FrameScan: Exploring O-O Framework Usage

Amr Kamel **Garry Froehlich** **Paul Sorenson**
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada, T6G 2H1
1 780 492 1564
{amr, garry, sorenson}@cs.ualberta.ca

ABSTRACT
While much research has been conducted on the design of object-oriented frameworks, few usability studies have been done with frameworks to determine how software practitioners can understand and use frameworks. The empirical study presented in this paper addresses how software practitioners obtain an initial understanding of a framework and how they build and maintain framework expertise within the project group.

The objective of this stage of our study is not to reach final conclusions about framework usage, but rather to generate hypotheses for further investigation in later stages of the study. It lays the foundation for the development of continuous studies leading to: i) understanding requirements of tool support for effective large-scale usage, ii) the feasibility of creating and maintaining an experience bases around the framework and iii) better understanding of the problems developers face in order to develop better framework documentation.

Keywords
object-oriented frameworks, software reuse

1 INTRODUCTION

From the very beginning of the discipline (McIlory [26]), building software systems out of reusable components has been a prime goal of software engineering discipline. Object-oriented application frameworks (frameworks for short) are becoming an increasingly popular strategy for component-based software development. A framework comprises an architecture that targets a wide range of applications within a specific domain, and an implementation of that architecture [38]. Frameworks can be commercially available products such as MFC or in-house developments used to support software product lines [6], [31], [32], [1]. Details of in-house frameworks are typically hidden to maintain competitive advantages.

In commercial frameworks, and in many proprietary frameworks, the framework is often developed by one group of practitioners and used by a different group [4]. Furthermore, to overcome its development cost, a framework is used by not one, but many development groups in a product-line. This emphasizes the importance of presenting the framework to its potential users in an easy to understand way. Yet, frameworks are generally difficult to understand [5]. A typical user is required to invest a lot of time and effort to learn and understand the framework before using it [28], [38]. Some researchers estimated the effort required as being equivalent to that of maintaining an existing application [31].

While building and designing frameworks has been extensively discussed in literature, few studies have been conducted to address frameworks' usability, with the goal of minimizing the learning curve of new users. FrameScan is an ongoing study with the goal of understanding how software practitioners can effectively and efficiently understand and deploy framework technology to construct and evolve different applications. Our study focused on white-box frameworks [18] documented using hooks [12]. A white-box framework is extended using inheritance to drive new classes and writing application specific methods. Hooks provide a structured template that describes in a pre-defined grammar the changes necessary to adapt the framework to fulfill a particular requirement. An example of a hook in a common user interface framework is the actual means of adding a new item to a menu (e.g. add the name, produce the method that responds to the event, etc.). The choice of white box frameworks was motivated by their wide use in industry [6]; the choice of hooks was motivated by their ease of automation [26] and [17]. Furthermore, the framework is intended a coarse-grained component in an application, rather than being a basis for the entire application.

The remainder of this paper will discuss some of the background of frameworks and framework usage (section 2), and then will detail the setup of our study and some of the initial results from it. In section 3 we discuss three questions we wanted to answer with the study, and we then carry those questions through the paper, first generating a number of interesting hypotheses based on our data from the study, and generating some initial conclusions. We then describe how future studies in the series will be setup to help answer some of the questions.

2 BACKGROUND

Little work has been done on addressing how to use a framework as opposed to how to design and build one. Most of the work on learning and using frameworks focuses on how a framework designer can document his/her framework and present it to its potential users. The purpose of the framework, its design and intended-use are the three roles proposed by Johnson [18] that documentation should fulfil for frameworks. In [1] Butler et. al., a number of types of documentation are discussed that are targeted towards one or more roles.

Hooks [12], motifs [23], cookbooks [22], tutorials [31] and exemplars are different documentation techniques targeting how a framework is meant to be used. The basic concept promoted by most of this work is to think of the framework in terms of the functionality it provides, and to document it in terms of related services. Each hook (a recipe in the cookbook, etc.) describes a service along with a demonstration of how to extend the framework to provide its functionality in developing an application. Exemplars [16] are used to show a prototypical examples of the framework functions and provide some means of tracing its control flow. The idea is to present to the user the framework's hot spots [27], where the framework should be modified, while maintaining the frozen spots intact. The underlying assumption is that framework developers will be able to adequately anticipate future uses of the framework and provide enough documentation for all these uses. This assumption may not hold true in many cases.

Empirical studies have shown that framework customization is more complex than just modifying a limited number of predefined spots [8]. Furthermore, these studies inferred that documentation that assumes the possibility of limiting the changes to a few points is too restrictive to provide the required support for many realistic problems. This promoted a maintenance style approach to framework learning. Studies adopting this approach [33] make no assumption about the hot spots of the framework. The framework users are responsible for where and how to modify the framework to support their application. In this approach, framework users have access to the same set of documentation available for the framework developers. In addition, they have access to different example applications built using that framework. The underlying assumption is that using a framework needs the same level of understanding needed for developing and maintaining it. Typically, developing and maintaining a framework requires substantially greater investment of time and effort than that of a single application. Requiring framework users to gain deep understanding of the framework, equivalent to that of its developers is a lot of unnecessary overhead defeating the whole concept of framework reuse. Furthermore, adopting this approach severely limits framework evolution and forward compatibility among applications.

