Patterns: A Training Experience

by

Brandon Goldfedder & Linda Rising

Overview

A group of respected individuals in the object-oriented community has been taking a hard look at the work of Christopher Alexander, a building architect. Alexander has some intriguing observations about an entity he calls a **pattern**, defined as follows [1]:

Each pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice.

Software design patterns have been defined by other authors, including Jim Coplien and Doug Schmidt [2]:

Design patterns capture the static and dynamic structures of solutions that occur repeatedly when producing applications in a particular context.

Since members of the software engineering community have been searching, unsuccessfully for the most part, for an answer to the problem of re-inventing the wheel, the patterns investigation has been attracting considerable attention. Patterns were introduced to the object-oriented community at large at OOPSLA '94, where a tutorial, ***Design Patterns: Elements of Reusable Architectures,*** was given and the Design Patterns text [3] first became available. This paper describes the initial steps to introduce patterns through training by the Dalmatian Group to AG Communication Systems. This training focuses on the patterns in the Design Patterns text. Since this early period, training in patterns at AG Communication Systems and elsewhere has expanded to include patterns introduced at PLoP[2,3] and "mined" from legacy systems.

Initial Efforts

The patterns initiative at AG Communication Systems began in April 1995 with an evaluation phase. Members of the evaluation team included functional area representatives, resident object-oriented gurus, and others who expressed an interest in understanding patterns. There were about twenty team members in all.

Copies of the Design Patterns text were ordered for the team and others. This was an important part of the process. We found that when the book is in the hands of designers, the patterns will begin to take hold. This is a big selling point for patterns - there is a book on patterns and designers will read it.

The evaluation meetings were open to anyone who was interested in learning about patterns. That provided a larger audience for the information and more input for the discussions. The evaluation phase was easy. The patterns sell themselves. The team had no trouble answering the questions:

1. Is the notion of patterns a worthwhile technology for AG Communication Systems? The consensus was an overwhelming yes.
2. Are the patterns in the Design Patterns text worth a training investment? Again, the consensus was an overwhelming yes.

At the end of the evaluation phase, the following recommendations were made:

1. Every interested designer should have a copy of the Design Patterns text.
2. Training in these patterns should be provided for teams to enhance communication and design knowledge. Some members of the team expressed their concern with the lack of real-time, embedded system considerations in the Design Patterns text. This was addressed in the training through the use of additional examples and implementation details lacking in the Design Patterns text.

During this first stage, patterns were already appearing on some projects.

Beta Training

At this point, we began working with The Dalmatian Group. One of their trainers, Brandon Goldfedder, delivered a three-day beta patterns course for the evaluation team in June 1995. After the course was presented, we considered feedback from the evaluation team members. Suggested improvements included: topic reordering, improvement of examples, additional motivational sections. It was also suggested that we reduce the amount of time spent reviewing object-oriented concepts to make room for more lab exercises.

An area of some concern was the challenge for most students to absorb the material in just three days. Ralph Johnson has observed that [4] "... people can't learn patterns without trying them out. Also, people need to find them in their own problem domain." This observation was confirmed by our experience with the beta course, even though the members of that group had been through most of the patterns in the evaluation phase.

During the evenings of the beta training, the trainer met with some of the students to suggest ways to apply patterns in their current projects. These students found this assistance valuable. As a result, we decided to incorporate consulting opportunities along with the training and change the course schedule from three full days to one full day followed by four half-days of training. This allowed four half-days of consulting. The opportunity for hands-on mentoring and consulting became an essential element for the success of patterns introduction.

We tried to target a specific team for each course offering to provide the biggest leap in abstraction and communication. Having a team go through training together also meant that examples meaningful for the team could be discussed and used in the consulting sessions. One team member who experienced the benefits of this mode of training wrote a pattern to describe the experience [6].

Course Structure

The course focused on applying the design patterns in the Design Patterns text. Emphasis was placed on change management, specifically long-term maintainability and extensibility. In addition, integrating patterns into the development lifecycle through design reviews and documentation was a key focus. The class provided both an overview of important concepts as well as detailed discussion of the use of the patterns discussed in the Design Patterns text. The format comprised lecture, discussion, and lab exercises. Labs and discussion material were designed to be extremely modular to allow reuse and modification for special versions of the course. This flexibility was necessary because of the diversity in both language and design skills of the students. Later incarnations of the course included an overview of patterns and advanced review sessions for each of the three categories of patterns: Creational, Structural, Behavioral, defined in the Design Patterns text. These presentations were built by reusing existing course modules and modified as needed. New examples and additional slides were added for groups with special interests and backgrounds.

