Investigating Reading Techniques for Framework Learning•

Forrest Shull[1], Filippo Lanubile[2], and Victor R. Basili[1]

[1]Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
College Park, MD, USA
{fshull, basili}@cs.umd.edu

[2]Dipartimento di Informatica
Universita' di Bari
Via Orabona, 4
70126 Bari, Italia
lanubile@di.uniba.it

ABSTRACT

The empirical study described in this paper addresses software reading for construction: how application developers obtain an understanding of a software artifact for use in new system development. This study focuses on the processes developers would engage in when learning and using object-oriented frameworks. We analyzed 15 student software development projects using both qualitative and quantitative methods to gain insight into what processes occurred during framework usage. The contribution of the study is not to test predefined hypotheses but to generate well-supported hypotheses for further investigation. The main hypotheses we produce are that example-based techniques are well suited to use by beginning learners while hierarchy-based techniques are not because of a larger learning curve. Other more specific hypotheses are proposed and discussed.

• This work was supported by NSF grant CCR9706151 and UMIACS.

1. INTRODUCTION

In almost any software development environment, the various work documents used (e.g. requirements documents, code and design plans) require continual review and modification throughout the lifecycle. This is due to the central role such documents play in many software engineering tasks (e.g. verification and validation, maintenance, evolution, and reuse). Software reading, i.e., the individual analysis of textual software work products aimed at achieving whatever degree of understanding is needed to accomplish a particular task, is thus a key technical activity for software development. We divide software reading activities into two distinct types: reading for analysis and reading for construction. Through our work in the Software Engineering Laboratory (SEL), we have evolved our understanding of reading for analysis, using a variety of experimental designs [Basili96a, Basili96b]. The experiment described in this technical report represents a further attempt to experiment with reading techniques in order to understand how widely applicable some of our conclusions from the earlier studies may be.

The reading techniques presented in this technical report are classified under Reading for Construction, which is aimed at answering the question: Given an existing artifact, how do I understand how to use it as part of my new system? Reading for construction is important for comprehending what a system does, what capabilities exist and do not exist; it helps us abstract the important information in the system. It is useful for maintenance as well as for building new systems from reusable components and architectures [Basili96b].

We chose to focus on the understanding of *object-oriented frameworks* as the artifact to be used in system development. An object-oriented framework is a class hierarchy augmented with a built-in model that defines how the objects derived from the hierarchy interact with one another to implement some functionality. A framework is tailored to solve a particular problem by customizing its abstract and concrete classes, allowing the framework architecture to be reused by all specific solutions within a problem domain. By providing both design and infrastructure for developing applications, the framework approach promises to develop applications faster [Lewis95]. 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.

The choice to focus on frameworks was motivated primarily by two reasons:

1. Frameworks are a promising means of reuse. Although class libraries are often touted as an effective means of building new systems more cheaply or reliably, these libraries provide only functionality at a low level. This forces the developer to provide the interconnections both between classes from the library and between the library classes and the system being developed. Greater benefits are expected from reusable, domain specific frameworks that usefully encapsulate these interconnections themselves.

2. Frameworks have associated learning problems that affect their usefulness. The effort required to learn enough about the framework to begin coding is very high, especially for novices [Taligent95, Pree95]. 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. As in maintenance, for a developer unfamiliar with the system to obtain this understanding is a non-trivial task. Little work has yet been done on minimizing this learning curve.

2. RESEARCH QUESTIONS

Since we approached this study from the viewpoint of software reading, our primary focus was on the processes developers would engage in, as they attempted to discover enough information about the framework to be able to use it effectively. We reasoned that the best approach would be to observe a number of different approaches or techniques and their effects in practice. From this information, we hoped to determine what kinds of strategies could be used, and for which situations they were likely to be particularly well- or ill-suited. Ultimately we hoped to gain an understanding of the deeper principles involved in framework usage by studying the interaction between the different techniques and the specific task undertaken.

Since our study took place in the context of a classroom assignment, we felt it necessary to give our students a starting point for using frameworks. In the absence of any empirical evidence or general agreement in the literature on the best way to teach developers how to use a framework, we selected two promising approaches. Each approach was the basis for a set of guidelines that was taught to half of the class, so that the strengths and weaknesses of the approaches could be compared. (We discuss the guidelines themselves and the process of their creation in a later section.) At the same time, we did not want to prevent the students from using work practices that they already knew, or discovered during the project, to be effective. Therefore, we allowed the students to modify these guidelines as desired. Our intention was to study the work practices of the students to determine when our guidelines were used, what other work practices were used, and how effective they were for particular tasks. We did not wish to constrain our subjects in any way to an artificial procedure, but to study and understand what they felt the most suitable approaches were for the problem. Our main research questions can be phrased as:

Can we identify strategies for learning frameworks?

What are their characteristics?

As it turned out, one of the learning approaches was viewed as too cumbersome for the environment of this study, and was not used significantly. Therefore, a straightforward quantitative comparison of the results of using each approach was not possible. Instead, most of the results presented in this paper come from a quantitative and qualitative analysis of the student experiences with learning and using the framework over the course of the semester. This type of information is useful for giving us a deeper understanding of what is important in learning to use frameworks.

3. RELATED WORK

A survey of the literature on frameworks shows that relatively little has been written on *using* frameworks (as opposed to building or designing them). Most of the work on using and learning frameworks tends to concentrate on strategies for framework designers to use in documenting their work. The primary weaknesses of this approach are that, first, the results are only applicable to frameworks for which the prescribed documentation has been constructed (that is, they do not directly contribute to general guidelines that would help developers in approaching any framework) and secondly, that usually very little empirical evidence is presented in order to demonstrate that the prescribed method is as effective as claimed. We present some of the main areas of framework documentation here and discuss representative papers for each. We would like to reiterate that we in no way see our study as competitive with these works. Our aim is not to suggest that these other approaches are right or wrong, or to present an alternative approach which we argue to be superior. Rather, we aim to provide a more low-level indication of what sources of information or types of activities are important in framework use - information which may be used to help identify weaknesses in higher-level techniques and focus them on aspects of the framework which are most important.

1. **Patterns and recipes:** Beck and Johnson [Johnson92, Beck94] advocate the use of "patterns" (interlocking descriptions of problem/solution pairs, similar to object-oriented design patterns [Gamma95] or cookbook recipes, e.g. [Apple86]) that describe a functionality supported by the framework, demonstrate how to implement the functionality, and discuss the impact of the implementation on the system) to describe frameworks. This seems a promising approach, because it seems capable both of showing the developer only as much detail as he or she needs for the current task and of directing the developer's attention to only the most relevant portions of the framework, and patterns have in fact been used as the sole form of documentation for the HotDraw framework. However, the only evidence presented as to its effectiveness is an informal study in which subjects were asked to learn HotDraw using patterns and provide feedback [Johnson92]. This study seems to have been very successful at its primary goal of helping the patterns' authors debug their work, but does not provide much detail as to how the learning process was influenced. Thus the presentation leaves unanswered questions as to whether any observed effectiveness was a function of the specific project undertaken or would be true in any environment.

   A related approach is the use of "hooks," which are meant to be similar to Beck and Johnson's patterns, although more structured and uniform and less narrative in style [Froehlich97]. Like patterns, each hook provides only the information necessary to solve a specific, focused problem. They are produced by the framework developer to illustrate how the framework is intended to be used.

   Johnson states [Johnson92] that it "would probably be worthwhile to try out the patterns in a controlled setting where it would be possible to watch how people use the patterns and what aspects of [the framework] are hard to learn." We feel that our study examines framework usage in exactly this way, although as we did not want to make *a priori* assumptions about the effectiveness of one method of documentation over another we did not work in an environment documented using patterns.

2. **Formal and/or searchable specifications of behavior:** Another tactic has been to formalize descriptions of the behavior of framework components, which then allows the creation of a search mechanism for finding useful components given a query. One such example is the prototype framework browser constructed at the University of Quebec [Mili97], which is especially promising in that it concentrates on finding a general solution which can be applied to any existing framework, regardless of the level of documentation supplied. However, Gangopadhyay and Mitra point out two major stumbling blocks for query-based learning of frameworks: first, searching for and reusing one component at a time does not allow the potentially subtle connections between components to be understood, and second, it is a very difficult problem to match a query which has been specified in a way meaningful to the developer with the description of the framework components [Gangopadhyay95].

3. **Architectural approaches:** Gangopadhyay and Mitra recommend instead a top-down approach to learning frameworks, by which they mean a concentration on the framework architecture rather than on individual components [Gangopadhyay95]. They recommend the development of exemplars, executable visual models that consist of instances of concrete framework classes along with explicit representations of their collaborations. An exemplar should contain at least one concrete subclass for each abstract class in the framework. This approach might prove difficult to use for frameworks that do not conform to good design style issues, namely having only a few abstract classes, and using abstract classes to implement important sites for customization in the framework. It is also unclear how helpful

the exemplar approach would be in cases in which the developer wants to make a modification the framework designer has not anticipated (i.e. a modification at a location that is not represented as an abstract class in the framework).

4. **Tutorials:** Other work has focused on tutorials created for users to follow which will presumably guide users through the most important points of the framework. (Two examples are [Vlissides91] for Unidraw and [Frei91] for ET++.) An interesting example of work in this area is Rosson *et al.*'s tutorial for learning Smalltalk [Rosson90] which applies Minimalist instruction techniques [Carroll90] and seems to corroborate the benefits that may result from a well-designed tutorial course (claiming to allow new users to develop code for interactive applications after only four hours). Like Johnson, Rosson undertakes some testing which is aimed not at testing hypotheses but at helping to debug the documentation. However, no study has been undertaken to examine the breadth of knowledge achieved, although this becomes an exceptionally pertinent question when the goal is radical decreases in learning time for a constant breadth of knowledge.