We are interested in conveying the intended use of the framework to application developers. Frameworks consist of hot spots (parts of the framework that can be configured or have custom application components added to) and frozen spots (parts of the framework that should remain fixed) [27] as shown in Figure 1. Documentation for the intended use such as hooks focus on these hot spots.

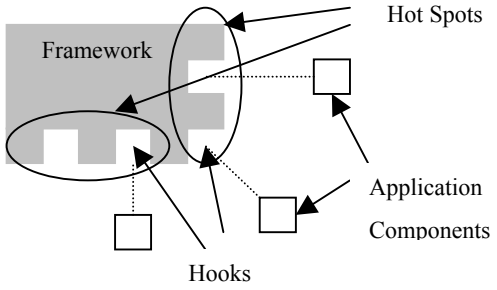

Figure 1: Hot Spots and Hooks

To overcome the problem of a limited number of predefined hot spots [8], we integrated a feedback loop. In our approach, input from framework users is used to modify the documentation for future users. Our thesis is that framework functionality can be better understood if the framework user's perspective is considered while writing the documentation. Furthermore, the documentation has to be continuously updated to include any usage that was not originally anticipated by the framework developers. Thus, understanding how the framework is actually used is of vital importance to achieve our goal.

3 RESEARCH QUESTIONS

The main goal of this study is to observe and understand issues related to framework usability from the framework users' standpoint. We reasoned that the best approach is to observe practitioners developing application with minimum interaction. From the information we collected, we hoped to gain an understanding of principles and issues related to developing applications using frameworks. We hoped to understand the development process and ultimately, recommend best practices, and build tools that support these best practices.

Our study took place in a classroom environment. A major challenge we had was to conduct the study, without perturbing the pedagogical goals of the course. We managed to achieve our goals by carefully planning the process steps of our study. For obvious reasons, we recommended what we envisioned as best practices and guidelines to the whole class. However, using these recommended practices was optional and we allowed students to make use of their expertise and experiences to modify these guidelines. We observed the processes

administered by the students and, determined how useful were our guidelines. The main research questions we had are:

1. *Is there an effective way to understand and learn how to develop applications using frameworks?*
2. *Does the use of hooks improve the understandability and usability of a framework?*
3. *Can technical reviews be used effectively to speed up the early stages of framework learning process?*

In order to formulate our hypothesis about these questions, we collected a wide range of information about the students and their academic and industrial background. We kept track, to the best of our ability, of all details that seemed affecting framework usage and understanding.

4 STUDY SETUP

Overview

To explore how frameworks are used, we ran a study as part of a software-engineering course at the University of Alberta in the spring term of 2000. A class of 34 computer science students was divided into six teams of five or six students each. Most of the study participants were at their senior year, and many them had worked outside the university on a sixteen-month industrial internship program (IIP)[1].

In order not to interfere with team synergy, students self-selected their team partners. To compensate for any bias, in collected data, that might result from the self-selection process (e.g. teams are not of equal capability), students background information, (e.g. courses taken, industrial experience, technical knowledge, etc.) was collected and considered during data analysis.

Over the time span of the course, each team was asked to develop a client-server application of their choice with the requirement of building the application using the CSF [14] framework. The developed applications were small to medium sized and covered domains like collaborative document development, long-distance learning and on-line gaming.

The CSF Framework.

The CSF (Client-Server Framework) is a small framework of approximately 50 Java classes developed to serve the purposes of this study. The framework facilitates persistence data management and platform-independent communication. The communication aspect allows objects within different programs running on different machines to exchange messages of any type and size. The mechanisms deployed are fairly simple and are not intended to compete with larger efforts such as CORBA.

[1] See www.cs.ualberta.ca/iip/

In order to facilitate its use, the framework comes with several types of documentation, covering all aspects outlined in [8] and [18]:

- Use-cases to give an overview of the use of the framework and points to individual hooks where developers have to provide their own classes or methods.
- Design documentation to provide a high-level overview of the major classes of the framework and their relationships to one another. This includes both class diagrams and collaboration diagrams along with textual descriptions.
- Hooks to document the framework's intended use. They show how and where the framework can be enhanced in order to meet application specific requirements.
- Examples show some specific uses of the framework and provide running code that the developers can experiment with.
- API and code to show details of methods, classes used in the framework. The source code of the framework was made available.

Additionally, the framework developer and the teaching team were available throughout the study to answer questions that arose.

In order to enhance our confidence in the framework's ability to provide its promised services, the framework was carefully designed. Commonality analysis was performed on other existing frameworks in the area along with other client server applications developed in the class. Design patterns [15] were incorporated where applicable. Furthermore, in a previous offering of the same course, two student groups had voluntarily used a beta-version of the framework in a limited manner. Revisions based on the results of this experience had enhanced the maturity of the CSF Framework.

