Investigating Focused Techniques for Understanding Frameworks

Victor Basili, Gianluigi Caldiera, Filippo Lanubile, and Forrest Shull
Department of Computer Science, University of Maryland
{basili | gcaldiera | lanubile | fshull}@cs.umd.edu

1. Introduction

An object-oriented framework is an OO class hierarchy augmented with a built-in model which defines how the objects derived from the hierarchy interact with one another. Thus, a framework is more than a class library: it is a generic solution within a problem domain because the model of interaction is domain-specific. A framework is tailored to solve a particular problem by customizing its abstract and concrete classes. The framework architecture is reused by all specific solutions in that problem domain. By providing both design and infrastructure for developing applications, the framework approach promises to develop applications faster [Lew95]. The most popular frameworks are in the GUI application domain (e.g., MacApp, ET++, CommonPoint) and in the drawing domain (e.g., HotDraw, UniDraw), but frameworks have also been developed in other domains such as multimedia, manufacturing, financial trade, and data access.

Developing an application by using a framework is closer to maintaining an existing application than to developing a new application from scratch. In framework-based development, the static and dynamic structures must first be understood and then adapted to the specific requirements of the application. The main difference between maintenance and framework-based development is that traditional maintenance is often performed on legacy systems with a decayed design, while frameworks are the result of many iterations and much effort aimed at improving the design. Most existing design patterns [Gam95] have been derived from experience in developing frameworks.

It is assumed that the effort to learn the framework and develop code within the system is less than the effort required to develop a similar system from scratch. Although it is recognized that the effort required to learn enough about the framework to begin coding is high, especially for novices [Tal95, Pree95], little work has been done in the way of minimizing this learning curve.

We have planned a study aimed at understanding the process of learning such a framework (or more generally, any unfamiliar system) and developing techniques that may minimize the effort expended on program understanding in particular situations. We have experimented before with focused reading techniques aimed at uncovering defects in software requirements documents [Bas96]. We are currently in the process of developing focused techniques that are aimed at providing programmers with a level of knowledge of a framework that would enable the use of the framework in developing a given application. We are performing a controlled experiment in which we test two focused reading techniques in a development context. We intend to study the characteristics of the context and the application task that make each technique more or less applicable to a given situation.

2. The Reading Techniques

Our approach is to generate families of reading techniques which depend on how an application is created using the framework. Each technique within the family is:

- tailored to the specific framework available
- detailed in that it provides specific steps to be performed
- focused on a particular coverage of the framework design and implementation

As a first step, we are considering two reading techniques for using a white-box framework [Sch96] to build new applications: a system-wide reading technique and a task-oriented technique. The main difference is the focus of the learning process: the system-wide technique focuses more on the "big picture" than on the detailed task to be accomplished (which is the focus of the task-oriented technique).

Programmers using a *system-wide* reading technique attempt to gain a broad knowledge of the framework design before adapting it to deliver the functionality required by the new application. They mainly read the framework documentation to learn about the abstract classes that form the basis of the framework, and the collaborative model that drives the behavior of the framework. Since many design patterns come from experience in designing frameworks [Gam95], they can be used as abstractions of the design characteristics of the framework. Example applications are used to better understand how the collaborative model works and how the design patterns are implemented. Programmers develop new applications mainly by specializing the abstract classes of the framework.

Programmers using a *task-oriented* reading technique attempt to locate the area of a framework directly relevant to the required change (task), and then gain a specialized knowledge of that part. They mainly use framework-based applications (examples) which show functionalities analogous to those of the application to be developed. Examples are interactively explored, with help of tools, to discover the specific features which characterize their functionality. The exploration of an example is driven by scenarios that can be thought of as use cases of the proposed application. The framework documentation is used to better understand the objects and interactions that are encountered during the exploration.

Both techniques look at the static structure and the run-time behavior of the framework, and both have access to the same sources of information. However, while the system-wide technique starts looking at the existing documentation, the task-oriented technique looks at the existing examples which are relevant for the required system.

3. A Controlled Experiment

To compare these two techniques with respect to their effect on framework learning and usage, we are undertaking a controlled experiment at the University of Maryland. We present graduate students and upper-level undergraduates, working in teams of three people, with an application task to be developed (an OMT object diagram editor) using a framework. One half of the class has been taught the system-wide reading technique and the other half the task-oriented reading technique.

The primary hypotheses for the experiment involve the different focus between the two techniques. We can expect that the novice users of the system can make changes faster using the task-oriented technique. The flip side of this hypothesis is that the system-wide technique requires more time per change for novice users, because of the overhead involved in gaining a broad understanding. However, experienced users of this technique have accumulated enough knowledge that they can make changes better than their experienced, task-oriented colleagues. In addition, it would be useful to look at the types of errors and the final systems that result from the use of each technique. We will also use the midterm and final exams to directly test the degree of framework comprehension which students have elicited after having been lectured on (midterm exam) and having applied (final exam) the different techniques.