An important weakness which is shared by all of these approaches is that they assume that the framework developer will be able to anticipate future uses of the framework adequately to provide enough patterns (or exemplars, or tutorial lessons) in sufficient detail. An alternate research approach would be to avoid making any such assumptions about framework usage in order to undertake an empirical study of how developers go about performing the necessary tasks. Such an approach is not new, and has in fact proven useful in understanding how developers perform related tasks such as understanding code [vonMayrhauser95] or performing maintenance [Singer96]. Other authors [Codenie97] who have applied this approach to studying framework usage in industrial environments agree that, in most cases, framework customization will be more complex than just making modifications at a limited number of predefined spots. Documentation that assumes this is possible will be too constraining and will not provide support for many realistic development problems, which far from requiring isolated changes may sometimes even require changes to the underlying framework architecture.

Our study belongs in this category of empirical study of practical framework use. It is similar in type to the study undertaken by Schneider and Repenning [Schneider95], which draws conclusions about the process of software development with frameworks from 50 application-building efforts supervised by the authors. The large number of projects followed allowed the authors to examine both successful and unsuccessful projects, and their observation of (and sometime participation in) the process allowed them to both characterize the usual process and to identify some conditions that contribute to situations where the process breaks down and leads to unsuccessful projects. Our results complement and in some cases extend the results from the Schneider and Repenning study, and we consequently discuss them in greater detail later in a later section.

4. DEFINITIONS

We include the following definitions to clarify and illustrate some terminology that we use often in this discussion. It is our hope that these definitions will help to make clear our model of framework usage and to keep certain concepts distinct throughout the discussion. For example, we should first be careful to differentiate the **framework developer** (the developer(s) who designed and implemented the framework) from the **application developer** (the developer(s) who design and implement a new system using the framework to provide certain key functionality), who is usually referred to as simply the "developer" in this discussion.

example application: an application which has been constructed using the framework. Such examples may be created by the framework developer to illustrate how to produce some functionality using the components provided by the framework, or may be a "real-life" application in whose development process the framework happened to be used.

functionality supported by framework: a particular functionality is either provided by an example application, or there is a one-to-one mapping between the functionality and a framework component at some level of granularity (e.g. subsystem, class, method, ...). An example might be the behavior of radio buttons (primitive GUI objects would be likely to be supported by a class in a GUI framework) or a linked list (most frameworks provide classes that encapsulate reusable abstract data types).

functionality provided by examples: an example application exhibits dynamic behavior which corresponds to the required functionality. The functionality may be implemented in one component or a combination of components which contain some code specific to the example (i.e. some but not all of the functionality may be inherited from the framework).

functionality supported by object model: the required functionality can be implemented as part of the system represented by the object model without too much change to the object model. Obviously the phrase "too much change" is unacceptably vague, but we do not yet have a useful way of characterizing the concept of when adjustment to an object model becomes excessive. Although there are some high-level studies of software architecture underway, we look upon the effort to provide guidance as to how much adjustment is possible in a given situation as an important and promising area of future research.

5. INITIAL DESIGN OF FRAMEWORK LEARNING TECHNIQUES

In the absence of a definitive approach to framework learning, we turned to the literature to identify useful approaches. Our first step was to identify helpful models of the framework, that is, helpful ways of thinking about the framework that would highlight the truly important features and could be used for finding particular functionality. The most common description of a framework uses the class hierarchy to describe the functionality supported by the framework and an object model to describe how the dynamic behavior is implemented. Most of the common descriptions of a framework in the literature (e.g. [Lewis95], [Taligent95]) present a model of the framework similar to this one. To teach subjects how to use this model, we created a set of guidelines that could be used to gain an understanding of the class hierarchy. The guidelines help developers understand the functionality provided by the framework by concentrating on abstract classes in the hierarchy to provide an understanding of the broad classes of functionality, then proceed through deeper and deeper levels of concrete classes to find the most specific instantiation. We refer to this procedure as the *Hierarchy-Based* (HB) procedure, to emphasize that the underlying model of the framework is the class hierarchy.

As an alternative model, we decided to look at the framework through a set of example applications which, taken together, were meant to illustrate the range of functionality and behavior provided by the framework. Although a detailed examination of learning frameworks by means of examples has not been undertaken, learning by example also seemed a promising approach. Sets of example applications have been used to document some frameworks (the framework we used came with such a set) and the approach has been recommended for similar types of activities: such as learning effective techniques for problem solving [Chi87], or learning how to write programs in a new programming language [Koltun83, Rosson90]. It has also been argued that learning by examples is well-suited for "domains where multiple organizational principles and irregularities in interaction exist" [Brandt97], which may be a fair assessment of the large hierarchy of classes in a framework.

The framework we used in this study came with a set of examples at varying levels of complexity that was constructed to demonstrate the important concepts of the framework. To help subjects use these examples to learn the framework we created a set of guidelines that would guide exploration through the example set, to particular examples, to particular objects in the implementation, to particular lines of code in the object. This procedure is referred to as the *Example-Based* (EB) procedure.

Once we had identified suitable models, we constructed detailed guidelines for identifying functionality that would be relevant to the system being developed. To do this we concentrated on identifying similarities between the classes that were specified by subjects in their original object models, and the classes in the framework. These guidelines were then also tailored to the models, and integrated with the procedures for understanding the framework (see figure 1).

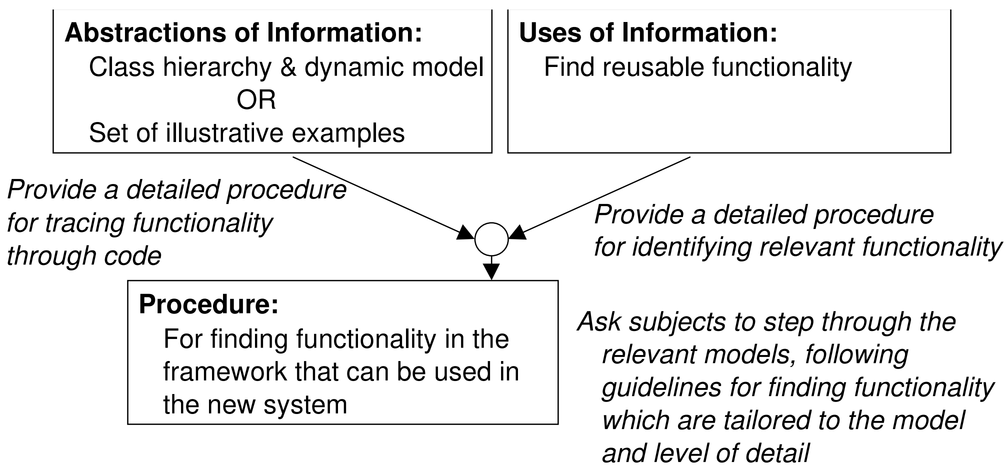

Figure 1: Producing focused, tailored procedures

The final guidelines were intended to be step-by-step procedures that could be taught to the students and used to find functionality in the framework. They are provided in an appendix.

6. DESCRIPTION OF THE STUDY

To undertake an exploratory analysis into framework usage, we ran a study as part of a software engineering course at the University of Maryland. Our class of upper-level undergraduates and graduate students was divided into 15 two- and three-person teams. Teams were chosen randomly and then examined to make certain that each team met certain minimum requirements (e.g. no more than one person on the team with low C++ experience) for the class. Each team was asked to develop an application during the course of the semester, going through all stages of the software lifecycle (interpreting customer requirements into object and dynamic models, then implementing the system based on these models). The application to be developed was one that would allow a user to edit OMT-notation diagrams [Rumbaugh91]. That is, the user had to be able to graphically represent the classes and different types of relations between them of a system, to be able to perform some operations (e.g. moving, resizing) directly on these representations, and to be able to enter descriptive attributes (class operations, object names, multiplicity of relations, etc.) that would be displayed according to the notational standards. The project was to be built on top of the ET++ framework [Weinand89], which assists the development of GUI-based applicatons. ET++ provides a hierarchy of over 200 classes that provide windowing functionality such as event handling, menu bars, dialog boxes, and the like.

Before implementation began on the project, the class was randomly divided into two groups and each was taught only one of the two framework models and its corresponding guidelines for use. During the implementation, we then monitored the activities undertaken by the students as much as possible in order to understand if our learning techniques were used, what other learning approaches were applied, and which of these were effective. To do this, we asked the students to provide records of the activities they undertook so that we could monitor their progress and the effectiveness of the techniques, and augmented this with interviews after the project was completed. (These data collection mechanisms are discussed below.) Although we felt our learning techniques made good starting points, we did not constrain students to use the techniques exactly as they were taught. We wanted to leave the students the flexibility to modify their development methods if the techniques were not well suited to a particular part of the implementation, or if they had personal techniques that would work better for them.

Since the analysis was carried out both for individuals and the teams of which they were part, we were able to treat the study as an embedded case study [Yin94]. Over the course of the semester, we used a number of different methods to collect a wide variety of data, each of which we discuss briefly below. Most of our collection methods are mentioned by Singer and Lethbridge in their discussion of the pros and cons of various methods for studying maintenance activities [Singer96], and we respond to some of their comments where appropriate. We hope this study provides an additional illustration of their conclusion that, in order to obtain an accurate picture of the work involved, a variety of methods must be used at different points of the development cycle in order to balance out the advantages and disadvantages of each method.

