Abstract—Frameworks are a useful way to enable object-oriented reuse, but their size and complexity make understanding how to use them difficult. Different people have proposed different ways to document frameworks, but it was unclear which ones actually were better. This study empirically investigated three documentation philosophies for new framework users, and discovered some guidelines for effective framework documentation, particularly in a Rapid Application Development environment. It is also found that different documentation styles are better for different goals.

Index Terms—Documentation, Empirical, Minimalist, Object Oriented Frameworks, Patterns, Statistics

1 INTRODUCTION AND MOTIVATION

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.

However, frameworks have a number of reuse problems, e.g. those identified by Kirk et al (2002). Frameworks have large amounts of behavioral and structural information. Both designing and learning to use a framework is difficult because much of this information is unfamiliar to developers new to a framework. The real benefits to a framework only come after one knows how to use it.

1.1 What are frameworks?

What exactly are frameworks? One can think of frameworks as reusable parts for a particular domain (Christensen, 2002). For example, in the physical world, building houses from reusable parts are much faster than building them from scratch. Similarly, one can build application programs from the objects in a framework. The parts embody decisions the designer made.

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 (Alexander, 1977). If the designer embodied a good interior design, one can rely on that design in houses. In the same way, if the designer embodied a good design in the framework, one can rely on that design in the program.

Just as in the physical world, having the pieces is not enough. A framework is a reusable design, not just a set of parts. 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. If a person is familiar with a framework, it does not mean it is trivial for him to learn how to use another.

1.2 Why framework documentation?

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 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-established rules for documenting framework.

We intend to make it easier to learn new frameworks by discovering better ways to document them. Thus, in this paper, we present Visual Basic framework as our study of documentation for Rapid Application Development (RAD) Environment. This paper complements the earlier Habanero framework study (Chai, 2000). The sample size of data in this study involves more than 280 subjects.

### 1.3 RAD environment for this experiment

Object-oriented philosophy is particularly suited for prototyping. The objects supporting the prototyping are similar to those that needed to be presented in the final product. In this experiment, the optimal timing of one hour is given to the subjects to grow the prototype into an enhanced payroll program that is feasible.

Horstman (2004) stated that most people recommended implementing a prototype in a RAD tool such as Visual Basic before converting the final product in the true object oriented programming language such as Java. The RAD environment in this experiment makes use of the evolutionary approach. Through the four different work tasks in the documentation, the experiment subjects fix mistakes and expand a prototype into the final implementation solution, which is checked via a survey at the end.

We chose Visual Basic (VB) due to the relative availability and convenience. The subjects of this experiment were undergraduates pursuing software engineering course in January 2004, and Visual Basic was the RAD tool taught in this course. In fact, Raadt (2004) has shown in his census that VB framework is the most commonly used language to introduce the RAD world, rather than other popular RAD tools, like Delphi and Power Builder.

### 1.4 Experiment Rationale

Table 1 summarizes some key points of the three philosophies being evaluated in this paper. See (Chai, 2000) and (Johnson, 1992) for more details on these philosophies. Each of these philosophies sounds reasonable from their description.

So, one might not know which is best. Therefore, we wanted to discover which documentation philosophy would work better than the others for documenting frameworks for new users. If we know the right way to structure the documentation, we could have more effective documentation.

Large RAD software systems are almost always expressed in free text documentation. The informal way of expressing user manuals created different ways to document frameworks in a RAD environment (Penta, 2002) (Antoniol, 2002). The experiment results provide guidelines for explanatory text to make documentation an effective means of communication.

### 1.5 Related Work

The assessment on Pattern Comment Lines (PCL) presented by Prechelt et al. (2002) suggests that design patterns always be documented explicitly in program source code. PCL in a program 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 al., 2001). Prechelt et al. 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 design pattern as unexpected new requirements often appear.

The most related experiment so far appears to be (Chai, 2000), which empirically investigated three documentation philosophies for new users in Habanero framework. This report presented experimental results on the Visual Basic framework, to determine which documentation styles are better for various goals.