We have chosen the GUI application framework ET++ [Lew95, Wei89] for the experiment. ET++ allows programmers to develop applications with consistent and highly interactive user interfaces that adhere to the desktop metaphor. ET++ has been implemented in C++ and runs on common UNIX platforms in different window environments such as X11. A programming environment is provided which includes code browsing and object inspection tools. A comprehensive set of example applications is also available, including both simple entry-level applications and more complex ones. The ET++ source code, the tools, and the sample applications are available for free via anonymous ftp. Furthermore, the design of ET++ has been thoroughly tested and improved from the initial version and incorporates seventeen of the design patterns in [Gam95]. ET++ is a sophisticated white-box framework that poses learning problems which can be major inhibitors against its use.

We already have provided the students with a general set of requirements of the application to be developed using the framework. First, students have been asked to develop object and dynamic models for the application requirements. Then, they have been asked to deliver the required system as incremental prototypes. The rationale for an iterative development process comes from the risk of project failures due to a combination of unknown development infrastructure and restricted schedule for delivery. The prototypes developed give us a good confidence that developers will deliver at least a part of the required functionality.

The controlled experiment contains two nested experimental designs. The first design can be synthesized as a randomized two-group design [Judd91] having the team as a unit of analysis. Using the notation for experiment design in [Cam63], the design takes the following form:

$$\begin{array}{ccc} R & X_1 & O_1 \\ R & X_2 & O_2 \end{array}$$

where each row represents a treatment group, R indicates random assignment, X represents the exposure of a group to a treatment (system wide or task-oriented), and O refers to observing and measuring the team performance in terms of both process and product (productivity, amount of functionality delivered and accepted, degree of framework reuse, and quality of delivered application). This design can be seen as a variation of the Posttest-Only Control Group Design [Cam63] where the comparison is made between a group given a treatment and a control group.

The second experimental design is a before-after two-group design [Judd91], having the individual as a unit of analysis. Using the same notation as before, the design takes the following form:

$$
\begin{array}{llll}
R & O_1 & X_1 & O_2 \\
R & O_3 & X_2 & O_4
\end{array}
$$

where each row represents a treatment group, R indicates random assignment, X represents the exposure of a group to a treatment (system wide or task-oriented), and O refers to observing and measuring the individual performance in terms of answers to comprehension tests. This design can be seen as a variation of the Pretest-Posttest Control Group Design [Cam63] where the comparison is made between a group given a treatment and a control group.

By comparing comprehension on the midterm exam (pretest) and final exam (posttest), we can test whether each technique has a significant short and long-term impact on program comprehension. Short-term impact will be measured during the midterm exam to verify that the training of the techniques was done effectively. Long-term impact will be measured during the final exam to understand if they have developed different skills in synthesis tasks. Writing or modifying a program are synthesis tasks, which require comprehension skills. It may be, however, that no difference is detected at the final exam between the techniques because students become adept over the semester at developing their own way of discovering necessary information not covered by their technique.

The analysis of quantitative data will be complemented by the qualitative analysis of the following information collected at different times of the project: (1) questions asked by students to project coordinators during the project, (2) feedback at the midterm and final exams; (3) comments in the weekly project forms; and (4) structured interviews at the final delivery of the required system.

4. Conclusions

We have described an empirical study aimed at increasing our knowledge of the relationship between techniques for understanding frameworks, or more generally any existing system, and the type of understanding achieved.

So far, we are halfway through the course of this semester-long experiment. At the end of the course project, we will be able to compare the results of the two groups in order to draw conclusions about whether the system-wide reading technique or task-oriented reading technique was more effective at gaining an initial understanding of a system suitable for developing an application.

5. References

[Bas96] V. R. Basili, S. Green , O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The empirical investigation of perspective-based reading", Empirical Software Engineering - An International Journal, vol. 1, no. 2, 1996.

[Cam63] D. T. Campbell, and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Houghton Mifflin Co., 1963.

[Gam95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., 1995.

[Judd91] C. M. Judd, E. R. Smith, and L. H. Kidder, *Research Methods in Social Relations*, 6th edition. Orlando: Holt Rinehart and Winston, Inc., 1991.

[Lew95] T. Lewis et al., *Object-Oriented Application Frameworks*, Manning Publications Co., 1995.

[Pree95] W. Pree, *Design Patterns for Object-Oriented Software Development*, ACM Press & Addison-Wesley Publishing Co., 1995.

[Sch96] H. A. Schmid, "Creating applications from components: a manufacturing framework design", *IEEE Software*, vol. 13, no. 6, November 1996.

[Tal95] Taligent Inc., *The Power of Frameworks*, Addison-Wesley Publishing Co., 1995

[Wei89] A. Weinand, E.Gamma, R. Marty, "Design and implementation of ET++, a seamless object-oriented application framework", *Structured Programming*, vol.10, no.2, 1989.