1. <u>Questionnaires</u> were used at the beginning (to report previous programming experience) and end (to report effort spent during the last week of implementation and the level of completion for each functional requirement for the project) of the semester. Although the information reported on the beginning questionnaires could not be verified, the end questionnaires were verified against the executables submitted for each team. The unit of analysis for the beginning questionnaires was the individual student, while the end questionnaires were filled out for the entire team. Both were mandatory although self-reported, and did not impact the students' grades.

2. <u>Exam grades</u> were recorded for certain questions on the midterm that could be used by us to gauge the students' level of understanding of framework concepts. These grades were recorded for the individual students, were assigned by us after evaluating the students' responses for the level of understanding exhibited, and were mandatory (as they constituted part of the students' grades for the course).

3. <u>Progress reports</u> were to be submitted by each team for each week of the implementation phase. They consisted of an estimate of the number of hours worked by the team for the week in implementing the project, and a list of which functional requirements had been begun and which completed. As the students were told that the progress reports had no bearing on their grades, many teams opted to submit them only sporadically or not at all. (In some ways, these reports were similar to Singer and Lethbridge's idea of logbooks, which allow the developer to record information at certain times throughout the development process. Singer and Lethbridge concentrate on the dangers of making the report too time-consuming, but we have noticed a quite opposite phenomenon: if the experimenter makes the report too minimal, the developer may assume that the information to be collected cannot be truly important and thus make completing the report a very low priority.)

4. <u>Problem reports</u> were requests for clarification or for help with ET++ that the students submitted (via email) to the course instructors. A record was kept of the general subject of each request, and by which team it had been submitted. In this way we hoped to maintain a record of the kinds of difficulties encountered by teams during the course of the project.

Problem reports were obviously not mandatory and had no effect on student grades, but were a resource that the students knew could be made use of at their discretion. (Singer and Lethbridge focus on the inaccuracies of retrospective reports, but our problem reports were actually an excellent way to get an accurate picture of where teams were having problems at the time they were having them - which may be, admittedly, unique to the classroom environment.)

5. Implementation score was assigned by us to each team at the end of the semester. Projects were graded by assessing how well the submitted system met each of the original functional requirements (on a 6 point scale based upon the suggested scale for reporting run-time defects in the NASA Software Engineering Laboratory [SEL92]: "required functionality missing", "program stops when functionality invoked", "functionality cannot be used", "functionality can only partly be used", "minor or cosmetic deviation", "functionality works well"). The score for each functional requirement was then weighted by the subjective importance of the requirement (assigned by us) and used to compute an implementation score that reflects the usefulness and reliability of the delivered system.

6. Final reports were collected from each team at the end of the semester. These reports consisted of documentation for the submitted system (object models and use cases) as well as records of the activities undertaken while implementing the project (object models of examples that had been studied, lists of classes that had been examined for some functionalities). Additionally, in-class presentations were given by each team in which they could present interesting details of the functionality available in their system, their experiences and difficulties with the techniques they used, and/or their general approach to implementation. The completeness of the final reports counted toward each team's grade, although their conformance to any particular technique did not.

7. Self-assessments were mandatory ratings in which each student was asked to rate the effectiveness of each member of his or her team (including him- or herself) as well as the team performance as a whole. Partly this was to detect if every team member had done their share of the work, and partly it was to ask students to think about what they had done rightly and wrongly during the course of the implementation. Although it was mandatory that each student return a self-assessment, they did not count directly toward the student grade (although in some cases, evidence from the self-assessments and the interviews led to individual grades being slightly adjusted).

8. Interviews were mandatory "debriefing" sessions at the end of the semester. Each team would come as a group to the course instructors, to be asked questions about what kinds of activities they did during the course of the semester, which of these they found particularly useful or useless, and what parts of the project were easiest and hardest. The original list of interview questions is found in the appendix, although additional questions were conducted in a dynamic manner. That is, the course of the interview was directed in new directions by us as unforeseen but interesting themes were raised.

Table 1 summarizes the data we collected along with the collection methods we used.

Aspect of Interest	Measures	Form of Data	Unit of Analysis	Collection Methods
Development Processes	Techniques used	Qualitative	team	interviews, final reports
	Tools used	Qualitative	team	interviews
	Team organization	Qualitative	team	interviews, self-assessments
	Starting point for implementation	Qualitative	team	interviews, final reports
	Difficulties encountered with technique	Qualitative	team	problem reports, self assessments, final reports
Product	Degree of implementation for each functionality	Quantitative	team	implementation score, final reports
Other Factors Influencing Effectiveness	Effort	Quantitative	team	progress reports, questionnaires
	Level of understanding of technique taught	Quantitative	individual	exam grades
	Previous experience	Quantitative	individual	questionnaires

Table 1: Types of measurements and means for collecting.

7. ANALYSIS

We then analyzed this mix of qualitative and quantitative data to gain some insight into what was going on within each team. By comparing and contrasting teams, we began to see implications that addressed our research questions. Since there has not yet been a large amount of work spent on understanding this area of framework use, our focus was on using this information to look for tentative but reasonable hypotheses and not on testing known hypotheses. The process of building theories from empirical research has been first proposed in the social science literature [Glaser67, Eisenhardt89] but it is also followed in the software engineering discipline [Seaman97].

7.1. Development Processes

The analysis approach we used was primarily a mix of qualitative and quantitative, in order to understand in detail the development strategies our subjects undertook. Our first step was to get an overview of what development processes teams had used. (By "development processes" we mean how the team had been organized, what techniques they had used to understand the framework and implement the functionality, whether they based their implementation on an example or started from scratch, and what tools they had used to support their techniques.) To this end, we performed a qualitative analysis of the explanations given by members of the teams during the interviews and final reports, and on the self-assessments. We first focused on understanding what went on within each of the teams during the implementation of the project. We identified important concepts by classifying the subjects' comments under progressively more abstract themes, then looked for themes that might be related to one another. Once we felt we had a good understanding of what teams did, we made comparisons across groups to begin to hypothesize what the relevant variables were in general. This allowed us to look for variations in team effectiveness that might be the result of differences in those key variables, as well as to rule out confounding factors.

In order to illustrate this analysis technique better, let us consider a small example from the analysis process. While trying to categorize the types of remarks students made during the final interviews, we noticed a lot of comments (some made spontaneously, some with prompting by the interviewers) concerning how teams spent most of their time during the implementation phase of the project. We grouped some of these remarks into a general category that deals with what kinds of activities students found useful in implementation; combined with other categories (which deal with, for example, what kinds of tools students found useful or what kinds of examples were helpful to examine) we began to get a better idea of what techniques students developed to help them in implementation. To get an accurate picture of what teams did over the entire course of the semester, however, we needed to look at other categories of remarks, concerning for example which of the two initial procedures (HB or EB) the team was taught, what their experiences were with the technique, and what parts of the technique were found not to be useful and were discarded. By making such abstractions from the students' comments, we built an understanding of what each separate team did.

With this understanding of the processes at work within teams, we compared experiences across teams to try to identify themes that emerge. For example, we noticed that most teams who began by modifying an example tended to do better than teams who began implementing from scratch. Our next step of the analysis was to test these provisional hypotheses, again by making comparisons across teams to find possible refuting evidence or confounding factors. For example, suppose Team X started their implementation from an example but turned in a very poor implementation in the end. Does this refute our provisional hypothesis? Perhaps it signifies that the trend we thought we had observed was simply a fluke, or perhaps we may notice a confounding factor - say, Team X was also very poorly organized - that may account for the seemingly anomalous results and needs to be taken into account as part of the analysis.

Although this is not a common method of analysis in computer science, it is a recommended approach for social sciences and other fields that require the analysis of human behavior [Eisenhardt89, Miles79]. It is well suited for our purposes here because our variables of interest are heavily influenced by human behavior and because we are not attempting to prove hypotheses about framework usage, but rather to begin formulating hypotheses about this process, about which we currently know little.

We found that teams used development processes that fell into 1 of 4 categories (Table 2). While no team used the HB technique for the entire semester (although these guidelines were partly incorporated into some of the hybrid techniques that were used), the EB technique did enjoy consistent use, both as taught and in combination with other techniques. It can be noted that even teams who were taught HB and not exposed to EB tended to reinvent a technique similar to EB on their own. (Some teams were even contrite about this. "We didn't realize at the time that this was the technique taught to the other part of the class, but it seemed the natural thing to do.")

Category	Number of Teams	Number Originally Taught EB, HB	Description
EB	5	5, 0	Students in this category used the EB technique as it was taught, following the guidelines as closely as they could.
EB/HB	5	0, 5	This is a hybrid approach that focuses on using examples to identify classes important in the implementation of a particular functionality. The main difference from the EB technique is that, in the hybrid technique, the student does not always begin tracing the functionality through the code, but may instead use the example to suggest important classes and then return to the framework hierarchy to focus on learning related classes.
ad hoc EB	4	2, 2	This was an ad hoc approach that emphasized the importance of learning the framework via examples, but ignored the detailed guidelines given. The primary difference between these techniques and the EB category is that ad hoc EB techniques are missing a consistent mechanism for selecting examples and tracing code through them.
EB/scratch	1	1, 0	The team used the EB technique to identify basic structure and useful classes, but implemented the functionality mostly from scratch.

Table 2: Description of development processes observed in the study.

7.2. Potentially Confounding Factors

We also undertook quantitative analyses to gauge the effects of potentially confounding factors in the study.

