testing (Galin 2004) because of the approach in examining every paragraph and section in order to identify any description that is not clear. The iterative refinements that the documentation went through means that all the versions were highly effective. Based on the suggestions by preliminary testers, the different versions were all improved before the real trials with the actual sample subjects.
- Instead of a 'common errors' page, encourage the novices to test at the end of every checkpoint. This is to prevent the big-bang testing approach. The incremental testing approach helped neophytes to be able to complete the whole work task in the end. In this way, the novices solve the problem right from the early stage. The code would not run unless one is cautious of the common errors from the beginning.
- Tell people things in the same order, as they will type them in the resulting source code example, even if the order is ultimately arbitrary. From observations, this way prevented needlessly complicating matters. From all the experiments, grouping the order of “before” and “after” complete source code into a systematic cyan-coloured box make it easy for people to find the code of the running example.
- In the VB experiment (Ho et al. 2004), which involved an easier task without design patterns, the subjects who used minimalist documentation completed their first compilation significantly faster that the ones using patterns documentation. This is also compatible with the Habanero experiment (Chai 2000). It was of medium complexity and found that minimalist group finished faster, while the group of novices using the patterns-style documentation understood better. Minimalism helped in a simpler task, such as creating a prototype basic payroll program. However, patterns-style documentation helps novices to get more complicated tasks, such as those involving the five design patterns, done faster and more accurately. As such, different documentation methods are better for different goals.

It is important to note the degree to which the results of this research can be generalized to the population under study and other research settings. The subjects who participated in the experiment were undergraduates and may not be the full representative of software learners. However, all the subjects’ demographic data were collected and showed that on average, they
underwent undergraduate course in the university for nearly three years. Thus, they were reasonably representative of novices in the software industry, since the undergraduates were to enroll into industrial training for at least four months in the subsequent trimester.

References