

Pedagogical Patterns: Capturing Best Practices in Teaching Object Technology

Abstract. The Pedagogical Patterns Project began at OOPSLA '95 and started holding workshops in 1996 at the ECOOP, TOOLS and OOPSLA conferences. The aim of the project has been to capture successful experiences in teaching and learning OT, from industry or academia, in a homogeneous, easily-accessible format. Patterns have been regarded as an effective method for achieving this. Since 1996, many other sessions have been held at numerous conferences in America and Europe, resulting in the collection of more than 50 teaching techniques written in pattern format. As this present collection is now being refined and expanded, educators are discovering the effectiveness of sharing their teaching experiences in the form of pedagogical patterns.

Why Pedagogical Patterns are needed

Most educators and trainers are not taught how to teach. Rather, they often find themselves teaching by accident. Typically, a person with a skill that is in demand, such as a particular programming language, will be asked to teach it. People assume that if the person is good in this programming language, she will be good at teaching it. But knowing the subject matter is very different than knowing how to teach it.

Effectively communicating complex technologies is often a struggle for information technology instructors. They may try various teaching strategies, but this trial and error process can be time-consuming and fraught with error. Advice is often sought from other “expert” instructors, but these individuals are not always readily available. This creates the need to find other ways to facilitate the sharing of teaching techniques between expert and novice teachers.

This is the goal of the Pedagogical Patterns Project (PPP). Pedagogy is a term that refers to the “systematized learning or instruction concerning principles and methods of teaching” [Web59]. Patterns provide a method for capturing and communicating knowledge such as pedagogy. As an example, imagine that you are looking for an effective way to teach message passing to experienced programmers in a week-long industry course. A friend who is teaching a semester-long object technology course to traditional age university students has found an effective technique. He shares it with you without dictating the specific implementation details. This allows you to use your own creativity to implement the technique in a way that is most comfortable for you and most useful for your industry students. This is the essence of patterns – to offer a format and a process for sharing successful practices in a way that allows each practice to be used by a variety of people in many different ways.

A collection of patterns could form a repository of techniques for teaching a specific subject such as object technology (OT). Ideally, many of the patterns would have even

broader scope than OT, but all of them would be useful in many different training environments because they are proven teaching techniques.

But even this is not the end of the story. Related patterns can be combined in either a pattern catalog [Bus96] or in a system of patterns [Fow97]. A third possibility is to relate several patterns within a common problem space, the result of which is a language of patterns that provides a resource for solving complex problems. The goal of the project described in this paper is to form pedagogical pattern languages for teaching object technology. This will provide OT instructors with the ability to share their effective teaching techniques in a common format, to document relationships between the techniques and to form powerful tools known as pattern languages.

Patterns

Patterns capture expert knowledge in an accessible and usable way. The first set of patterns appeared as part of a book titled “A Pattern Language – Towns, Buildings, Constructions,” published in 1977 by the architect Christopher Alexander et. al. He defines a pattern as follows:

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. [Ale77]

This definition shows that a pattern is not a recipe or a kind of step-by-step documentation. Rather, it is a solution represented in an abstract, rather than a concrete way, with the context of the problem, the forces that create the problem and the reasons for and consequences of using the solution.

A pattern is not a simple trick, a magic technique, or a representation of just one person’s idea. To become a pattern, an idea must go through an abstraction and validation process so that the essence is factored out into a form that is both proven and generally applicable. Additionally, its applicability is verified by documenting three known uses of it. This allows patterns to be applied with confidence by different people in different situations.