Noticing the difficulties teams were having with the HB technique, our first concern was that we might have in effect penalized one half of the class by teaching them a technique that was not useful to their current environment. We undertook an analysis of the number of person-hours spent by different groups in order to understand if the amount of effort a team spent on implementation was largely dependent upon the technique they had been taught. The test for differences in the two groups could not be conclusive, as only six teams reported their overall effort data, and a team with organizational difficulties had to be again discarded from the analysis as an extreme outlier (as defined by [Ott93]). The analysis did, however, show no significant difference between the average amount of effort spent on the project over the course of the semester by teams who had been taught each of the different techniques (p-value of 0.5859 obtained from t-test). Student remarks from the interviews tended to support this. Most teams who had been taught the HB technique reported that they switched their development approach usually after trying to apply HB for the first 1 to 2 weeks. Regardless of the technique a team had been taught, the heaviest investments of effort for almost all teams came toward the end of the implementation phase.

We also wished to examine whether students actually had a different initial understanding of the framework based on the models and procedures we had taught them. We gauged their initial understanding by means of two questions that appeared on their midterm, after they had been taught one or the other of the framework models, but before they had actually had any experience using the ET++ framework. The first question was intended to measure how well the students grasped the concept of the framework hierarchy of classes (Table 3 shows the distribution of grades on this question by procedure taught). The second question measured understanding of the model of interaction (Table 4). We tested whether response rates were independent of the procedure taught by using Wilcoxon rank sum tests. The results of neither test were significant ($p = 0.4881$ and $p = 0.5259$ for the first and second questions, respectively) showing that there is no difference in how well the questions are answered with respect to the technique taught. This shows that, although taught different procedures for using the framework, neither group of subjects started out at a disadvantage to the other in terms of their understanding of the framework itself.

		Grade Achieved			
		A	**B**	**C**	**D**
Technique	**Example-Based**	**5**	**5**	**5**	**6**
Taught	**Hierarchy-Based**	**9**	**3**	**4**	**6**

Table 3: Distribution of grades on exam question dealing with the framework class hierarchy.

		Grade Achieved			
		A	**B**	**C**	**D**
Technique	**Example-Based**	**10**	**3**	**2**	**6**
Taught	**Hierarchy-Based**	**6**	**7**	**4**	**5**

Table 4: Distribution of grades on exam question dealing with the framework model of interaction.

A final concern in all studies of this type has to do with the experience of the subjects. We were concerned that the effectiveness of our teams might have more to do with the level of experience the team members had with implementing similar projects than with any of the variables under study in our experiment. We removed the extreme outlier from analysis and used the Pearson correlation coefficient [Hatcher94] to measure the strength of the linear relationship between experience and implementation score (with scores close to 1 or -1 representing an exact linear relationship and scores tending to zero representing no linear relationship). We found no correlation between the total amount of experience of a team with programming in an academic environment and its effectiveness at the implementation of the project (Pearson's correlation coefficient of -0.0015), but did uncover a correlation between total experience programming in industry and effectiveness at implementation (Pearson's correlation coefficient of 0.55, Figure 2). However, an Rsquare value of 0.31 for the model means that industrial experience accounted for only 31% of the observed variation in the implementation score.

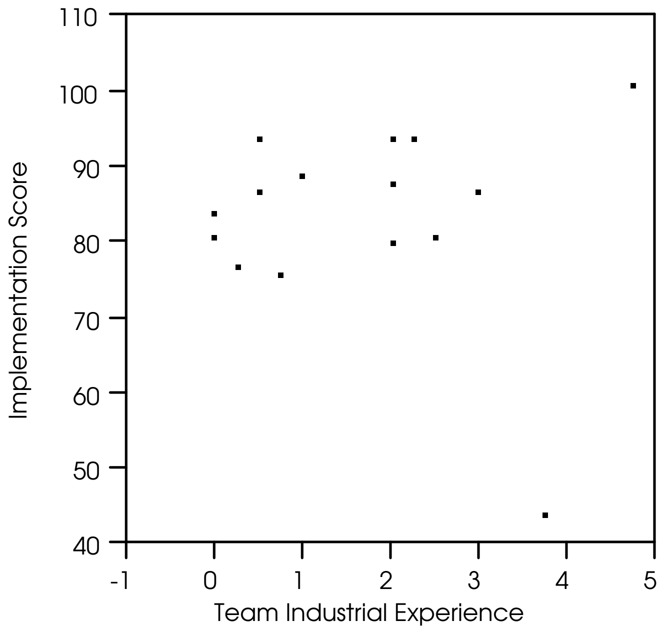

Figure 2: The total industrial experience of a team (in years) and its correlation with its effectiveness at implementation of the project (the team with implementation score of 44 has been removed from analysis as an outlier).

8. Results

In this section we present the hypotheses that result from our study of framework usage. Along with each we present the relevant indications from our study, so that the reader may judge the strength of the evidence which supports these hypotheses.

8.1. Hypotheses About Our Models of the Framework

In order to understand the process of learning a framework by means of examples, we can generalize not only from the EB procedure itself but from its example-based derivatives as well. From Table 1, it can be seen that all students who were taught EB ended up using this technique (or a close derivative) throughout the implementation phase. Perhaps more importantly, all students who were taught the other technique ended up employing a less rigorous example-based technique on their own. It seemed, therefore, that not only our EB technique, but example-based learning in general, was a natural way to approach learning such a complicated new system. This leads us to formulate:

HYPOTHESIS 1: Example-based techniques are well-suited to use by beginning learners.

In contrast, subjects tried to use the HB procedure but eventually abandoned it. Qualitative analysis was necessary to understand why this happened. What was wrong with the HB procedure that made it not useful in this environment? We analyzed the student remarks from the problem reports, self-assessments, and final reports to see if characteristic difficulties had been reported. A common theme (remarked upon by half of the teams who had been taught HB) was that the technique gave subjects no idea which piece of functionality provided the best starting place for implementation, or where in the massive framework hierarchy to begin looking for such functionality.

Teams also registered complaints about the time-consuming nature of the HB technique - especially compared to an example-based approach where implementation can begin much more rapidly, hierarchy-focused approaches seem to require a much larger investment of effort before any payoff is observed. One team pointed out that they had explicitly compared their progress against other (as it happened, Example-Based) teams: "We talked to other groups, and they seemed to be getting done faster with examples. So after the first week we started going to examples, too."

Despite these difficulties, students reported that they felt that the HB technique would have been very effective if they had had both sufficient documentation to support it and more time to use it. "The Hierarchy-Based procedure would be helpful if you have the *time*," said one group in the final interviews, "but on a tight schedule it doesn't help at all." Another opinion was expressed by the group who said, "It's the technique I normally use anyway - and it would have been especially good here when the examples are not enough for implementing the functionality." There seemed to be a consensus that it would have allowed them to escape from the limitations of the example-based approach and engage in greater customization of the resulting system, but simply wasn't effective in the current environment. Five teams were able to create effective strategies that were hybrids of the hierarchy-based and example-based methods (EB/HB). However, the lack of guidance as to how to get started, and the time required to learn the necessary information to use it effectively, meant that no development teams used it exclusively for a significant portion of the implementation phase. (By no means was this a completely negative development, as we now have more detail on techniques that minimize that crucial learning curve.)

HYPOTHESIS 2: A hierarchy-focused technique is not well-suited to use by beginners under a tight schedule.

8.1.1. Practical Implications for Using an Example-Based Procedure

The analysis in the last section should not be taken to imply that our EB procedure was without problems. We also undertook a qualitative analysis of subject satisfaction with the Example-Based procedure in order to understand where it could be strengthened for future use. We found that there were also characteristic problems with EB that were encountered by beginners.

Although the teams using this technique usually managed to get the functionality working in the end, almost all (4 out of 5) of the groups in the study who used our EB technique reported difficulties in finding the necessary functionality within the example set. The problems with finding it in the first place seemed especially acute when the functionality needed was a very small part of a much larger example (e.g., a characteristic way of displaying items onscreen, or the dialog boxes discussed under "key functionalities", below). This indicates a problem with our EB technique – further guidance is required to assist developers in finding and extracting small pieces of functionality embedded within larger examples. As there is currently little guidance available in this regard, more work will be have to be done in this area to enable effective example-based techniques.

This study also provides indications that characteristics of the example set can influence the performance of an example-based approach as well. One-fourth of all the teams in our study had trouble making use of the examples because the example set provided did not conform to a consistent organization or structure. Some examples were based on Model-View-Controller interaction [Goldberg83] while others were not constrained by any such separation of functionality, and different examples seemed to achieve the same functionality by using different classes from the framework hierarchy. As others [Rugaber90] have pointed out, learning how to implement functionality from existing applications is difficult because the rationales for design choices, which explain why the finished implementation looks the way it does, are usually not included in the documentation. When attempting to reuse functionality from existing applications, developers are implicitly asked to reconstruct the choices that led to the finished implementations they are studying. This situation can actually be made worse in a framework-based environment, where effective reuse requires the developer to understand the rationales behind a number of applications, not just one. This is a problem which will have to be addressed by any example-based technique.

HYPOTHESIS 3: The effectiveness of an example-based technique is heavily dependent on the quality and breadth of the example set provided.

8.2. Hypotheses About the Level of Specificity in the Procedures

This study provides us with an excellent opportunity to understand whether the procedure we created was at a useful level of specificity. Because some teams followed the procedure exactly while others (i.e. EB/HB, ad hoc EB, EB/scratch) followed it only to a certain extent, we can compare these two groups to understand if the procedure at its current level of detail is useful.

To perform this analysis, we focused on certain *key functionalities*, that is, certain requirements for which there was a large degree of variation between teams in terms of the quality of the implementation. Recall that we had graded each required piece of functionality for each project on a 6-point scale. To select key functionalities we looked for functionalities for which there was enough variation among all 15 of the teams on the rating scale (regardless of what type of technique they used) that teams could be easily divided into at least two groups based on their score. We then looked for important attributes (such as techniques used, types of problems reported, what kind of starting point was used for implementation) that seemed to correlate with teams which were able to implement a particular functionality in a more complete or sophisticated way. In each case, the differentiating variable seemed to be the level of detail at which the procedure was followed.