Running the Study

Due to the lack of guidelines discussing framework deployment in literature, we had to develop our own, relying heavily on our experience from the previous offering of the course. The projects' lifecycle was separated conceptually into two phases, exploring the framework and using it to build an application. In the first phase, team members explore the basic functionality of the framework and set their reuse strategy. The objective of the first phase is to build some confidence within team members that the framework serves the purposes of their application. In the second phase, the framework is actually used to produce the application. Typically, the first phase is completed during project analysis and the second phase starts with the application design.

During the first phase, a CSF-expert (the developer) gave a three-hour overview on the framework distributed over two sessions. The sessions covered the framework design and its documentation style. The use of the framework was also demonstrated using a simple example application to give a concrete instance of the abstract classes of the framework.

In the third week of the course, a technical review was held with the purpose of swapping technical information between development teams and the CSF-expert. In view of their application, team members were asked to individually review the framework and prepare a list of questions and concerns. During a thirty minute collection meeting, each team had the opportunity to address their concerns and ask questions of the CSF-expert. Due to time limitations, discussion was kept to a minimum. Depending on the question/concern, the CSF-expert chose to either provide quick answers, or defer the answers until later. The rework session of the review produced a set of Frequently Asked Questions (FAQ) that was attached later to the framework documentation. Outcome of these reviews was the corner stone for our feedback approach discussed earlier.

For the second phase, the guidelines were provided as a set of deliverables at predetermined milestones. Each team had to produce an analysis document and a detailed design document. Deliverables to product testing, consisted of updated version of the two documents along with test plans, integration plans, reports on the process used, and user documentation.

A second technical review was held on the eighth week of the course. The purpose of this review was to find defects in the design documents. The two reviews followed a similar process, but they differed in roles, and reading technique. In the second review, the teaching team was the reviewer and student teams were the authors. Checklists were the recommended reading technique for the first review, while the second review used ad hoc technique. The checklist for the review was prepared according to guidelines.

The course was organized to mimic industrial setups, where the class instructors played the role of upper management. Due to this setup, project progress was monitored through two ways. Weekly meetings were held between management and each project team to gauge their progress and address any concerns they might have. Second, project team members were required to keep time logs of their project-related activities.

Data and Analysis.

The six student teams were all given the same framework as outlined above. Over the course of the project life span, we collected a wide variety of data using a multitude of techniques. Below we list the data collected and the collection technique.

1. Project Marks – official grades
2. Self Assessment – questionnaires
3. Progress Reports – reviewed by instructor and teaching assistants
4. Problem Reports – received and dealt with by framework expert
5. Review Results – reviewed by instructor and teaching assistants
6. Acceptance test results – reviewed by another team and further assessed by instructor
7. Project Documentation – reviewed by instructor and teaching assistants

The students' questions for the CSF developer that came in over the term were answered using the FAQ and recorded for later study.

Because the development teams were composed of students, some guidance was provided in the development process. In addition, to monitoring project progress several deliverables were required at predefined milestones in the project lifecycle. After the final product was delivered, each team filled out a short survey to document their subjective experience of using the framework. In addition, course TAs and the framework expert undertook detailed analysis of the application code looking for the correct use of the framework in their products.

We also used qualitative analysis methods [35], supported with quantitative comparisons where possible. Qualitative analysis methods have been heavily used in software engineering for process elicitation [28] and auditing [6], and more generally, where human aspects are considered [34].

Although we collected a lot of data, the remainder of this paper will focus on the three research questions outlined in section 3. To answer the first question on how people learn and use frameworks, we kept track of the teams through meetings, daily logs, and project deliverables throughout the entire course of the study. We also queried them using a final survey regarding their approach to framework use and application development.

In order to assess the value of hooks in answer to the second question, the hooks for the framework were divided into two sets of roughly equal complexity. One set was made available to the development teams, while the other was not. We then looked at the time logs, final products and surveys to see if the students spent more time in understanding the parts of the framework for which the hooks were not available as opposed to the hooks that were available.

To answer the third question on the utility of reviews, two reviews were conducted; one during the project's analysis phase and the other after the design phase. The first review

focused on the rate and extent of learning how to understand and use a framework. The second review focused on the effectiveness of traditional objectives of reviews such as defect finding and progress reporting as they apply to framework use.

5 SUMMARY OF RESULTS.

In summarizing our results, we focus our initial analysis over the three research questions outlined above.

Question 1: ***Is there an effective way to understand and learn how to develop applications using frameworks?***

In using the framework, the developers encountered several difficulties that they had to overcome. These problems can be grouped into three broad categories: architectural understanding, interactions understanding and technical problems.

Architectural understanding is about learning the basic model of the framework and gaining a high level understanding how the key pieces fit together. It is the first key step in using a framework. From our experience, it is evident that understanding the framework architecture is key for designing and building a correct application using the framework. Teams that did not grasp the CSF architecture early on struggled to produce a correct design. These teams had many problems at the more detailed interaction level.