Each pattern is simple and focused on a specific problem. However, the real power of a pattern becomes apparent when it is embedded in a *language* of related patterns. Combining patterns into a language allows them to work as a system. This is explained by Appleton:

... pattern languages provide a dynamic process for the orderly resolution of problems within their domain which indirectly leads to the resolution of much broader problems. [App98]

The effort to create pattern languages is not a trivial one and usually requires the work of many minds. Many people have attempted to do this since patterns were first

introduced to the software development industry. In 1995, the first collection of software patterns illustrated the value in capturing simple and elegant solutions to common problems in designing object-oriented systems [Gam95]. It sparked a rapidly-growing interest in creating many other types of patterns that capture best practices in a variety of system development tasks such as testing [DeL98], creating system architectures [Bus96], [Sch00], building software development organizations [Cop95], analyzing system requirements [Fow97] and managing the development process [Coc98]. As explained in a recent publication of the patterns community:

Software patterns document general solutions to recurring problems in all software-related spheres, from the technology itself, to the organizations that develop and distribute it, to the people that use it [Har00].

Patterns for Pedagogy

The Pedagogical Patterns Project is an effort to use patterns to capture best practices in teaching object technology [Man98], [Sha98]. Other projects are also attempting to record experiences in computer science education in other ways. For example, The On-line Computer Science Teaching Centre contains a refereed repository of resources to support teaching and learning in computer science, such as Java applets, tools, demonstrations, and lab exercises [Onl]. Another effort, EPCOS, uses a notion of a 'bundle' to capture computer science teaching techniques [EPC]. This has led to ask what benefits are gained from capturing experience as patterns rather than as guidelines, templates or simple stories. This question was posed in the *Do we need patterns for pedagogy?* workshop at the OT'2000 conference [Sha00-1]. Participants commented that 'bundles' appear to contain a set of techniques one can pick up and use; they are brief and terse and are not linked to other bundles. In contrast, patterns are linked together and appear to supply the principle behind the technique. It was suggested that a bundle could serve as a concrete example for a pattern. Therefore, while these two entities, patterns and bundles, appear to record different types of information, it was suggested that they may be able to work together to effectively capture pedagogy.

The Pedagogical Patterns Project is an international one with individuals from more than twelve different countries contributing successful practices for pedagogical concerns such as planning and conducting a short course [Eck00], teaching various techniques such as abstraction [Lim96], helping learners work in teams [Mey98], conducting peer review sessions [Abr97], developing a course in computer science [Ber00], and increasing the efficiency of class lectures [Man01].

The project began in the OT community because it was recognized that the complexity of OO makes this topic difficult to teach. Today, people interested in pedagogical patterns still belong primarily to the OT community, but there are efforts to widen the project to other teachers, educators and trainers [Ber01]. For example, one of the most mature pedagogical pattern languages, *Seminars*, captures pedagogy for teaching seminars on topics that expand far beyond OT [Voe99].

However, most of the contributions, forming a collection of more than fifty pattern drafts, have been contributed by those who teach OT [PPP]. They come from a wide selection of people with varying circumstances and backgrounds, and they vary

considerably in focus, from curriculum design to techniques for teaching or learning a specific concept. Over the years in which these patterns have been collected, it has become clear that the initial scope was too wide to reap the full benefits that a pattern language could potentially offer. Therefore, some of the patterns are now being “refactored” in order to produce multiple languages, such as a one for experiential learning [Mar01].

To further explain the format of a pattern, one of the refactored patterns from the evolving language for experiential learning follows. The pattern itself is presented in boldface type, while the standard weight type contains comments. References to other patterns reveal the beginnings of the relationships that will eventually form the language.

Example Pattern: Student Design Sprint

A pattern is given a short, memorable name to allow the technique to be easily communicated between individuals.

Student Design Sprint

Each Pattern starts with the description of a problem.

Students need to practice design at all levels. They also need quick feedback and peer review on early attempts. Most educators recognize now that students need to be exposed to design early. Most also recognize the need for team work and for critical analysis. We eventually need to teach system design, but beginners need program design as well. If we don't teach it then students will develop their own ad-hoc techniques that may reinforce bad habits. If you use a Spiral [JB2] approach the elements of simple design should come in the first cycle.

The solution is presented, often starting with the word “therefore”.

Therefore, use some variation of the following highly structured activity. This activity can take place in a seminar, classroom, or in a lab.