To back up this qualitative analysis, we attempted to use statistical tests to verify the correlation. Since both of the variables in this analysis (technique followed and result from implementation) are on a nominal scale (i.e. expressed as categories) we can organize data into contingency tables where each dimension corresponds to a variable and numbers represent frequencies. We want to test whether the proportion of teams in each type of implementation varies due to the type of technique that was used. Specifically, the null hypothesis is that the proportion of teams who achieved each type of implementation is independent of the technique that was used to perform the implementation. In order to test this difference we apply the chi-square test of probability[1]. Due to our small sample sizes and the exploratory nature of this study, we used an α-level of 0.20, which is higher than standard levels. We also present the product moment correlation coefficient, *r*, as a measure of the effect size [Judd91]. (An *r*-value of 0 would show no correlation between the variables, whereas a value of 1 shows a perfect correlation.) We realize that these tests do not provide strong statistical evidence of any relationship, but instead see their contribution as helping detect patterns in the data that can be specifically tested in future studies.

We identified 4 such key functionalities: links, dialog boxes, deletion, and multiple views. They illustrate 3 types of situations that may arise when functionality is being sought in examples:

---

[1] We base our use of the chi-square test, rather than the adjusted chi-square test, on [Conover80], which argues that the adjusted test “tends to be overly conservative.”

1. **The examples don't provide all of the functionality desired.** Key functionality 1 (links) fits into this category. OMT notation requires that links be drawn between classes in the model which interact in some way with each other. The specifications for our OMT editor stated certain other specifications for handling links, such as how they should be entered into the diagram, and that they should automatically update when their associated classes are moved. Provided with the example set was the "er" entity-relation diagram editor which provided similar functionality that allowed two linked objects to be represented by means of a line that connected their centers. Although useful as a starting point, this implementation was not sophisticated enough for the project, because the same two classes in an OMT diagram may be connected by multiple links. If these are all represented by lines drawn from center to center, every link between these classes will overlap. There is no example provided with ET++ that shows functionality that explicitly addresses this concern.

   About half of the teams implemented the link functionality as in the er example. Of the teams that implemented a more sophisticated version that allowed links to be uniquely represented (i.e. multiple links between two classes do not overlap), there were a number of solutions: randomly displacing links by a small offset, recording the point where the user drew the link across the boundary of the class, and breaking the line down into a series of line segments which may then be individually positioned.

		Technique Followed	
		EB	**Modified-EB**
Result of	**Sophisticated**	1	6
Implementation	**Simple**	4	4

Table 5: Number of teams achieving sophisticated versus simple implementation of links, whether using EB or a modified version of the EB technique.

   Almost all (4/5) of the teams who used the EB technique implemented the less sophisticated version of the functionality found in the er example. By comparison, less than half (4/10) of the teams who used a modified version of EB turned in the less sophisticated implementation (Table 5). A chi-square test of independence was undertaken to test whether the level of sophistication was dependent on the type of technique used, although we recognize that the small number of data points involved can lead to some inaccuracies in the results [Ott93]. The test resulted in a p-value of 0.143, which is statistically significant at the selected α-level. An *r*-value of 0.38 confirms that this shows a moderate correlation between level of sophistication and type of technique [Hatcher94]. From this example we hypothesize that:

   HYPOTHESIS 4: A detailed Example-Based procedure can cause developers to not go beyond the functionality that is to be found in the example set.

2. **The functionality was completely contained in (perhaps multiple) examples.** Key functionalities 2 and 3 (dialog boxes and deletion) provide evidence that the EB technique performed about the same as the variant techniques in this case.

   For dialog boxes, examples existed which showed how to create dialog boxes containing graphical devices (e.g. text fields, radio buttons) and how to use them to display and store information. The difficulty was that this functionality was spread piecemeal over multiple examples and students had a hard time finding and integrating all of the functionality they needed. About half of the class (7/15) managed to get the dialog box functionality working correctly and interfaced with the rest of the system (Table 6). All techniques were distributed roughly equally between the teams who did and did not get this functionality working correctly. The chi-square test here yielded a p-value of 0.714, for which the related *r*-value is 0.10. This confirms that response levels are very likely effectively equal between the two categories.

		Technique Followed	
		EB	**Modified-EB**
Result of Implementation	**Fully correct**	2	5
	Incomplete	3	5

Table 6: Number of teams who achieved fully correct versus incomplete implementations of dialog boxes, whether using EB or a modified version of the EB technique.

For deletion, there was at least one example that clearly contained functionality to delete classes and links from a diagram. All teams were able to implement this functionality. Getting the functionality to support the ability to undo or redo a deletion was apparently more challenging, however, although the examples covered this as well. Partly, this may have been due to students simply forgetting to implement this part of the functionality since it was not explicitly mentioned in the requirements (although adequate testing should have revealed the need to interface correctly with this standard ET++ functionality!).

Teams basically implemented deletion in one of three ways:

1. The ability to undo or redo was integrated fully with deletion.

2. The ability to undo or redo was not implemented for deletion, but deletion was implemented in such a way that a later call to undo or redo would not cause a core dump (this *minimally* satisfied the requirements for the project, in letter if not in spirit!).

3. The ability to undo or redo was not implemented at all for deletion; deletion could be used but a later use of undo or redo could cause a core dump if the program tried to manipulate an object that had been deleted.

Again, the different techniques for learning frameworks seem pretty equally distributed among these three categories (Table 7). The chi-square test for this functionality yielded a p-value of 1, with an associated *r*-value of 0, which is not significant and indicates that response rates are exactly equal regardless of the type of technique used.

		Technique Followed	
		EB	**Modified-EB**
Result of Implementation	**Impl. 1**	1	2
	Impl. 2	2	4
	Impl. 3	2	4

Table 7: Number of teams achieving each of the three levels of implementation for deletion, whether using EB or a modified version of the EB technique.

From these two examples we hypothesize:

HYPOTHESIS 5: When the functionality sought is contained in the example set, Example-Based techniques will perform about the same, regardless of the level of detail provided.

3. **The examples provide a more sophisticated implementation than is required.** Key functionality 4 (views) fits here. The requirements for the OMT editor stated that the program must provide multiple views of the currently opened document. Examples existed which satisfied the project's requirements about views (that they update automatically, be independently scrollable, and allow resizing). However, there were also examples that gave an even more sophisticated implementation that allowed views to be dynamically added and deleted.

		Technique Followed	
		EB	Modified-EB
Result of Implementation	Sophisticated	3	9
	Simple	2	1

Table 8: Number of teams achieving sophisticated versus simple implementation of views, whether using EB or a modified version of the EB technique.

All but three teams chose the more sophisticated implementation. Two of these three teams turning in less functionality used the EB technique (Table 8). The chi-square test resulted in a p-value of 0.171, which is significant at the selected α-level. An *r*-value of 0.35 shows a moderate correlation between the variables and allows us to hypothesize:

HYPOTHESIS 6: When the example set contains functionality beyond what is required for the system, a sufficiently detailed Example-Based procedure can help focus developers on just what is necessary.

The practical consequences of this analysis may be that the level of detail that is appropriate in an Example-Based procedure is strongly dependent on the breadth of the example set provided. We hypothesize that the more detailed a procedure is, the more it focuses the developer on using only functionality provided by the examples. This may be because developers become too reliant on the examples and do not understand the system at a sufficient level of detail to implement effectively from scratch when necessary. Alternatively, it may be that integrating functionality written from scratch becomes more and more difficult when more and more of the system is taken from examples that someone else has written.

Of course, there are other benefits to providing additional detail in such procedures. Among the most important may be packaging experience to guide new developers. For example, in our study we noticed that half of the teams who used a variant of the EB procedure wasted time and effort during the course of the implementation phase by having to re-implement some functionality that they had implemented previously in a short-sighted way. Since only 1 of the 5 teams who used our EB procedure exactly reported the same difficulty, we feel that a definite benefit of the procedure was that it helped guide our subjects to focus more on features which were important to the implementation as a whole, rather than just the portion they happened to be working on at a particular time.

8.3. Hypotheses About Beginning Implementation

Because we did not constrain the students as to how they went about implementing the functionality, we could also study whether the way in which the implementation was begun had an effect on the rest of the implementation process. Some teams decided to start from scratch, beginning with no functionality on top of the framework and slowly growing the code as they determined how to implement specific functionalities. Other teams chose to start with an example which seemed to contain some of the functionality they would need in the finished system. They then modified the functionality that was there and added whatever else was required to produce the finished system. We wanted to see if one approach tended to get more functionality working more reliably (as measured by the team's implementation score).

All teams who started from an example chose the same one, a simple entity-relation diagram editor (known as "er"). It was similar to the OMT editor to be developed, but much simpler: as specified for the OMT editor, the ER diagram editor allowed simple shapes to be added to an editable document, and allowed these shapes to be selected, moved, and associated with one another. However, more sophisticated functionality, such as entry of attributes via dialog boxes and the maintenance of separate regions within a particular shape, were lacking.