Architectural misunderstanding manifested itself in several ways. Some teams attempted to extend frozen parts of the framework (those parts not intended to be extended or modified). Some attempted to duplicate functionality already present within the framework. For example, one application required a message passing system with particular properties. They did not recognize that the framework provided these properties; in part, this came through resistance to the idea of using the framework itself. The team mirrored the one already present within CSF, and then faced the difficult task of reconciling the two similar implementations of a message-passing scheme into a working product. Most of these problems were caught during the design review.

Understanding the interaction between the framework and an application is the next crucial step in using the framework once a developer understands the basic model. Problems with interactions are localized to the framework services or interface. For example, one team had problems with sending synchronous messages across the network when their application was not initializing that particular part of the messaging system correctly.

Lastly, dealing with technical problems is a part of any software project, including those involving frameworks. Problems such as errors in the framework, hardware and software incompatibilities etc. are usually caught during application testing. Our experience showed that when using third party software, testing might be problematic. Users may look at the source code to trace the problems not just through their own application code, but through the framework code as well.

The need for tool support for the testing procedures was obvious. Our development teams were able to manage through direct communication with the framework developers; however, in a general setup, easy access to the framework developers might not be possible.

Hypothesis 1. *When developing applications using frameworks users go through the stages of investigation, integration and testing. Each of these stages can be enhanced through documentation and tool support.*

Unlike many of the other aspects of the study, the effect of using frameworks on the development process was relatively uniform across all teams. They first investigated the framework, to build an initial understanding of its architecture, and have an initial view of how it will integrate within their application. The key process enacted during this stage was the information swapping technical review. Almost all groups made use of this session. They reviewed the framework documentation, raised concerns and asked questions about the framework usability and its documentation. From our initial study we are convinced that further tools could aid in investigation by stepping users through the process of using the framework on an example or linking the different types of documentation.

During the integration stage, the process is characterized by round trips between hooks, examples and use cases. Studying and running examples, investigating the code of the framework itself and from time to time asking questions of the developer were the basic techniques used during this stage. The challenge remains to identify guidelines or best practices to support this stage. Required support needs to integrate hooks with the examples in an intuitive way, yet not limit the hook to the specifics of the example. Tools at this stage could help users properly integrate with the framework by suggesting where to extend the framework and by enforcing the correct integration details.

For the testing stage, the design review indicated some positive potential. Yet, the reviews were not that successful due to the amount of documentation that needed to be reviewed. We recognize the need for more testing support to both identify and correct errors made using the framework. Tools could help at this stage by actually automatically generating test cases based on the hooks the developers have used.

Hypothesis 2. *Consultation based development model is well suited for developing applications using frameworks.*

Depending on the choice of the team, not everyone had direct experience with using the framework. The number of people responsible for learning and using CSF, shown in Table 1, varied from one person to nearly the entire group.

Grades of the design document were the major indication for team success. The significance of this measure was cross-checked using records of our weekly meetings and the type and volume of questions addressed to the CSF expert after the first technical review. Successful teams delegated all framework-related tasks to one or two team members. These members invested time and effort understanding the framework, then served as framework experts during the analysis and design phases of the project. The alternative approach involved replicating framework expertise across the team with all team members investing the same time and effort trying to learn and understand the framework.

Team	# of members using CSF	Design Assessment
A	1	98.67%
B	2	92.00%
C	3	88.67%
D	4	55.33%
E	2	93.33%
F	3	83.33%

Table 1: Team Members Using the CSF

Learning the framework requires a lot of time and effort. Resources assigned to that job must be kept at minimal level. However, the team gains collectively from having an expert with good understanding of both the framework and the application. This expert was able to propose solutions and answer question when the need arose without having other developers to devote extra time to understanding the framework themselves.

When asked how confident they were in building another application using the CSF, all members involved with the CSF expressed a high level of confidence. This helps indicate that each CSF expert invested the required amount of effort to fully understand the framework. However, from the project perspective, teams using consultation based model made a better investment for their time.

Question 2: Does the use of hooks improve the understandability and usability of a framework?

In order to assess the usefulness of hook documentation, we divided the framework into a number of services (similar to the idea of framelets [30]) and then looked at the students perceived difficulty and actual errors made relating to each service. The services are:

- *Asynchronous communication* – sending messages across the network without blocking the sender.
- *Synchronous communication* – sending messages across the network and then waiting for an immediate reply within a given time limit.
- *Mail server* – choosing and using the correct 'engine' for message passing based on the type of application being produced.
- *Persistence* – saving and restoring objects to and from persistent storage.
- *Data proxies* – providing a local representation of data being stored on another machine.

Furthermore, we focused on the utility of hooks and examples. In order to do that, we provided varying levels of documentation for each service. Design diagrams and use cases for all services were provided. Table 2 shows which type of documentation was provided for each of the services.

Service	Hooks	Examples
Asynch.	Yes	Yes
Synch.	No	Limited
Mail server	Yes	Limited
Persistence	No	Yes
Proxy	No	No

Table 2: Documentation Provided for Framework Services

Limited refers to cases where the documentation covered some key aspects of use, but not others. *Yes* means that the documentation covered all key aspects of use, while *no* means no documentation of that type was provided.