At the end of each class offering, student and instructor critiques were reviewed and updates made to session plans, instructor notes, course slides, or supplemental slides. This ensured that the experiences of all instructors of this class could be shared and the course improved in a controlled manner. The course went through several profound changes that helped to improve the class. The discovery of the best ways to teach certain patterns occurred naturally under this process.

Course Implementation

Lab exercises are essential to giving students the "Aha" experience. After each category of patterns was covered with lecture and discussion, groups of three or four students were given a set of requirements and asked to present a design solution to the rest of the class for review. The class presentation and peer review addressed long-term maintainability, extensibility, and flexibility of the solutions. Exercises were done with pencil and paper or CASE tools, depending on the students' work environment.

Covering all the patterns in a short period of time usually resulted in all the patterns blurring together or students spending more time understanding the details of many different examples than on the patterns themselves. To address this difficulty, a single example, an Alarm Monitoring System , was used for discussion and the lab exercises. The students could consider each exercise in the context of this running example without having to learn the details of a new problem setting. This also avoided considering any pattern as only useful for a single example and allowed the focus of the class to remain on the intent of each pattern.

We felt it was important to cover all the patterns in the book and to show how they complement each other. When students begin to use patterns and understand that any pattern is just one piece in the system, they begin to see design at a much higher level.

Some patterns were introduced through the discovery process. For example, students were given a traditional solution to the problem of designing an alarm system and asked which areas could not be modified without risking major upheaval to the system. We restructured the architecture to exploit this understanding. As each new requirement for change was presented, additional patterns were introduced as well as additional examples common to the students' domain. As a result, students had a better, more complete understanding of the pattern and its appropriate use.

It was difficult to determine the amount of code to present in the examples. It is important for students to realize that patterns are applied in the design phase and not to be overly concerned with specific implementation issues. Without seeing some code, however, the discussion can become too academic. The challenge was to find an appropriate level of detail for the students to understand the implementation. We have observed that as the course progresses and in follow-on consulting sessions, the need to see code to understand the pattern decreases and the focus shifts to design issues.

There is a certification process that seemed to be present. Some developers could not accept design guidance from someone without the details of C++ syntax and the behavior of compilers. Although design patterns are at a much higher level than code, students needed convincing that the code produced will be at least as good as their existing approaches. The exact mechanism used varied from class to class. In some classes, a quick code example sufficed. In other classes, a more detailed discussion of the code generation of the C++ compiler was required, often with explanations of virtual table costs. This was especially true in introducing the state pattern. Often students solved the problem addressed by this pattern with complex conditional code or an complicated lookup table. The state pattern is conceptually simpler and implementation easier.

Brief discussions of implementation issues were conducted during class time, as needed. Longer, more specialized discussions were postponed to breaks and follow-on consulting sessions. It is essential that the instructor be able to address these concerns. Once past this hurdle, developers became open to high level concepts.

Observations

In contrast to our experiences teaching other software engineering classes, there was little resistance to these new concepts. In many design classes considerable time is spent selling the concepts, but it is extremely difficult to show, in a convincing manner, implementation improvements to someone who is not open to new ideas or has become accustomed to solving problems one way. In this course, it is easy to provide justification for improvements to efficiency, maintainability, modifiability, and extensibility of the code as well as show significant reduction of time spent in documentation through the use of patterns. The exact mechanism for showing these benefits varied significantly. In some classes it was sufficient to show the C++ implementation, in others it was necessary to discuss compiler output. By addressing these issues up front, rather than hand-waving them as insignificant, students could not easily dismiss this technology as not offering performance improvement.