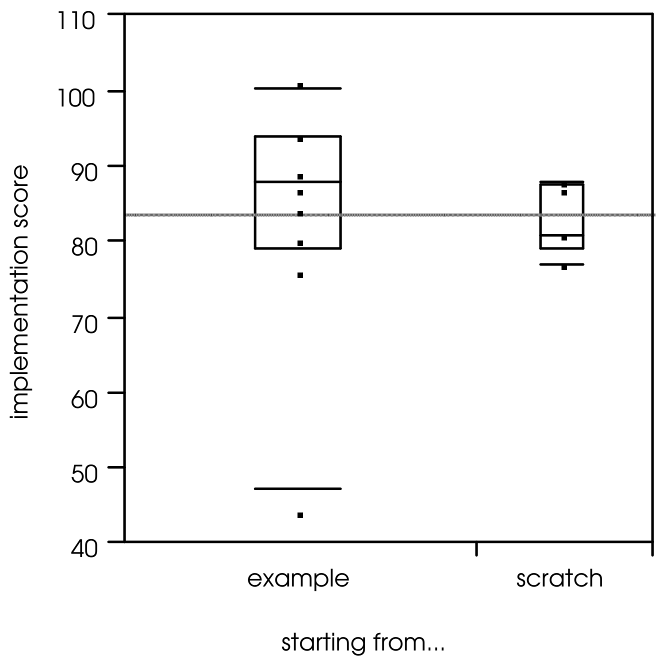

Figure 3: Team implementation scores for teams starting from scratch versus teams starting by modifying the "er" example. 90%, 75%, 50%, 25%, and 10% quantiles are shown. The team with implementation score of 44 has been removed from analysis as an outlier.

We used a t-test to determine whether teams starting from the "er" example tended to perform significantly better than teams starting from scratch (Figure 3). One point, representing a team which experienced severe organizational difficulties which were primarily responsible for a very low implementation score, must be removed from this analysis as an extreme outlier (according to the definition given in [Ott93]). The test yielded a p-value of 0.15, which is significant at the 0.20-level and provides some evidence that teams who started by modifying an example tended to be more effective than those starting from scratch. This indicates that, overall, the benefits of relying on an existing example as a starting point (which may include being able to exploit an existing file structure and to model new classes on similar ones which already exist in the example) outweigh the negatives (the extra work involved in identifying relevant functionality and removing irrelevant code).

HYPOTHESIS 7: Teams who begin their implementation using an existing example for guidance are more effective than those who begin implementing from scratch.

It was especially noteworthy that every one of the teams who started from an example used the "er" editor as a starting point. There was, in fact, a second example application which could conceivably have served as a starting point also. This second important example application was a small drawing editor that seemed to provide most of the functionality needed for the OMT editor, as well as much extraneous functionality. It is very interesting that no teams chose the drawing editor as a starting point. Of the teams who commented on this decision during the final interviews, most seemed to agree with the reasoning that the drawing editor just had too much "extra functionality and complicated code that we do not need", and that the large amount of extraneous functionality would be too confusing to enable the code responsible for the functionalities of interest to be easily separated out. If we had been able to compare the performance of teams who had started from each of the examples, we might have been able to draw useful conclusions about what attributes of an example make it a better or worse starting point for implementation. This is a promising direction for future work.

8.4. Hypotheses About the Importance of the Object Model

From the qualitative analysis of interviews and self-assessments, there were some general indications about the importance of the object model:

- 6 of our development teams mentioned – without being prompted by us - the importance of their object models as guides to implementation. 3 of these teams reported that they had been able to stay fairly close to their original object model of the system during the course of implementation. All of these teams ranked in the top half of the class with regards to implementation score. The remaining 3 teams were reporting problems; they had strayed from their original model during implementation. It seems that this inability to follow the model had some negative effects, as all 3 were ranked in the bottom half of the class. (Since 5 of these 6 teams had all received the same grade on the original model, it seems unlikely that the variation in performance could have been caused by factors outside the implementation phase, such as the quality of the model itself.)

- 2 of the 3 teams who were not able to follow their object model also reported difficulties due to the inconsistent nature of the examples.

Throughout this study we have observed the central importance of the object models in development. Although we assume that object models are important and necessary parts of any software development effort, their interaction with framework development seems even more noteworthy. When using frameworks, there is the important question of whether to modify the object model of the system, so as to exploit a piece of functionality offered by the framework that might not exactly fit the original plan, or to keep the object model "as is", even if it makes implementing the application on top of the framework harder.

The general indications from the interviews and self-assessments leads us to:

HYPOTHESIS 8: Teams who were able to stay close to their original object model of the system during implementation seemed more effective.

It seems that one cause of trouble may have been that these teams did not understand the examples well enough to adapt the functionality illustrated by the examples to the system being developed. Teams who implemented incorrect functionality may have been simply too willing to make modifications to their planned system to accommodate the examples more easily.

Schneider and Repenning [Schneider95] present an interesting study of framework use that comes to similar conclusions. What we describe as straying from the original object model, they identify as projects for which "the application design process had been driven by features of the framework," a condition which is described as being present in some development efforts which "ended up with really messy designs, or were cancelled." As we noticed through observation of our teams, they identify the likelihood that a team will have to reimplement some functionality as one of the major negative consequences arising from this situation: "Premature design decisions made during the feature-driven phase can corrupt application system architecture or require abandonment of much work." They further point out that the developer's temptation is to deal with the easier problem of adapting functionality provided by the framework first, leaving more difficult functionality (which is not guaranteed to fit nicely into the framework-based development) for later and making breakdowns all the more devastating when they occur.

Some of the successes of our EB technique may also have come from its treatment of the object model. We noticed in our analysis of problems that a much higher percentage of teams using variants of EB reported wasted effort due to incorrect implementations than teams using EB. We attribute this to the EB technique's focus on the object model as a guide for implementation, which may not have been so evident in ad hoc variants.

These results create an interesting tension with our experiences with example-based techniques, which seem to imply that reuse of framework functionality is always a positive thing. Taken together, our conclusions indicate that, within proper bounds, exploiting functionality from the framework and example set is the most helpful direction - otherwise, it can be very bad indeed. Schneider [Schneider95] draws a similar conclusion: although overuse of framework functionality can lead to negative effects, as described above, exploitation of the low-level framework features is a sensible trend that presumably pays off, when used within bounds. Discerning just where these bounds may lie is a crucial question that awaits further study; for now, we conclude only that development difficulties are liable to appear unless the object model of the system is well supported by the framework and its example set (i.e., the framework permits an implementation of the system that does not require major deviations from the object model).

9. Answers to Research Questions

In this section, we relate our observations to our specific research questions for the study.

1. Can we identify strategies for learning frameworks?

Hypotheses 1 and 2 address this question.

The example-based technique was identified as an effective strategy in our environment. While we cannot say in all cases that an example-based learning approach would be superior to one based on the class hierarchy and model of interaction, the indication of this study was that for novice users, the examples were a more effective way to learn. Within the category of example-based techniques, we further differentiate "strictly adaptive" from "ad hoc adaptive." The EB technique is considered strictly adaptive because it was focused entirely on guiding the developer to find useful functionality in the example applications, which could then be tailored to the current system. The ad hoc techniques augment this basic approach with other techniques that the students found to be effective for them. Although these techniques share many common features, our analysis of their use in practice discovered certain characteristic differences. This leads us to classify them as separate strategies.

Although the hierarchy-based approach cannot be deemed effective in our environment, as it was abandoned and not used to implement any of the semester projects, we cannot assume that a hierarchy-based approach is always inferior. The most important environmental condition appeared to be the subjects' familiarity with the particular framework being used. Several of our subjects had recognized benefits of the HB technique but were unable to apply it due to their lack of familiarity. They expressed this in comments during the interviews such as: "The HB procedure was more similar to what I normally do, but…" or "I found the examples limiting in some ways and thought the HB procedure would address this problem, but…" It is possible that, if our subjects had had more experience with the framework, the HB procedure would have proven better suited to their needs. Indications are that hierarchy-based procedure required more experience with the framework to be used effectively.

2. What are the characteristics of these learning strategies?

Since the subjects of our study only had significant experience with the EB technique, we can report only on the characteristics of example-based strategies. (Our observations on this subject were recorded in hypotheses 3 through 8.) We identified two main types of example-based strategies: strictly adaptive and ad hoc adaptive, each with its own strengths and weaknesses. The relative effectiveness of each seems to be most strongly determined by how closely the object model of the system to be developed corresponds with the existing applications.

Hypothesis 5 expresses our observation that when the functionality called for by the object model is well-contained in the set of existing applications, just about any example-based technique should be helpful. However, as illustrated by hypothesis 4, a strictly adaptive technique can't take the developer far beyond what is provided by the existing applications themselves. In a situation in which the set of applications is sparse and does not contain the necessary functionality, an ad hoc technique may be more appropriate. As hypothesis 6 indicates, if the set of applications is particularly large, then a strict adaptation technique may be most helpful. Despite its weaknesses, such a technique in procedural form was shown to guide the developer toward implementing the object model "as is" and away from "gold-plating," or spending time providing extra features that seem nice but are not necessary.

From the experiences of our beginning learners, we also have evidence about other characteristics that are required by example-based techniques in order to be successful. Hypothesis 3 indicates that future studies need to be undertaken to determine if we can add better guidance for helping developers find functionality in existing example applications. Hypotheses 7 and 8, respectively, may indicate that example-based techniques should guide developers to begin their implementation from an existing application, if a suitable one can be found, and to stay closely to the original object model once implementation has begun. (It is possible that, as developers get more experience with the framework, it may be possible to synchronize the design of the system more closely to the framework infrastructure from the beginning, thereby minimizing the problem of mismatch between the system design and the framework. Our study did not address this possibility.)

10. Threats to Validity

There are three tests which can be considered to evaluate the quality of any empirical study: construct validity, internal validity, and external validity [Judd91].