In terms of complexity, we rated the services as simple, moderate and complex based on the type of hooks required in order to use this service. Proxy is a complex service requiring a substantial effort from developers. Mail server is a simple service, requiring only choosing the right component and initializing it correctly. Finally, Persistence, Synchronous and Asynchronous Communications are of moderate complexity. Using these services mostly requires the developers to supply application specific code in certain places.

Hypothesis 3. *Examples are perceived as the most important type of documentation in the integration stage.*

On a scale of one to five, with one being the lowest, students rated code as the most useful type of

documentation, with examples as a close second. The results are shown in Table 3. We believe that examples are more important than code because, as one student mentioned "... the examples are very useful and when we encounter a problem that the examples could not give answer, then I went to the code to see the details." However, they received lower score because "I did not find them as clear as they could have been" as stated by another student.

Documentation	Average Score	Median Score
Code	4	4
Hooks	2	1.5
Examples	3.67	3.5
Design	1.83	2
Use Cases	2.5	2.5

Table 3: Documentation Usefulness

Examples provided three advantages. First, they illustrated specific uses of the framework and documented instances of using the hooks. That is, concrete examples of many of the hooks were given in the sample programs. Second, the examples were executable and could be used to help understand the dynamics behind the system. Finally, the examples provided code that could be scavenged and used directly in the applications.

We associate the usefulness of examples with the integration stage in part, due to the subjects' memory effect. The usefulness was assessed at the end of the project when developers had been focussing on the code for some time, rather than at the beginning when they were still learning the framework.

Hypothesis 4. *Examples are not enough to understand how to use a framework. Hooks provide additional tangible benefit in understanding the interactions between an application and the framework.*

The examples only showed specific instances of framework usage, rather than the general cases described in the hook documentation. Some students had problems extending the functionality beyond what examples offered. Some groups attempted to apply the examples directly to their applications then had to go back and modify or re-implement them when the examples didn't exactly match their requirements. In our experience, developers looked at specific examples first and then tried to abstract the required knowledge.

The lesson we learned is the importance of tying the general case (the hooks) to the specific concrete cases (the examples) so that the developers can see the abstractions more clearly.

Hypothesis 5. *Complexity is not the deciding factor in the perceived difficulty of a framework service.*

We then examined the surveys provided to the students and the actual difficulties encountered. The services were ranked in the following order of difficulty to understand and use:

1. Synchronous Communication, and Proxies: difficult
2. Persistence and Mail Servers: moderate
3. Asynchronous Communication: easy

Synchronous communication proved to be clearly the most difficult to understand and use. In this case, the limited examples and lack of hooks hurt understanding, particularly when certain key aspects of communication differed between the cases illustrated in the examples and what the developers wanted, when no hooks were provided. Data proxies were also difficult to use, but since there was not even an example of their use, only one group attempted to use them, with limited success. The other three services were ranked much more closely and in general. While developers experienced some difficulties, they found them relatively easy to use.

Finally, we compared the complexity of the services against their ease of use. Complexity is based on the types and sizes of hooks that are required to use the service. The type of hook can be *option* (requiring the selection of an option), *template* (requiring the user to fill in specific data) or *open* (allowing the user to develop custom functionality). The complexity of the five services can be ranked as follows:

1. Proxies
2. Persistence, Synchronous Communication, Asynchronous Communication
3. Mail Servers

As can be seen, complexity did play a part in the ease of use, but it was not the deciding factor. We expected proxies to be the most difficult and mail servers to be the easiest; however, that did not prove to be the case. The coverage of the documentation proved to be the deciding factor in how difficult the services were to use regardless of how complex they were. Many groups struggled with synchronous communication as opposed to asynchronous communication. As seen in Table 2: Documentation Provided for Framework Services, asynchronous communication was better documented than its synchronous counterpart. Even though the two services share much of the same underlying mechanisms within the framework, the preliminary results indicate that the right type of documentation was very important. Persistence was easy to use when the application requirements matched the examples provided, but much more difficult when the application had to go beyond the example, and no

generalizations through hooks were provided.

Question 3: ***Can technical reviews be used effectively to speed up the early stages of framework learning process?***

Hypothesis 6. *Technical reviews with the purpose of information swapping speed up the learning curve for new framework users.*

Technical review held early in the term helped to raise the level of understanding of the framework's architectural model. First, it required users to review the documentation and second it allowed them to address immediately their questions and concerns to the framework expert. Students rated the usefulness of the technical review towards understanding the CSF as high. Furthermore, some students added "The technical review was the first step in development, and I thought it was an important first step in understanding the CSF."

Results of this review can be used to gauge a team's level of understanding of the framework concept. During the review, it was evident that one group was resisting the framework and trying to deploy another technology they are more familiar with. During the review session, motivation and benefits of using the framework were emphasized, thereby reducing the resistance.

Potential Confounding Factors.