When looking at students' ability to successfully apply this technology, the amount of C++ and object-oriented experience or years of experience in a single project were not significant. The exposure to a variety of systems was extremely important. There is an interesting lack of correlation between object-oriented project experience and acceptance of and ability to apply patterns. Some of the more experienced developers have been among the least able to understand what patterns are all about, while some of the developers from non-object-oriented projects grasp the concepts readily. We believe that a lot of experience in only one language and/or project leads to a narrow focus and a belief in students' minds that they have a greater breadth of knowledge than may actually be present. Regardless of the language, methodology, or systems students have used, those who are exposed to more ideas are kept in a learning mode and remain more open to new ideas. Additionally, they have far more diverse experiences to call upon. We have further found that this broadness provides benefit for system architects and good designers.

Frequently, students were unable to attend all the sessions in a week of training. These students were accommodated by offering them a chance to make-up sessions in later courses. We observed that many would elect to take several days over again, in addition to the ones they had missed. Going through the material again, helped those who were struggling to sort out pattern details. Our response was to offer this option to anyone, whether they had missed scheduled training or not.

The use of patterns promotes a common vocabulary that allows a team to focus on problems rather than the details of implementation. It is exciting to see the change in character in reviews as higher levels of design can quickly be communicated. This also provides an interesting side effect: it is now possible to recognize and characterize the similar problems that projects share. We are in the process of documenting solutions to these common problems as patterns to allow cross-project knowledge transfer and reuse.

Ongoing Efforts

We recently introduced a one day overview of patterns to reach those who did not have the time to attend the full course. Most of the attendees felt that they could not afford NOT to attend the full course and signed up for the next full class.

Training of the object-oriented teams, over 100 people, was completed at the end of August 1995. This training period was followed by a few visits from the trainer to provide review sessions on each pattern category and to offer additional consulting opportunities. It was our experience that training alone, without the consulting and hands-on mentoring, would not have been as effective.

AG Communication Systems has the problem of any organization, that new hires come in and others leave the company. Now that our developers have been trained by the Dalmatian Group, we need to introduce new employees to the patterns in the Design Patterns text and others in our growing collection. An internal training organization has taken over the task of "spreading the word" about patterns. This group offers courses in the patterns, using a project architecture as a running example, presents a users course, a writers course, and a brainstorming workshop for experts to mine patterns in a given area. The AG Communication Systems patterns model is one of capturing patterns from experts and creating courses to share these patterns with the rest of the company. In examining the behavior of other engineering disciplines, we find that students trained in the contents of an engineering handbook don't suddenly begin to leaf through this handbook whenever a new problem is presented. Instead, they simply use the handbook to reference details of a solution they already know. Therefore, training is an essential part of the use of patterns in any environment.

We feel the time is right for patterns. Patterns are not the long sought-after silver bullet but they provide benefit with minimal investment. There's an interesting side-effect in the patterns activities. Engineers at AG Communication Systems feel they're really on the cutting edge and they are! There's a sudden interest in reading the literature, buying books, attending conferences and classes, including the Pattern Language of Program Design (PLoP) conference. Several of us attended OOPSLA '95 to get all the new patterns information we could.

When asked about productivity benefits from patterns, we are quick to point out that people who are excited about their work and feel a part of something stimulating and challenging will be more productive, regardless of the technology that's involved. It's gratifying to be part of something that brings this kind of benefit to other software engineers. The issue of metrics to show this productivity are under consideration. In the mean time, we rely heavily on subjective reports such as [5].

Acknowledgments

The authors would like to thank the reviewers of this paper for their helpful comments and suggestions that added greatly to the quality of the final version.

References

1. Alexander, Christopher, et al. A Pattern Language, Oxford University Press, New York, 1977.
2. Coplien, James O. and Douglas C. Schmidt, ed. Pattern Languages of Program Design, Addison-Wesley, 1995.
3. Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.
4. Vlissides, John, Norm Kerth, James Coplien eds., Pattern Languages of Program Design - 2, Addison-Wesley, 1996.
5. Johnson, Ralph E., Electronic mail message.
6. Olson, Don, "TrainHardFightEasy," http://c2.com/cgi/wiki?/TrainHardFightEasy.
7. Duell, Michael, "Experience in Applying Design Patterns to Decouple Object Interactions On the Ingage ™ Platform," presented at OOPSLA '96.

"Patterns: A Training Experience," Communications of the ACM, Vol. 39, No. 10, October 1996, pp. 60-64.