### 1.6 Paper Structure

In Section 2, we present a brief walkthrough of the three philosophies of documentation used in this study, while Section 3 describes the experimental design, the subjects’ background and how the experiment was done. Section 4 presents the results for this experiment, and Section 5 summarizes and raises questions for future research.

## 2 Walkthrough of the Three Philosophies of Documentation

The general problem of how to document frameworks is large. Thus, we have decided 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.

In this experiment, we have selected three philosophies of documentation to study: traditional step-by-step instructions, minimalist documentation, and patterns-style documentation. These are directly competing philosophies in that some of their guidelines contradict each other.

2.1 Traditional Step-by-Step Instructions

The philosophy behind Step-by-step Instructions is that 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 student learn things that are not obvious from the steps. The documentation should be in one clear, flowing narrative.

This is the traditional form that most documentation takes.

2.2 Minimalist Documentation

Dr. John Carroll’s innovation, minimalist documentation, is based on two ideas: Firstly, people do not want information irrelevant to the task at hand and secondly, people are not good at following step-by-step instructions. They attempt 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 (Carroll, 1998).

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 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.

One important thing that Carroll does not list in these guidelines is the overriding sense throughout his book that minimalist documentation is about giving the minimal amount of information necessary for the reader to be able to get the task done. Perhaps he does not list it because it is so implicit in everything minimalist documentation is about.

2.3 Patterns

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.

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. In contrast, a traditional, step-by-step tutorial usually gives readers the steps to take without explaining why they should take them.

Patterns often refer to other patterns when a problem or its solution is too big to discuss in one sitting. Hence, they lend themselves well to hypertext presentations such as those found on the Web.

Gerard Meszaros and Jim Doble (1996) say that patterns should have these elements:

- Pattern Name: so that people can refer to the pattern
- Problem: the description of the problem it solves.
- Solution: there may be different solutions to the same problem depending on the context.
- 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.

The patterns style is a documentation technique that has been used to document good object-oriented design practices (Gamma, 1994), (Brugge, 2004), software packages, such as Enterprise Java Beans (Marinescu, 2002) and frameworks (Braude, 2004) as well as how to do many other things. Some of the diverse things people have documented using patterns include persuasive public speaking (Jaffe, 2004), how to run an organization (Coplien, 2001), and how to write patterns themselves (Meszaros, 1996).

3 DESCRIPTION OF THE EXPERIMENT ON THE VISUAL BASIC FRAMEWORK

We now give a short description of the experiment design and conduct. Visual Basic (VB) Framework (Griever et al., 2000) was selected for this study as a RAD environment. Further details, including the original experiment documents, such as the programs,
questionnaires, work tasks, and documentation materials used in this experiment are available from http://pesona.mmu.edu.my/~sbho/Writing/.

3.1 Experiment Objectives

It is tempting to use pattern and minimalist documentation solutions even if the actual programming problem is simple, while it may be sufficient just to use traditional step by step documentation. In this experiment, we wanted to test whether using pattern and minimalist documentation in such cases is helpful.

All this cannot be easily surmised by looking at the source code. Thus, documentation can help new programmers figure out what they need to do.

3.2 Hypotheses

The hypotheses will be judged based on their mean or expected value. The following are the hypotheses or expectations that we investigated in completing a prototype of Rapid Application Development (RAD).

E1: Pattern documentation does not improve the performance when compared to subjects doing the same exercise using minimalist documentation.

E2: Pattern documentation does not improve the performance when compared to subjects doing the same exercise using step by step documentation.

E3: Minimalist documentation does not improve the performance when compared to subjects doing the same exercise using step by step documentation.

The interpretations of this experiment are derived from the rejection or non-rejection of these hypotheses.

3.3 Subjects and Groups

We divided the subjects into three groups according to the subjects’ tutorial sections. Each subject worked on all four sub tasks for the prototyping program. The subjects who took part in this experiment are summarized in the upper part of Table II.

The 281 subjects are all undergraduates taking a software engineering course at Multimedia University, Cyberjaya, Malaysia. These subjects consist of mixture students of two faculties, i.e. the Faculty of Information Technology (FIT) and the Faculty of Management (FOM). On average, they had been studying at Multimedia University for 2.8 years.