Since we don't have the same level of control as in experimental study, identifying and eliminating the effect of potential confounding factors is vital for the correctness of the study findings. Ideally, the effect of potential confounding factors should be gauged using quantitative analysis. In similar studies, researchers sometimes used a high α-level [1] A statistical test at this significance level does not provide strong evidence of relationship [36]. The limited sample size (six teams) made it infeasible to use quantitative statistical tests, even with high α-level. The exploratory nature of the study allowed us to use qualitative techniques to assess the effect of potential confounding factors. These techniques are used frequently in social science research in similar situations [2], [39]. The effect of subjects' background is a usual concern in this type of study. Typically, industrial experience [39] and professional training of a software developer are used to assess his/her background.

Professional training was assessed in terms of academic records in previously studied computer science courses and the count of these courses. Based on their grades, they were categorized as above average, average or below average. Assuming a normal distribution for students' grades, a student is considered:

- above average if his/her GPA is greater than the class average GPA + $\sigma/2$, (31% of the class)
- below average if his/her GPA is less than the class average GPA – $\sigma/2$, (31% of the class)
- average otherwise. (38% of the class).

All subjects in our study have experiences in designing and implementing course projects, and they all worked in development teams. We also assessed their development experience in industrial setting. A student was assessed as:

- novice if she has less than one year of experience in industrial setup
- experienced if he has more than five years of experience in industrial setup, and
- limited experience otherwise.

Background data was assessed using a Likert-type scale in the following manner: above-average (3), average (2), below-average (1), experienced (3), limited experience (2) and novice (1).

Table 4, provides a summary of each team's background profile. The team score is considered as a percentage of the maximum score they could have achieved. For example, if the academic records of a team of six students shows 2 above average students, 3 average and one below average, the team score would be 55.56% (2*3+3*2 + 1*1 = 13 divided by 6*6).

Team	Industrial Experience	Professional Training	
		Score	Count
A	58.33%	66.67%	14.2
B	58.33%	72.22%	15.8
C	40.00%	55.56%	10.33
D	41.67%	53.33%	14.5
E	66.67%	77.78%	10
F	55.56%	73.33%	11.4

Table 4: Subjects Background Profiles

The results indicate that there is no apparent correlation between the score in professional training and quality of the design. The correlation seems clearer with the industrial experience.

6 STUDY FINDINGS.

In order to prepare for the next round of the study, we related our observations to the original research questions. Findings from this study were used as objective input for the next round of studies. First we discuss our answer to the research questions we had at the beginning of the study. Next we describe the changes we introduced for the next

round of studies based on our experience with this study.

Is there an effective way to understand and learn how to develop applications using frameworks?
Frameworks dictate the solution space and how the solution can be delivered for the problem addressed by applications constructed using a framework. The implication is that an additional process dedicated for framework understanding is necessary. In our environment, successful strategies have a uniform process pattern. The pattern consumed different resources and relied on different documentation depending on the project lifecycle. Roughly speaking, the pattern was divided into *investigation, integration* and *testing*.

The objective of the investigation stage is to gain architectural understanding of the framework. Not having the correct model of how the framework operates lead to greater numbers of integration problems. All project members participated equally at this stage. High-level framework documentation (use cases, high-level design diagrams) were the source of information at this stage. The analysis stage starts before project analysis.

The objective of the integration stage is to understand the interaction details between the framework and the application. Level of participation of project members varied depending on their role in the project. Documentation addressing low-level details of the framework was the main resource at this stage. Examples, hooks, and sometimes the framework code were heavily used. Development of expertise is the key success factor at this stage; few developers assigned to become framework experts was a successful strategy. When needed, these experts provided advise to other project members.

The objective of the testing stage is to ensure the proper integration of the framework in the application and that the application specific code is fully tested. The main emphases in this study was on the development of thorough acceptance test plans and the requirement that another team could install the product and properly execute the acceptance test plan. Test plans had to show that each major functional requirement was met if the acceptance tests successfully executed. The development of a testing tool that could assist the developer in creating test cases for each hook is still in the research stage and is not yet available for framework users.

Does the use of hooks improve the understandability and usability of a framework?
The results for this question were mixed; hooks alone are not enough to provide the required support for developing applications from a framework. While hooks proved useful in supporting the framework usability, they did not improve the needed architectural understanding of the framework.

Hooks proved useful in pointing developers to the places in the framework that they needed to extend in order to produce their applications. While reviewing the design, hooks help in pointing out defects in the interaction between the application and the framework. They were less useful in guiding the developers initially. Examples filled this role, though they failed when the problem didn't fit the example.

Can technical reviews be used effectively to speed the early stages of framework learning curve?
Technical reviews proved to be useful during the investigation stage of framework development in a variety of ways. The formal nature of the meeting set a milestone for the development teams.

Before the review meeting each developer had to read the documentation, reflect on their application and prepare their list of questions and concerns. In our case, the review session marked the end of the investigation stage and the start of the integration one.

While preparing for the review, each developer had a chance to reflect on his/her application and see where it may interface with the framework. During the meeting, developers had the chance of voicing their concerns and asking their questions in a structured way and to ensure that every point raised is addressed.