10.1. Construct Validity

Construct validity aims to assure that the study correctly measures the concepts of interest. The main problem is that variables never measure only the construct of interest but also other extraneous sources of variation. One tactic to enhance construct validity is triangulation: the use of multiple sources aimed at corroborating the same fact or phenomenon [Yin94, pp.90-94].

In our study we applied data triangulation, by including multiple measures for the same aspect of interest and different collection methods for the same measure (Table 1).

10.2. Internal Validity

Internal validity aims to establish correct causal relationships between variables as distinguished from spurious relationships. Although a case study cannot have the same internal validity as a controlled experiment, because the investigator has little control over events, there are analysis techniques that can strengthen the internal validity, even for exploratory studies like this.

We made inferences using the qualitative analytic technique described in [Eisenhardt89]. It consists of performing a within-case analysis, to gain familiarity with each case and find emerging patterns, followed by cross-case analysis, to look for similarities and differences between cases. Although this is not a common method of analysis in computer science, it is a recommended approach for social sciences and other fields that require the analysis of human behavior [Eisenhardt89, Miles79]. It is well suited for our purposes here because our variables of interest are heavily influenced by human behavior and because we are not attempting to prove hypotheses about framework usage, but rather to begin formulating hypotheses about this process, about which we currently know little.

A specific threat to internal validity might be that we have constructed one of the techniques incorrectly, which would explain the differences in performance. As regards the example-based technique we used, EB, we attempted to minimize the odds of making this mistake by basing the specific technique on the example set which is provided by the framework's authors themselves. For the hierarchy-focused technique, HB, we based our model of the framework on the most important facets of the framework definition. We feel that the use of the inheritance hierarchy means that our model is complete because all functionality provided by the framework must be encapsulated in one of the classes of the class hierarchy, with the inheritance relations showing how the functionalities provided are related to one another. Thus, while there may be other models more adept at supporting framework learning, we feel confident that our model is adequate to the job.

We also undertook quantitative analyses to test the effects of potentially confounding factors (such as differences in effort spent, understanding achieved, or previous experience) which could be rival explanations to our findings.

10.3. External Validity

External validity aims to assure that the findings of the study can be generalized beyond the immediate study. Although generalization can be achieved only through replication in multiple studies, we believe that our findings are relevant for a larger population than this single study.

A first threat to external validity might be that professional developers would have behaved differently than the students that we used as the subjects of the study. Certainly, this is always a danger in studies of this sort. However, in this case we feel that this difference would not be a strongly significant one. Although the level of industrial experience in the class was not high, all students had experience both programming in the language used (C++) and in object-oriented techniques. More importantly, even professional developers would almost certainly have been novices in terms of the use of the ET++ framework, so that the most immediately applicable experience would not have significantly varied in either case.

A second threat to external validity might be that our findings are tied to the framework we used, ET++. Although we cannot completely rule out this threat to validity, ET++ has been thoroughly tested and improved from the initial version and it incorporates seventeen of the design patterns in [Gamma95]. From this point of view, we consider ET++ representative of the class of sophisticated white-box frameworks that pose learning problems, which can be major inhibitors against their use.

11. Directions for Future Research

Perhaps the most interesting result of this study is the indication that hierarchy-based and example-based techniques are not competing, but must be used together for effective framework use, with the one extending and complementing the other. There is no meaningful way to compare and contrast their effectiveness, because they are not suited for the same types of tasks; the interesting question is not "Which technique is better (more effective, more time-efficient)?" but "For which types of tasks is this technique best suited?" As our study has shown, applying the wrong technique to a task can result in an incorrect or incomplete implementation of the required functionality, or a needlessly time-consuming search through the framework class hierarchy.

Whether or not we can give guidance in these respects is an important question. We have evidence from the study that applying a stage at the wrong time can be a critical error. Several teams said they wasted valuable time implementing and then reimplementing functionality, because the teams had made choices that were good in the short run but not for the larger system. In many cases, these mistakes were the result of having paid insufficient attention to the object model,

A new experiment in this area should focus on, first, improving both the example-based and hierarchy-based techniques, and secondly, putting bounds on the type of task to which each is best suited. To do this we will first redesign the EB and HB techniques to incorporate what we have learned about their weaknesses in this study, and then run a new experiment aimed at uncovering their interaction with the type of task undertaken.

The EB technique should still concentrate on the object model, but more guidance must be given for finding functionality in the example set. Additionally, the technique should explicitly refer the reader to the HB technique when the functionality has been discovered to be unsupported by the framework. HB in contrast must explicitly rely on EB to focus it on a useful portion of the framework hierarchy.

For the next experiment in this area, a smaller case study may be more appropriate than another controlled experiment. A study using protocol analysis (e.g. [vonMayrhauser95]) would be able to focus in more detail on the specific process developers go through when attempting to determine whether a functionality is supported by the framework. Protocol analysis would also result in a deeper understanding of what kinds of conditions lead to a situation in which developers run into difficulty with the technique. With this information it would be possible to further strengthen the techniques and to propose more detailed guidelines as to when example-based techniques should be used, and when hierarchy-focused.

ACKNOWLEDGEMENTS

Our thanks to Gianluigi Caldiera for his invaluable assistance designing and running this experiment as a part of the course CMSC 435 (Fall 1996) at University of Maryland, College Park. Our thanks also go to the students of CMSC 435 for their cooperation and hard work.

APPENDICES

A. Example-Based Procedure

TERMINOLOGY:
Application refers to the program you are building - here, the OMT diagram editor.
Example refers specifically to one of the example applications presented with ET++, each illustrating some aspect of the framework.
Original object model is the object model of Project 1, which models the OMT editor system without taking ET++ into account.
Original dynamic model is the dynamic model of Project 2, which models the OMT editor system without taking ET++ into account.

DESCRIPTION:

1. **Based on the increments, define the functionality to be sought and find a suitable example.**

   INPUTS: Functionality of application, set of examples
   OUTPUT: Example containing functionality and set of use cases.

   a. Run the examples provided. Select for further examination the example that contains the maximum coverage of the functionality sought. If there isn't one example that contains the entire range of functionality, find the largest set of functionality that is covered by one example, and concentrate on implementing just that set for now. Record the example used and your rationale for selecting it.
   b. Compare the original object model to the example. Which classes from your original object model seem to have a corresponding object in the example? (Run the example and explore its range of functionality to determine what components are likely to be objects: What window components exist? What components respond to user events? Build use cases to illustrate the system functionality you discover.)
   c. Use your original dynamic models for the classes identified in b to determine which operations are supported by the objects in the example. Are there operations in your original dynamic models which are missing from the example?

2. **Execute and study the example selected, with respect to the functionality being sought. Build an object model of the example, selecting appropriate features.**

   INPUTS: Example and set of use cases, original object model
   OUTPUT: Event traces and object model for example

   a. Based on the increments, find the set of use cases for the functionality in this example which is relevant to the OMT system. For each use case, build an event trace that shows how the classes you identified above interact.
   b. Run the example. For each event trace:
      i. Find an object onscreen that participates in the event trace. Remember that objects in ET++ can be user-created objects, window components, etc.
      ii. "Inspect-click" on the object you have identified. This brings up a list of all objects existing at that point on the screen. Select the topmost object listed

that handles the functionality, and edit its source. Now we have reached the code for a class responsible for handling a portion of the functionality sought.

iii. Use the class definition and implementation to discover how this class handles the events you have identified in this event trace. Search through the code for this class to discover the attributes which are modified and the methods which are called; if you cannot locate a class operation which handles an event, remember to use the hierarchy browser to investigate the code for the ancestor classes as well. Does the implementation require objects which you have not identified in your original object model? Update the event trace correspondingly if so.

iv. Using the information discovered, incorporate this class into the object model of this example. Record the attributes and operations you encounter as you trace through the code. Record relationships between classes.

v. While there are still classes in this event trace that haven't been fully documented, select another class from the event trace. Access its code either through "inspect-clicking" or directly through the code browser. Return to step iii.

vi. Once all of the relevant classes in all the event traces for all use cases are in the object model, go on to step 3.

3. Add new needed features.

INPUTS: Event traces and object model for example, original object model
OUTPUT: List of classes from example to add to application

a. Compare the object model for the example with your original object model. Determine which classes from the example will have to be added to your original model. Determine which attributes and operations are missing from your original object model that will have to be added to the class definitions.

4. Instantiate solution.

INPUTS: List of classes from example, application
OUTPUT: New version of application

a. Add new classes to your program. Modify the class definitions and implementation from the example to fit your application.
b. Add new attributes and operations to existing classes. Find the code corresponding to the initialization of the attributes and the necessary operations; modify it as appropriate to fit your application. Write the code which is necessary to extend the functionality for your application or to integrate the new functionality with the existing system.
c. Return to step 1 to implement the remaining functionality.

The following must be turned in:

1. List of examples chosen and rationales for their selection.
2. Use cases, event traces and object models for the relevant functionality of each example. The order in which you created the event traces must be indicated.

B. Hierarchy-Based Procedure

TERMINOLOGY:

Application refers to the program you are building - here, the OMT diagram editor.

Original object model is the object model of Assignment 1, which models the OMT editor system without taking ET++ into account.

Original dynamic model is the dynamic model of Assignment 2, which models the OMT editor system without taking ET++ into account.

DESCRIPTION:

1. **Given ET++, study the class hierarchy.**

   INPUTS: ET++ hierarchy, ET++ architectural description
   OUTPUT: List of high-level, abstract classes containing useful functionality.

   a. Identify which portions of the ET++ architecture (basic building blocks, application classes, graphical building blocks, system interface layer) contain functionality that will need to be used or modified for your application.
   b. Using your assessment of step a, identify abstract classes that can be used in the new application.