We performed Pearson Chi-Square tests to identify whether there are any differences between the three subject groups. Demographic data about skill level and programming experience was gathered by a questionnaire instrument during the experiment.

Based on the questionnaire, interestingly, the Pearson Chi-Square tests in Table II show that there is no significant difference detected, with all the p-values > 0.05. The random assignments of the three tutorial groups are balanced in terms of their skills level in Java, C++ and the C programming language.

The subjects’ median grade obtained for prior programming courses on Java (Object Oriented Programming Language: Java Grade C++ Grade C Language Grade p-values: 0.338 0.612 0.187)
P-values are the results of the Pearson Chi-square tests on Programming Language Grades for the three documentation groups. Those subjects who did not take either one of these three programming language courses are excluded.

Fig. 1. Box and Whisker plots show frequencies of programming grades for the subjects in each group. Left: Java grade. Middle: C++ grade. Right: C Language grade.
Programming), C++ (Computer Programming II) and C language (Computer Programming I) was basically grade ‘B’, as shown in Figure 1.

In this study, the grades of the students are generally categorized into grade ‘A’ (4 points) for highest achievement, ‘B’ (3 points) for average, ‘C’ (2 points) for poor, F (1 point) for fail and ‘None’ (0 point), if the student has not taken the course at all.

The subjects in this experiment was a good representative sample of the total population for Rapid Application Development (RAD) beginners, since most likely beginners are usually located in the average grade group.

3.4 Design

Our experimental design uses four independent variables (factors) and five dependent variables. The independent variables are the documentation type, grade of the three programming courses, i.e. Java, C++ and C language; the dependent variables are the completion time, number of difficulties faced and semi completion time, accuracy, workings and comprehension (understanding of the exercise).

- “Documentation Type”: We use three different documentation philosophies, each with similar purpose, to create a prototype Enhanced Payroll program using Rapid Application Development tool. This is the central variable of our experiment. However, the subjects did not know that this variable was in the experiment at all. They only knew “the experiment helps identify areas for documentation improvement.”

- “Java Grade”: This course provides a good understanding of the key features of object-oriented technology as well as an industry standard methodology, Unified Modeling Language (UML) for Object-Oriented Analysis and Design. Subjects who have taken this course will be expected to analyze and design an Object-Oriented system in UML and to implement it using the Java programming language.

- “C++ Grade”: Most of the subjects have taken this course. Its aim is to initiate skills to write algorithms and programs and to introduce various programming paradigms, programming methodology, and the object-oriented language C++.

- “C Language Grade”: This course gives an introduction to basic programming concepts through the use of a high-level programming language such as C. It covers the basic notions and techniques for algorithm development and the implementation of algorithms in a high-level programming language. This course is the prerequisite for the C++ course.

Writing the code

6. Write the initialisation code to set up the values.

   To set up the array for the unit prices and unit measures
   Double click on the form to get the Code window.
   Select the object ‘(general)’. You will find the Procedure to be changed to ‘(declaration)’.
   Type the following array declaration:
   Dim employee_info(5,2)
   This declares a 6 by 2 array (indices by default start at 0).

7. Create the start-up code for the form.

   Select the object ‘Form’ and the procedure ‘Load’.
   Type the following codes to put employees in the List and values in the array.
   Object: ‘Form’ – Procedure: ‘Load’
   Sub Form_Load ()
   employee.AddItem “Ahmad”
   employee.AddItem “John”
   employee.AddItem “Chong”
   employee.AddItem “Raja”
   employee.AddItem “Samy”
   employee.AddItem “Anne”
   employee_info(0, 1) = 10000
   employee_info(0, 2) = “Director”
   employee_info(1, 1) = 8000
   employee_info(1, 2) = “Manager”
   employee_info(2, 1) = 5000
   employee_info(2, 2) = “Assistant Manager”
   employee_info(3, 1) = 3000
   employee_info(3, 2) = “Supervisor”
   employee_info(4, 1) = 2000
   employee_info(4, 2) = “Executive”
   employee_info(5, 1) = 900
   employee_info(5, 2) = “Clerk”
   End Sub

Fig. 2. This documentation fragment shows an example from the Traditional Step by Step Documentation.
• **Dependent variable “completion time”:** The time taken to finish the entire exercise.