Results of the rework phase were used as shared experiences among development teams. The added FAQ provided a preliminary experience base around the CSF.

Next Round of Studies
Our plan is to continue the study for more semesters. For the next round of the study, we have planned to introduce some changes in the development process as well as framework documentation to reflect findings and further test the hypothesis developed in this study. We have added another section to the framework documentation that provides developers with guidelines on how to use the framework.

On the process side, reviews have been fine-tuned in the following manner:

- Reviews have been scheduled earlier to give students enough time to incorporate findings from the design review in their final product.
- For the design review, material to be reviewed has been altered to focus primarily to the application's interface with the framework and not the entire application design. This change allows us to study effects of misunderstanding CSF usage early in the design process.

Finally to improve the data monitoring, we plan to add a questionnaire after the first review to collect more information about the review process and to monitor how students view and understand the framework before and after the integration stage.

7 CONCLUSIONS AND FUTURE RESEARCH

As we emphasized earlier, the ultimate goal of this study is to develop and share a continually evolving and expanding experience base for development using large-grained software-component. Our study has revealed some lessons-learned and characterizations that apply to application development using small frameworks, involving teams of five to six developers. This study contributes to framework understanding and usability along two dimensions. First, the study procedures set an easy-to-replicate example about framework usability and understanding studies. Second, it established future objectives about development documentation and support tools for frameworks.

So far, we have found that no one form of documentation is sufficient to support the development. We see the need for a spectrum of documentation. This documentation should cover i) the purpose or domain of the framework, ii) limitations placed on applications that use it, iii) how it is intended to be used both at a concrete and more general level, and iv) both the design and implementation of the framework itself. We also found the need for a documented general process for using the framework that would provide greater guidance to the new user. Furthermore, different stages of development need different types of documentation. Finally, we identified three types of tools that are required to support the three stages of framework understanding (investigation, integration and testing).

To reduce the steep learning curve during analysis stage, experiences gained from framework users should be documented, organized and made available to other developers using the framework. Future users should then be able to browse and reason about available information to reach the best solution for the problems they encounter. Generally two types of experiences are needed: technical and process experiences. Technical experiences address particulars of the framework, such as what hooks should be used in what order. Process experiences, on the other hand, address the development process in general and hence should be easily generalized to development efforts using other frameworks.

During the development phase, application developers need to relate prescriptive-style documentation (as in hooks) to how and where changes to the framework typically occur during application development. Usually, a developer needs to trace through a set of supporting documentation, such as use cases and examples, to understand this relation. A tool that could automatically support such trace activities is of vital importance.

From our own results and from discussions with companies, we recognize that testing is a critical issue with framework use. During the testing stage, problems can be difficult to track down due to complex framework/application interactions, and users of the framework may not know all of the cases that the framework requires that they test for. We are working on generating test cases based on how a framework is used.

On the process side, we recommend technical reviews as best practice for building applications using a framework. Technical reviews proved useful in reducing the framework learning curve and for testing design issues related to the interface between the framework and applications.

We strongly believe that studies such as FrameScan are sorely needed to better understand the issue related to framework usability. At the same time, caution must be followed in interpreting and generalizing results of such studies. We encourage the research community to replicate our study using the same framework [14] in order to validate (or falsify) our hypotheses. Furthermore, similar studies using large frameworks and/or large team environment are needed to study the limitation of our findings.

REFERENCES

1. Basili, V.R. and Reiter, R. Jr. A controlled Experiment Quantitatively Comparing Software Development Approaches. *IEEE Transactions on Software Engineering.* 7(5), 1981, 299-320.
2. Bassey, M. *Case study research in educational settings,* Doing qualitative research in educational settings series. Open University Press,1999.
3. Baumer, D., Gryczan, G., Knoll, R., Lilienthal, C., Riehle, D. and Zullighoven H. Framework Development for Large Systems. *Comm. Of the ACM.* 40(10), 1997, 52-59.
4. Bosch, J., Software Product Lines: Organizational Alternatives. In *Proceedings of the 25th ICSE.*. 2001, 91-100.
5. Booch G. *Object Solutions: Managing the Object-Oriented Project* Addison-Wesley, Reading, MA, 1995.
6. Briand, L., Melo, W. and Seaman, C. A Qualitative Analysis Method for Auditing Software Maintenance processes and Organizations. *Software Process Newsletters.* No. 4, 1995, 3-5.
7. Brugali, D., Menga, G., and Aarsten, A. The framework Life Span. *Comm. Of the ACM,* 40(10), 1997, 65-68.
8. Butler, G., and Denommee, P. Documenting Frameworks. In *Building Application Frameworks*. Fayad, Schmidt and Johnson ed. Wiley Computer Publishing, New York. 1999, 495-503.
9. Codenie, W. DeHondt, K. Steyaert, P. and Vercammen, A. From Custom Applications to Domain Specific Frameworks. *Comm. Of the ACM, 40(10),*

1997, 71-77.