Divide the students into groups of two (or three). Give them a design problem and ask the teams to produce a design outline in 15-20 minutes. There should be a written sketch of the design in that time, perhaps with CRC cards if it is an object design. The instructor can look over shoulders and comment or not, but few hints should be given. Questions should be answered freely.

At the end of 15-20 minutes, the instructor poses a set of questions about the designs without asking for answers. The questions should be such that they cannot be favorably answered by some set of poor designs.

The students are then regrouped by combining pairs of nearby groups, so that you now have groups of 4 or five students and each group has two of the original designs. The task is now modified slightly and the groups are asked to produce a new design.

After another 15-20 minutes the instructor again poses a set of questions for thought, regroups the students again into still larger groups, modifies the task slightly and again puts the students to work.

This can continue for as many cycles as the instructor wishes. At the end, the instructor should evaluate the resulting designs and make comments. It may be enough to show one or two of the best designs and explain why these are better than the others. If poor designs are also to be shown, it might be best if the names of the designers are not attached.

Alternatively, the groups can be required to present and justify their designs and the rest of the class can critique them.

For some situations one cycle may be all that is needed, followed by a discussion of the issues. In this case the instructor can ask the groups which designs had certain characteristics.

To validate that a pattern (rather than an idea of only the author) has been captured, the known uses are summarized. Since this pattern is in early draft form, uses by those other than the pattern author are not yet known. It is during the review process that the known uses will surface and be documented.

Alistair Cockburn [AC] has a wonderful exercise for students designing a coffee machine in about three or four cycles in which the requirements become more sophisticated each cycle. In the first cycle the machine can deliver coffee for 35 cents. In the second it can also deliver soup for 25 cents.

This can be used in program design in the early phases of a student's learning. The task can be to write a function with a given set of pre and post conditions. The tasks in the later cycles can be to tighten the pre conditions and/or strengthen the post conditions.

Alternatively, the task could be to develop some code with a given invariant and the questions can involve ways that the invariant might be invalidated by a user if the design is not sound.

This pattern can be used when learning data structure design. For example, the students can be asked to design a linked list, without telling them how it will be used. They must design a protocol and pick an implementation strategy. The instructor can then suggest some uses to which a linked list might be put and ask if the design supports that use.

[AC] Alistar Cockburn, *Website*, <http://members.aol.com/acockburn/>

[JB2] Joseph Bergin, *Spiral*, <http://www-lifia.info.unlp.edu.ar/ppp/pp32.htm>

The life of a pattern

As previously stated, a pattern does not capture the idea of only one person. While it may be authored by an individual, it is refined and validated by a larger community. In the PPP, this community has been composed of OT educators and trainers.

A pedagogical pattern usually starts with an instructor who has found an effective way to teach a specific concept. She writes a pattern by abstracting the problem, the context, the forces that act upon the problem, the solution and the rationale for and consequences of using that solution [Mes98]. This author then offers it for review by sending it to colleagues and/or posting it on the web.

In an attempt to help OT educators write patterns from their successful teaching experiences, the PPP has also held several sessions at conferences such as TOOLS, ECOOP and OOPSLA. These sessions, known as *pattern mining*, have used a variety of techniques for allowing educators to “discover” the patterns within them. For example, at the TOOLS USA '96 conference, participants in the "Challenges and Successes in Teaching OT" workshops created some drafts for pedagogical patterns by first generating a series of challenges, then brainstorming on the solutions to many of these challenges, and finally formatting a rough draft of pedagogical patterns to document these challenges and solutions.

The following is an example from this pattern mining session:

One of the challenges is expressed:

Students often get OO concepts confused with the procedural concepts they already know. As a result, there is often more interference than reinforcing.

Potential solutions are brainstormed:

- The instructor can make comparisons for students.
- Or, the instructor can attempt to make students think in an object way from the beginning by postponing answers to students' inquiries regarding how their present knowledge (e.g. procedure-oriented) compares to OO.
- Or, the instructor may wish to lead students through the design of a system in two ways, first using a "traditional" way and then transitioning OO.