• **Dependent variable “semi completion time”:** The time taken for the subjects to do their first compilation. We originally took this data because it was convenient. Later on, we discovered some interesting results.

• **Dependent variable “number of difficulties faced”:** Instead of giving all the detailed steps, some parts of the documentation let the learner interact with the system. Thus, the subjects were to record and accumulate numbers of problems they encountered.

• **Dependent variable “comprehension”:** The subjects are asked to list the number of objects and procedures that they have updated to complete the exercise. There are eight objects with respective procedures to test how well their understanding on the objects.

• **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 figure out the code in assigning variables to array elements.

---

**Fig. 3.** Example from the Minimalist Documentation Web Pages. More detail, including the original experiment documents such as Quick Start, Prototyping, Menu Objects and Common Errors are available from http://pesona.mmu.edu.my/~sbho/Writing/VbMeta/

**Fig. 4.** Example from the Pattern Documentation style. More detail, including the original experiment documents such as Quick Start, Prototyping, Menu Objects and Common Errors are available from http://pesona.mmu.edu.my/~sbho/Writing/VbPattern/
For this experiment, all groups achieved near-perfect accuracy, comprehension and workings. So, we will often ignore these three dependent variables.

3.5 Developing the Materials

Our experimental design used three sets of documentation for the experiments. For examples of materials from the three documentation styles, see Figures 2, 3, and 4.

They had the same purpose: to teach the reader how to build a prototype program using a Rapid Application Development (RAD) tool. We have selected Visual Basic for the purpose of this experiment.

The full documentation consists of the Basic Topics and the Intermediate Topics, but this paper only focuses on the Basic Topics because we had to limit the scope of the task in the experiment to something that the subjects could complete in one session. The Basic Topics alone was about half the size of the entire documentation, and described what most prototyping with an RAD tool would need.

4 RESULTS AND DISCUSSION

The expectations form the basis of our discussion and interpretation of the quantitative results. Note that this style of discussion is dense, but no simpler form would adequately characterize the effects we observed.

4.1 Statistical Analysis on the Results

For the actual statistical analysis, we did not want to rely on the assumption of normal distributions as made in the standard analysis of variance techniques. Therefore, we tested for the normality of the dependent variables.

From the normality test, we discovered that all dependent variables except “number of difficulties” are normally distributed for each of the documentation types. As such, univariate analysis can be applied and Scheffe test (Neter et al., 1996) is used to examine the significance differences because the sample sizes for the three documentation groups in this experiment are unequal.

On the other hand, the number of difficulties does not conform to normality. Thus, medians will be used as the expected values, rather than the means. We pick the usually nonparametric technique, i.e. Kruskal-Wallis test (Neter et al., 1996) to determine whether there is any significance for this dependent variable.

4.2 Results for VB Framework

We analyzed the data to see if one of the sets of documentation let the subjects compile (SEMICOMPLETION) and finish the fastest (COMPLETION) with the number of difficulties recorded by the subject at these intervals (DIFFICULTIES), as well as understand the most (COMPREHENSION). We also checked for the relevance of the programming skill via the grades they obtained for programming courses, as well as the correlation to the self-tested scores of how well the documentation taught them how to rapidly build an application (ACCURACY) and test scores on how well their knowledge in the inner workings of Visual Basic (WORKINGS).

### TABLE III

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Mean</th>
<th>StdDev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semicompletion (hh:mm)</td>
<td>0:18</td>
<td>0:16</td>
</tr>
<tr>
<td>2</td>
<td>Completion (hh:mm)</td>
<td>0:41</td>
<td>0:38</td>
</tr>
<tr>
<td>3</td>
<td>Comprehension (# of objects)</td>
<td>4.13</td>
<td>5.04</td>
</tr>
<tr>
<td>4</td>
<td>Accuracy (scale: 0, 1, 2)</td>
<td>1.92</td>
<td>1.92</td>
</tr>
<tr>
<td>5</td>
<td>Workings (scale: 0, 1, 2)</td>
<td>0.98</td>
<td>1.42</td>
</tr>
</tbody>
</table>