2. **Identify classes related to the problem.**

   INPUTS: List of abstract classes, original object model
   OUTPUT: List of ET++ classes to be used in application

   a. For each class in your original object model, find the appropriate ET++ class to subclass it from. Use the hierarchy browser to help you understand how attributes and operations are inherited. Use the code viewer to examine the details of classes. Document your work as you proceed.
      i. For each abstract class you identified in 1b, examine its attributes, operations, and relations to other classes.
      ii. If the abstract class is a good match to classes in your original object model, this class can be used to subclass classes in your application. Record this class in your documentation and explain which of its attributes and operations are relevant to your application. Examine its children for an even closer match. Continue examining its siblings, or return to the parent class if there are no more siblings to examine.
      iii. If the current class contains functionality which is too specific and cannot be adapted for your application, do not consider it any longer. Record this class in your documentation and explain (either in terms of functionality or attributes/operations) why it was rejected.
   b. If you cannot find an ET++ class that provides all of the operations required by a class from your object model, it is possible that your class will have to be formed as an aggregation of two or more ET++ classes. Find ET++ classes that cover subsets of the operations you need and begin thinking of ways to combine these classes.

3. **Modify your original object model for the application, using classes from ET++.**

   INPUTS: List of ET++ classes, original object model
   OUTPUT: Modified object model

a. Update your original object model to show which of your classes will be subclassed from ET++ classes (as identified in step 2a) and which will be composed of aggregates of ET++ classes (as identified in step 2b). Make sure that all relevant attributes and operations (as provided by ET++) are included.
b. If you have identified relevant classes in the ET++ class library which were not a part of the original object model, update the original model to include these classes and any appropriate links.
c. Check that the modified object model corresponds to the model of interaction provided by ET++.
d. This modified object model must be turned in.

4. **Based on the modified models constructed in step 3, develop the system, exploiting the classes offered by ET++.**

   INPUTS: Modified object model
   OUTPUT: Implementation of application

   a. Develop the application, subclassing and aggregating where possible to make use of the attributes and operations offered by ET++ classes.
   b. Write the code necessary to extend the functionality to the specific application, adding your own classes as needed.

The following must be turned in:

1. List of classes examined and rejected, with rationales.
2. Modified object model, showing classes provided by ET++ with their relevant attributes and operations.

C. Questions for Final Interview

The final questions were asked to every team during the final interview, with the researchers conducting the interview asking for additional detail as needed. In some cases, additional information was requested (such as the team's experience with the object model) as interesting but unforeseen points were raised. The term "guidelines" refers to the procedure (example-based or hierarchy-based) which the team was given to use, while "increments" refers to the functional requirements for the system. (The requirements were presented as building one upon the other, and we gave the students an example of how the system could be built incrementally.)

Final Interview Questions

Team: ________

1. Describe how the effort you spent on this project is distributed over the 5 weeks of implementation.

   What happened the last week - did you spend more effort, less effort, or about the same as in previous weeks? Why?

2. How much did you rely on the tools provided as part of ET++?

   How much did you rely on the ET++ documentation?

What other things did you do?

3. How closely did you follow the guidelines?

   If not closely, when did you depart from them? Did you follow the other technique, or come up with something ad hoc?

4. Which increments were especially hard? Which were especially easy? Why do you think this was so? (If they don't mention increment 10, ask them about it explicitly. [Increment 10 required a dialog box to be brought up for data entry, which would then modify the diagram appropriately. From the problem reports, we knew many subjects found this a challenging requirement.])

5. For the most part, were you able to find the functionality you needed, or did you implement most of it from scratch?

REFERENCES

[Apple86] *MacApp Programmer's Guide*. Apple Computer, 1986.

[Basili96a] "The Empirical Investigation of Perspective-Based Reading", V. Basili, S. Green , O. Laitenberger , F. Lanubile, F. Shull, S. Soerumgaard, and M. Zelkowitz, Empirical Software Engineering: An International Journal , vol. 1, no. 2, 1996.

[Basili96b] V. Basili, G. Caldiera, F. Lanubile, and F. Shull. Studies on Reading Techniques. In *Proc. of the Twenty-First Annual Software Engineering Workshop*, SEL-96-002, pages 59-65, Greenbelt, MD, December 1996.

[Beck94] K. Beck, R. Johnson. Patterns Generate Architectures. In *Proc. ECOOP'94*, Bologna, Italy, 1994.

[Brandt97] D. S. Brandt. Constructivism: Teaching for Understanding of the Internet, *CACM*, Vol. 40, No. 10, October 1997, pp. 112-117.

[Carroll90] J. Carroll. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press, 1990.

[Chi87] M. Chi, M. Bassok, M. Lewis, P. Reimann, and R. Glaser. Self-Explanations: How Students Study and Use Examples in Learning to Solve Problems. Technical Report UPITT/LRDC/ONR/KBC-9, University of Pittsburgh, 1987.

[Codenie97] W. Codenie, K. DeHondt, P. Steyaert, A. Vercammen. From Custom Applications to Domain-Specific Frameworks, *CACM*, Vol. 40, No. 10, October 1997, pp. 71-77.

[Conover80] Conover, Practical Nonparametric Statistics, 2nd Edition, NY: John Wiley & Sons, 1980.

[Eisenhardt89] K. Eisenhardt. Building Theories from Case Study Research, *Academy of Management Review*, 14 (4), 1989.

[Frei91] C. Frei and H. Schaudt. *ET++ Tutorial: Eine Einführung in das Application Framework.* Software Schule Schweiz, Bern, 1991.

[Froehlich97] G. Froehlich, H. Hoover, L. Liu, and P. Sorenson. Hooking into Object-Oriented Application Frameworks. In *Proc. of the 19th International Conference on Software Engineering*, pp. 491-501, Boston, MA, May 1997.

[Gamma95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Object-Oriented Software Architecture*. Addison-Wesley, 1995.

[Gangopadhyay95] D. Gangopadhyay, S. Mitra. "Understanding Frameworks by Exploration of Exemplars," In *Proc. of 7th International Workshop on CASE*, pages 90-99, July 1995, IEEE Computer Society Press.

[Glaser67] H. G. Glaser, A. L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Publishing Company, 1967.

[Goldberg83] A. Goldberg, *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley, Menlo Park, 1983.

[Hatcher94] Hatcher, L. and Stepanski, E.J. 1994. *A Step-by-Step Approach to Using the SAS® System for Univariate and Multivariate Statistics.* Cary, NC: SAS Institute Inc.

[Johnson92] R. Johnson. Documenting Frameworks with Patterns. In *Proc. OOPSLA '92*, Vancouver, BC, October 1992, SIGPLAN Notices, 27(10): 63-76.

[Judd91] C.M.Judd, E.R.Smith, and L.H.Kidder, Research Methods in Social Relations. Holt, Rinehart and Winston, Inc., Forth Worth, sixth edition, 1991.

[Koltun83] P. Koltun, L. Deimel Jr., and J. Perry. Progress Report on the Study of Program Reading, *ACM SIGCSE Bulletin,* Volume 15, Number 1, February 1983, pp. 168-176.

[Lewis95] T. Lewis *et al., Object Oriented Application Frameworks.* Mannings Publication Co., Greenwich, 1995.

[Miles79] M. Miles, Qualitative Data as an Attractive Nuisance: The Problem of Analysis, *Administrative Science Quarterly*, 24(4): 590-601, 1979.

[Mili97] H. Mili, H. Sahraoui, I. Benyahia. Representing and Querying Reusable Object Frameworks. In *Proc. of the Symposium on Software Reusability,* Boston, May 1997.

[Ott93] R. Ott. *An Introduction to Statistical Methods and Data Analysis*, Duxbury Press, Belmont, CA, 1993.

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

[Rosson90] M. B. Rosson, J. M. Carroll, and R. K. E. Bellamy. SmallTalk Scaffolding: A Case Study of Minimalist Instruction. In *Proc. CHI '90*, April 1990.

[Rugaber90] S. Rugaber, S. B. Ornburn, and R. J. LeBlanc, Jr. Recognizing design decisions in programs. IEEE Software, 7(1): 46-54, January 1990.

[Rumbaugh91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W Lorensen. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Schneider95] K. Schneider, A. Repenning. Deceived by Ease of Use: Using Paradigmatic Applications to Build Visual Design Environments, In. *Proc. of the Symposium on Designing Interactive Systems*, Ann Arbor, MI, 1995.

[Seaman97] C. B. Seaman, V. R. Basili. An Empirical Study of Communication in Code Inspection. In *Proc. ICSE'97*, Boston, MA, 1997.

[SEL92] Software Engineering Laboratory. *Recommended Approach to Software Development, Revision 3*, SEL-81-305, June 1992.

[Singer96] J. Singer and T. C. Lethbridge. Methods for Studying Maintenance Activities, in *Proc. of 1st International Workshop on Empirical Studies of Software Maintenance*, Monterey, CA, 1996.

[Taligent95] Taligent, Inc. *The Power of Frameworks,* Addison-Wesley, New York, 1995.

[Vlissides91] J. Vlissides. *Unidraw Tutorial I: A Simple Drawing Editor*. Stanford University, 1991.

[vonMayrhauser95] A. von Mayrhauser and A. M. Vans. Industrial Experience with an Integrated Code Comprehension Model, *Software Engineering Journal*, September 1995.

[Weinand89] A. Weinand, E. Gamma, and R. Marty. Design and Implementation of ET++, a Seamless Object-Oriented Application Framework, *Structured Programming*, 10 (2), 1989.

[Yin94] R. Yin, *Case Study Research: Design and Methods*, Sage Publications, London, 1994.