A pedagogical pattern is drafted from one of the three potential solutions. The third solution is formatted into the first draft of a pattern, as follows:

- Students design and implement a project using a traditional approach.
- Instructor lectures on object concepts and suggests how this project could be repartitioned into objects.
- Project is redone, by the students, using objects.

The final step is to describe this pattern using the pedagogical pattern format.

As soon as the pattern has been described in a generally applicable form either in a pattern mining workshop or by an instructor who has abstracted her effective teaching technique, it goes eventually further through the validation process. Ideally, the author chooses to submit it to a patterns conference¹ where, in the course of submission, an experienced pattern author (known as a *shepherd*) is assigned to help her improve and further refine the pattern [Har99]. It is then discussed during the conference in a rather structured, but supportive session known as a *writers' workshop* [Cop00]. This is a valuable opportunity for pattern authors to receive feedback that can be used to further improve their patterns.

Conclusions

Good teachers are often experienced teachers. One can learn to become a good teacher by studying the experiences of others. Because of the complexity in teaching OT, instructors have found that it is necessary to share successful practices. The PPP is showing that patterns are an effective way to do this. The patterns approach offers a format for capturing knowledge and a process for validating and making it part of a community. This international effort has been collecting individual patterns for more than five years and is now in the process of building pattern languages. It is expected that this will allow educators to solve even more complex problems in teaching OT and other topics in information technology.

To make this a successful attempt, we are looking for people who are interested in contributing to the project in different ways [PPP]:

Testpilot: The power of patterns comes from their solid connection to practical experience. A technique cannot be called a pattern if it has not been tried successfully, and is not continuously practiced. The job of a pattern testpilot is to try the patterns and give feedback to the authors.

Reviewer: The more people who see and read a pattern, the more opportunities there are for improving it. The job of a reviewer is to read patterns and give the authors feedback on content, language and form. As previously mentioned, when a reviewer works with an individual pattern author on the author's pattern(s), the reviewer is known as a shepherd.

¹ Many patterns conferences are held throughout the world each year. For more information, see <http://www.hillside.net/patterns/conferences/>

Author: There are countless effective teaching techniques that have the potential to be patterns. If you are an educator in industry or academia, we encourage you to share your experiences by becoming an author in the pedagogical pattern project.

Advertiser: It is important to spread the word. The more people who know about this project, the more the material can improve.

References

[Abr97] Abreu, Fernando Brito e, *Peer Review and Corrective Maintenance (PRCM) Pattern*, <http://www-lifia.info.unlp.edu.ar/ppp/pp14.htm>

[Ale77] Alexander, Christopher et.al. (1977). *A Pattern Language: Towns - Buildings - Construction*. Oxford University Press.

[App98] Appleton, B. *Patterns and Software: Essential Concepts and Terminology*.
<http://www.enteract.com/~bradapp/docs/patterns-intro.html>

[Ber00] Bergin, Joseph. Course Pattern Language (work in progress). *Website*.
[csis.pace.edu/~bergin/patterns/coursepatternlanguage.html](http://www.cs.pace.edu/~bergin/patterns/coursepatternlanguage.html)

[Ber01] Bergin, Joseph. Pedagogical Patterns and Pattern Languages for the Early CS Courses. Working Group Submission to ITISCE 2001, a SIGCSE /SIGCUE conference on Innovation and technology in computer science education.
<http://www.cs.ukc.ac.uk/events/iticse2001/>

[Bus96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, England: John Wiley & Sons.

[Sch00] Schmidt, D., Buschmann, F., Stal, M., Rohnert, H. (2000). *Pattern-Oriented Software Architecture: Patterns for concurrent and networked objects*. Chichester, England: John Wiley & Sons.

[Coc98] Cockburn, A. (1998). *Surviving Object-Oriented Projects: A Manager's Guide*. Reading, MA: Addison-Wesley.