The means and standard deviations of all categories, where columns are (from left to right): Traditional Step by Step (Trad.), Minimalist (Min.), and Patterns (Pat.). The bolded values show the most preferred mean.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pat. vs. Min</td>
<td>0.629</td>
<td>0.380</td>
<td>*0.000</td>
<td>0.902</td>
<td>*0.000</td>
</tr>
<tr>
<td>2</td>
<td>Pat. vs. Trad.</td>
<td>0.129</td>
<td>0.493</td>
<td>*0.000</td>
<td>0.690</td>
<td>*0.000</td>
</tr>
<tr>
<td>3</td>
<td>Min. vs. Trad.</td>
<td>*0.030</td>
<td>0.071</td>
<td>0.120</td>
<td>0.502</td>
<td>*0.046</td>
</tr>
</tbody>
</table>

The statistically significant differences, where columns are (from left to right): Semicompletion (Semi.), Completion (Compl.), Comprehension (Compr.), Accuracy (Accur.) and Workings (Work.). * indicates there is evidence of differences with p-value < 0.05 significance level. These values were obtained via multiple comparisons with adjustment using Scheffe test.

For this experiment, all groups achieved near-perfect accuracy, comprehension and workings. So, we will often ignore these three dependent variables.

### RESULTS AND DISCUSSION

The expectations form the basis of our discussion and interpretation of the quantitative results. Note that this style of discussion is dense, but no simpler form would adequately characterize the effects we observed.

4.1 Statistical Analysis on the Results

For the actual statistical analysis, we did not want to rely on the assumption of normal distributions as made in the standard analysis of variance techniques. Therefore, we tested for the normality of the dependent variables.

From the normality test, we discovered that all dependent variables except “number of difficulties” are normally distributed for each of the documentation types. As such, univariate analysis can be applied and Scheffe test (Neter et al., 1996) is used to examine the significance differences because the sample sizes for the three documentation groups in this experiment are unequal.

On the other hand, the number of difficulties does not conform to normality. Thus, medians will be used as the expected values, rather than the means. We pick the usually nonparametric technique, i.e. Kruskal-Wallis test (Neter et al., 1996) to determine whether there is any significance for this dependent variable.

### RESULTS FOR VB FRAMEWORK

We analyzed the data to see if one of the sets of documentation let the subjects compile (SEMICOMPLETION) and finish the fastest (COMPLETION) with the number of difficulties recorded by the subject at these intervals (DIFFICULTIES), as well as understand the most (COMPREHENSION). We also checked for the relevance of the programming skill via the grades they obtained for programming courses, as well as the correlation to the self-tested scores of how well the documentation taught them how to rapidly build an application (ACCURACY) and test scores on how well their knowledge in the inner workings of Visual Basic (WORKINGS).
In terms of COMPLETION, when looking for the standard significance level of 0.05 (i.e. 95% probability) in Table III, we see that none of the treatments make a significant difference. Therefore, we conclude that there is no significant difference between the three styles as to how long it takes for the subjects to complete the experiment.

For SEMICOMPLETION, the subjects using the minimalist documentation completed their first compilation faster than the ones using the other two documentation styles. When we used the standard significance level of 0.05, we find that subjects in the minimalist group significantly faster to compile than those in the traditional group.

Regarding COMPREHENSION, subjects who used patterns exhibited significantly higher comprehension scores than those using minimalist or traditional documentation.

As for ACCURACY, there was no significant difference between how well the subjects thought the three kinds of documentation taught them how to rapidly develop application. This might be because the students were able to accomplish the task in the end. Their programs produce the correct answers based on the parameters of the exercise.

Pertaining WORKINGS, subjects who used patterns documentation rated patterns more highly in terms of their effectiveness on teaching them how Visual Basic works internally than other two documentation styles. In this particular sample, the subjects who used minimalist documentation also on average rated them higher than those who used traditional documentation. Interestingly, there are significant differences in the workings factor, which means that the three hypotheses in section 3.2 are rejected. These rejections show that the three documentation styles are not the same in teaching the subjects about the inner workings of Visual Basic.