10. Fayed, M.A. and Schmidt, D.C. Object Oriented Application Frameworks. *Comm of the ACM, 40(10), 1997, 32-38.*
11. Freedman, D.P. and Weinberg G. *Handbook of Walkthroughs, Inspections and Technical Reviews.* 3rd Ed. Dorset House, NY, 1990.
12. Froehlich, G., Hoover H.J., Liu, L., and Sorenson, P. Hooking into Object-Oriented Application Frameworks. In *Proceedings of the 1997 International Conference on Software Engineering* (Boston, Mass, 1997), 491-501.
13. Froehlich, G., Hoover H.J., Liu, L., and Sorenson, P. Reusing Hooks. In *Building Application Frameworks*. Fayad, Schmidt and Johson ed. Wiley Computer Publishing, New York. 1999, 219-235.
14. Froehlich, G. "Client Server Framework (CSF}", http://www.cs.ualberta.ca/~garry/framework/
15. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
16. Gangopadhyay, D. and Mitra, S. 1995. Understanding Frameworks by Exploration of Exemplars. In *Proceedings of the 7th International Workshop on Computer Aided Software Engineering (CASE-95)* (Toronto, Canada, 1995), 90-99.
17. Hakala M., Hautamäki J., Koskimies K., Paakki J., Viljamaa A., Viljamaa J.: Annotating Reusable Software Architectures with Specialization Patterns. *In WICSA* 2001.
18. Johnson, R.E. and Foote, B. Designing Reusable Classes, *Journal of Object Oriented Programming.* 1(5), 1988, 22-35.
19. Johnson, R.E. Frameworks = Patterns + Components, *Communication of the ACM.* 40(10), 1997, 39-42
20. Johnson, R. Documenting Frameworks Using Patterns. In *Proceedings of OOPSLA'92* (Vancouver, Canada, 1992), 63-76.
21. Kolodner, J. *Case-Based Reasoning*, Morgan Kaufmann Publishers, CA, 1993.
22. Krasner, G.E., and Pope, S.T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3), 1988, 26-49.
23. Lajoie, R. and Keller, R. Design and Reuse in Object-Oriented Frameworks, Patterns, Contracts and Motifs in Concert. In *Proceedings of the 62 Congress of the Association Canadienne Francaise pour l'Advancement des Sciences.* (Montreal, Canada, 1994).
24. Lange, D.B. and Nakamura, Y. Interactive Visualization of Design Patterns Can Help in Framework Understanding. *In Proceedings of OOPSLA'95*. (Austin, TX, 1995), 342-357.
25. Leake, D.B. *Case-Based Reasoning: Experiences, lessons and future directions*, Chapter 2, The MIT Press, 1996.
26. Liu, L. *Hook Master: A Tool to Support the Enactment of Hooks*. MSc. Thesis, University of Alberta, 1999.
27. McIlory, M.D. Mass Product Software Components. In *Software Engineering* Report on A Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 1969.
28. Parra, A., Seaman, C.B., Basili, V.R., Kraft, S., Condon, S., Burke, S. and Yakimovich, D. The Package-Based Development Process in the Flight Dynamics Division. *Proc. Of the 22nd Software Engineering Workshop,* NASA/Goddard Space Flight Center, Software Engineering Laboratory (SEL), 1997, 21-56.
29. Pree, W. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley Publishing Company, Reading, MA, 1995.
30. Pree W., Koskimies K.: Framelets - Small is Beautiful. In: *Building Application Frameworks: Object-Oriented Foundations of Framework Design* (M.E. Fayad, D.C. Schmidt, R.E. Johnson, ed.), Wiley 1999, 411-414.
31. Schmid, H.A. Creating Appplications from Components: A Manufacturing Framework Design. *IEEE Software,* 13(6), 1996, 67-75.
32. Schmidt, D.C. Applying Patterns and Frameworks to Develop Object Oriented Communication Software", *Handbook of Programming Languages,* MacMillan Computer Publishing, 1997.
33. Schneider K. and Repenning, A. Deceived by Ease of Use: Using Paradigmatic Applications to Build Visual design Environments. *Proceedings. Of the Symposium on. Designing Interactive Systems.* 1995.
34. Seaman, C.B. and Basili V.R., Communication and Organization: An Empirical Study of Discussion in Inspection Meetings. *IEEE Trans. On Soft. Eng.* 24(7), 1998, 559-572.
35. Seaman, C. B. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. On Soft. Eng.*25(4), 1999, 557-572.
36. Shull, F., Lanubile F., and Basili, V.R. Investigating Reading Techniques for Object-Oriented Framework

Learning. *IEEE Trans. On Software Engineering*. 26(11) 2000, 1101-1118

37. Sparks, S., Benner, K. and Faris, C. Managing Object-Oriented Framework Reuse. *IEEE Computer*. 29(9) 1996,52-62.

38. Taligent, Inc. *The Power of Frameworks.* Addison-Wesley, New York, 1995.

39. Yin, R.K. *Case Study research: design and methods.* Applied Social Research Methods Series, Thousand Oaks: Sage Publications. 2nd edition, 1994.