[Cop95] Coplien, J.O. (1995). *A Generative Development-Process Pattern Language*. In: Coplien, J.O., Schmidt, D.C. (eds). Pattern Languages of Program Design. Reading, MA: Addison-Wesley Longman, Inc.

[Cop00] Coplien, J.O. (2000) *A Pattern Language for Writers' Workshops*. In Harrison, Foote, Rohnert (eds), Pattern Languages of Program Design 4. Reading, MA: Addison-Wesley.

[DeL98] DeLano, D., Rising, L. (1998). *Patterns for System Testing*, In R.W. Martin, D. Riehle, F. Buschmann (eds.), Pattern Languages of Program Design 3. Reading, MA: Addison Wesley.

[EPC] Effective Projectwork in Computer Science
<http://www.cs.ukc.ac.uk/national/EPCOS/>

- [Eck00] Eckstein, Jutta (2000). Learning to Teach and Learning to Learn. Running a Course. In Conference Proceedings of EuroPLoP 2000. Kloster Irsee, Germany.
- [Fow97] Fowler, Martin (1997). Analysis Patterns. Reusable Object Models. Reading, MA: Addison-Wesley Longman, Inc.
- [Gam95] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley Longman, Inc.
- [Har99] <http://jerry.cs.uiuc.edu/~plop/plop2k/shepherding.pdf>
- [Har00] Harrison, N., Foote, B, Rohnert, H. (2000). Pattern Language of Program Design 4. Reading, MA: Addison Wesley Longman, Inc.
- [Lim96] Lim, B.L. *Programming in the Tiny, Small, Large (TSL)*, <http://www-lifia.info.unlp.edu.ar/ppp/pp10.htm>
- [Man98] Manns, Mary Lynn, Helen Sharp, Phil McLaughlin, Maximo Prieto, (1998) Capturing successful practices in OT education and training, *Journal of Object-Oriented Programming*, Vol 11, No 1, March/April.
- [Man01] Manns, Mary Lynn. *See Before Hear*. <http://www.cs.unca.edu/~manns>
- [Mar01] Marquardt, Klaus, Markus Voelter, et.al. (to be published July 2001). Patterns for Experiential Learning. In Conference Proceedings of EuroPLoP 2001. Kloster Irsee, Germany.
- [Mey98] Meyer, Jeanine. *Team Teaching Pattern*, <http://www-lifia.info.unlp.edu.ar/ppp/pp40.htm>.
- [Mes98] Meszaros, G., Doble, J. (1998) *A Pattern Language for Pattern Writing*, In Martin, Riehle, Buschmann (eds), Pattern Languages of Program Design 3. Reading, MA: Addison-Wesley.
- [Onl] The On-line Computer Science Teaching Centre In Conference Proceedings 6th Annual Conference of the Teaching of Computing and 3rd Annual Conference on Integrating Technology into Computer Science Education, Dublin, <http://ei.cs.vt.edu/~cstc/>
- [PPP] Pedagogical Patterns, Updated, *Website*, <http://www.pedagogicalpatterns.org>
- [Sha98] Sharp, Helen, Mary Lynn Manns, Phil McLaughlin, Maximo Prieto, (1998) 'Patterns for Proficiency', *Applications Development Advisor* Vol 1, No 4, March/April.
- [Sha00-1] Sharp, Helen, Jutta Eckstein, Mary Lynn Manns (2000). Workshop: Do we need patterns for pedagogy? In Conference Proceedings of OT 2000, Oxford, UK.
- [Sha00-2] Sharp, Helen, Mary Lynn Manns, Jutta Eckstein. The Pedagogical Patterns Project. In: OOPSLA 2000 Addendum.
- [Voe99] Voelter, Markus, Astrid Fricke. SEMINARS, - A Pedagogical Pattern Language about teaching seminars effectively, Proceedings of EuroPLOP 1999, or at <http://www.voelter.de/seminars>
- [Web59] Webster's New Collegiate Dictionary. (1959). G & C Merriam Co.