### 4.3 Minimalist Advantage in Debugging

Minimalist documentation was faster than traditional step-by-step and patterns documentation in the semi-completion time and completion time. Moreover, there was significant difference in the semi-completion time between minimalist and traditional documentation style. This suggests that the advantage that minimalist had over step-by-step and patterns documentation came into play mainly during the debugging phase, because semi-completion time is the time at which the subject did their first compile, before debugging.

This is further supported by the proxy of NUMBER OF DIFFICULTIES. Since this dependent variable was not normally distributed, we used the Kruskal Wallis test. With the asymptotic significant value in Table IV less than 0.05, the number of difficulties faced by the subjects has no significant difference between minimalist and the two other documentation styles.

Most debugging tools, particularly in the RAD environment are oriented toward inspecting and following the activity of a single thread of control (Donat, 2003). With minimalist documentation, users need not read background information that may not be relevant in reducing their number of difficulties. Patterns and traditional documentation provide this background information.

Among the strong proxies that confirmed minimalist advantages include fastest semi-completion time, fastest completion time, one of the highest accuracies and no significant difference of difficulties. Hence, we conclude that minimalist documentation is relatively superior to others in debugging.

### 4.4 Qualitative Results and Discussion

Our findings in this VB experiment both support and contrast with some of our earlier findings in the Habanero framework experiments (Chai, 2000). For both these experiments, we found that documentation cannot be expected to work without testing. This situation is similar to programming: just as one cannot write software and expect it to work without testing, one cannot write documentation and expect it to be effective without testing. The iterative refinement this VB documentation went through means that all the three versions were highly effective. Based on the suggestions by a preliminary tester, the three versions were all improved before the

<table>
<thead>
<tr>
<th>Documentation type</th>
<th>Sample size, n</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Removal of invalid cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Step by Step</td>
<td>96</td>
<td>1.50</td>
<td>4.35</td>
<td>30 not valid cases</td>
</tr>
<tr>
<td>Minimalist</td>
<td>73</td>
<td>1.00</td>
<td>2.20</td>
<td>3 not valid cases</td>
</tr>
<tr>
<td>Patterns</td>
<td>77</td>
<td>2.00</td>
<td>3.02</td>
<td>2 not valid cases</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Chi-square</th>
<th>DF (Degree of Freedom)</th>
<th>Asymp. Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.179</td>
<td>2</td>
<td>0.204</td>
</tr>
</tbody>
</table>

With asymptotic significant < 0.05, no difference for all the three types of documentation in difficulties factor.
real trials with the actual sample subjects.

VB is an example of a framework where the users do not need to understand how things work internally. The users simply need to be able to use things quickly. The mean values in Table III indicate that all three documentation groups achieved a nearly perfect accuracy of 2, in spite of the fact that different groups ended up with different amounts of knowledge of VB’s inner workings.

Instead of a "common errors" page, point out common pitfalls in the main text. This builds upon the earlier findings in (Chai, 2000). Instead of a separate page of “common errors” as was done in the earlier study, errors are highlighted right before the instructions where users commonly make the error. In this way, most users can follow the steps without having to figure out how to solve the problem on their own. The code would not run unless one takes caution on the common errors.

Segregate the navigational instructions on menus and windows from the code. The code fragments are put in cyan-colored boxes to highlight them to the users. However, unlike in the earlier Habanero study, these study users did not skip over the explanatory text and just read this code. This is because they know they need to navigate the menus and windows, and hence they are forced to read this part of the notes. Unfortunately, some people ignored the background information and skipped directly to this part.

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. This avoids needlessly complicating matters. We found that 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 running example.

Carroll’s observations (Carroll, 1998) still hold in a simple sequential problem. This VB experiment presented the step by step instruction to a sequential prototype problem without any explanatory notes – basically just step 1, step 2 etc. We observed that minimalist documentation still give the fastest time, although the steps in traditional documentation are not interspersed with explanations.

5 CONCLUSIONS

We began this work with the idea of discovering which of three philosophies worked best for new user framework documentation in a RAD environment.

Step-by-step documentation is the one traditionally used for software documentation. The philosophy behind it is that if one gives people clear, sequential steps, they can follow them and learn how to do the task. Any extra information can be interspersed among the steps at relevant places.

Minimalist documentation cuts all explanations into small interlinked pieces and generally avoids giving background information. Hence documentation should be minimal, where learning using this philosophy involves a self-directed manner. Also, it should be random-access so that the reader can proceed in a self-directed manner, so minimalist documentation promotes using small cards or pages that can be read in the order the reader desires.

Patterns centers upon the idea of giving the solution to a problem in its context. It also has the random-access idea of minimalist documentation, but disagrees with it in that contextual information is not minimal - it usually contains information other than what needs to be done.

5.1 Which documentation works better?

Minimalist documentation helps people get the task done faster, while the patterns style helps people to better understand the internal workings.

Hence, if one has a situation where speed is more important than the understanding of the internal structure, minimalist documentation would be better. Black-box components may be such an application. If, on the other hand, understanding how the framework works is more important, then patterns would be better. Most frameworks are not completely black-box, and tend to have areas where some understanding is needed, so this would appear to be the most widely-applicable solution in the world of frameworks.

As for step-by-step documentation, we found that people do not like to follow step-by-step instructions, nor to be forced to read a lot of information. Even though the step-by-step documentation contained as much explanation as the patterns documentation, the subjects using it scored lower in the comprehension test. This suggests that they were skimming too much in their attempt to avoid following and reading everything.

Skippable sections are helpful - users can read them if they are interested, but can safely skip them and still get the job done if they are not interested in deeper understanding. The patterns style is organized that way. Try not to surprise people. Surprise is good for a novel, but can mislead people in documentation.

Aside from the documentation’s organization, a major part of producing good documentation is to have good text. Just as good software requires testing, we found that good documentation requires testing. A good piece of documentation goes through much iteration of testing and revision.

5.2 Future Work

These experiments answered some questions, but also raised many more. The following questions
appear most important: First, if we increase the resolution of the scale of WORKINGS and ACCURACY, say, from two to seven, will our observation still hold? Perhaps, with many more questions in the exercise, there will be a more accurate reckoning of how much knowledge the users gain from the different documentation types.

Second, how did the patterns subjects know how to answer the comprehension questions correctly if they skipped the patterns describing the internal workings? Third, is it better for the explanations to be arranged around the examples? Fourth, should there be one running example, and attempt to fit that running example into all situations, or should there be examples crafted to illustrate different points?

Another area that future experiments can explore is which kind of diagrams, e.g. UML (Priestley, 2004) and entity-relationship diagram (Schach, 2004), work better in helping application programmers understand the framework in order to use it? Sixth, is it possible or beneficial to take into account different learning styles, or is it more reasonable to provide a general set of documentation that tries to cater to all learning styles? Seventh, will deeper understanding of the RAD objects come with several days of using the documentation?

We have applied the proposed methodology of pedagogical framework documentation to rapid application development. This empirical study compared different documentation styles to identify ways to improve framework documentation. However, the proposed paradigm and mechanisms only address a small portion of the need for software engineering methodologies in documentation.

ACKNOWLEDGEMENTS

The authors would like to thank all their experimental subjects and the course tutors for making the experiments possible.

REFERENCES


**Ho Sin Ban** received a B.Sc. and an M.Sc. in Computer Science from the University of Science Malaysia, Penang, Malaysia in 1998 and 1999. His primary research interests include empirical software engineering, with his main focus on framework documentation, design patterns and the empirical research approach. He is currently pursuing a Ph.D. in the area of framework documentation with patterns. He is a member of the IEEE.

**Ian Chai** received a B.Sc. and an M.Sc. in Computer Science from the University of Kansas, Lawrence, Kansas in 1988 and 1990, respectively, and a PhD in Computer Science from the University of Illinois, Urbana, Illinois in 1999. His research interests include Object Oriented Programming, frameworks, reuse, and patterns. He also has some industrial consulting experience. Dr. Chai is a member of the ACM.

**Tan Chuie Hong** received a B.A. Hons. (Math) and an M.App. Statistics from University of Malaya, Kuala Lumpur, Malaysia in 1999 and 2001. Her favourite research interest includes demography, focusing on the statistical data analysis.