Originally presented at “Pedagogical Patterns: Successes in Teaching Object Technology”
OOPSLA'96 Workshop, October 6, 1996, San Jose, California, USA

Pedagogical Patterns: Picking the Metaphor from the OO Design Community

Fernando Brito e Abreu
INESC and Lisbon Technical University (ISEG), Portugal
(email: fba@inesc.pt)

Abstract
Pedagogical Patterns are a recent proposal for formalizing and condensing the usually diffuse and scattered knowledge on proven solutions to teaching and training in software related matters. As with Design Patterns, every Pedagogical Pattern uses a common architecture including the following sections: intent, motivation, applicability, structure, consequences, implementation, related patterns, example instances and resources. Two complete pedagogical patterns, the PRCM (Peer Review and Corrective Maintenance) Pattern and the PIPR (Preparation, Industrial Presentation and Roundtable) Pattern are presented here.

1. INTRODUCTION

Training and educating people is a challenge with many hard questions and with no single easy answers: How to motivate students or trainees? How long should a presentation on new topics take? How to prepare them to real-world situations? How much work should they be assigned? Which kind of work should be assigned? How should they be graded? How to grade individuals when group assignments are carried out? How to distribute students in teams? When should you review and summarize? How can you build a course so that distinct instructors can teach it in a equivalent, repeatable fashion? The list is endless ...
In the engineering fields it is often the case that newcomers in the teaching “business” often hold an academic degree like a M.Sc. or Ph.D. but had no specific pedagogical education or training. If, like often is the case, there is nobody there to tell them the tricks of the trade, these novice teachers are condemned to learn by repeating the same mistakes that generations of peers did in the past.
On the other hand, I often come across experienced teachers that have long crystallized their teaching method. While they would argue their selection is based on the best tradeoff they came across on their lifetime career, I suspect that the real reason is that they had no other sources of pedagogical clues either than the outcomes of the trial and error approach they used in the past. Worst than that, they do not share their own clues found “the hard way” and the same situations arise again and again with others.
Faced with a lack of pedagogical help the OO education and training community has organized several birds-of-a-feather meetings such as the Educators’ Symposium at the OOPSLA conferences and the Symposium on Teaching Object Technology at TOOLS conferences. Although many interesting ideas are suggested in such meetings and also in education and training columns in journals such as JOOP or Object Magazine, very little has been done to produce an homogeneous and condensed repository of best practices.

Pedagogical patterns are a breakthrough in this status quo [Lilly96]. Their best contribution is not the originality of the approaches they advocate but the simple documentation of solutions that have been proven to do a good job in solving common problems, such as the ones mentioned above, in a reusable and adaptable format.
With a catalog of pedagogical patterns, even experienced professors will find useful hints from other colleagues. Any instructor should be able to instantiate a pedagogical pattern for different lessons or to adapt it using its own accumulated knowledge.

2. HOW ARE PEDAGOGICAL PATTERNS BORN?

During the last TOOLS'96 USA conference at Santa Barbara, California, in early August, I participated in the "Challenges and Successes in Teaching OT" workshop. There I was first faced with the pedagogical pattern concept, its architecture and a few examples. After a brainstorming session where a series of challenges and possible solutions were generated, I suddenly realized that I could formalize some of my successful experiences of a decade in the teaching and training business in a reusable and configurable format. I developed a rough draft of the two examples included in next section in this fashion. Back home I had the time to elaborate their current version and although

budget constraints didn’t allow me to be present, I submitted the two to the Workshop on Pedagogical Patterns during the OOPSLA’96 at San Jose, California, in October.
The purpose of the referred workshops is to gather the material to create, in the short to medium term, a publication with a comprehensive collection of pedagogical patterns, just like the “gang of four” did with design patterns [Gamma95].

3. SOME EXAMPLES

A bunch of pedagogical patterns were produced in the past few months [Lilly96] [Manns96]. These include the PDR (***Plan-Do-Reflect***) and the EU (***Extended Unit***) Patterns by Susan Lilly from IBM’s Object Technology University (USA), the LDLL (***Lab-Discussion-Lecture-Lab***) Pattern by Mary Lynn Manns from the University of North Carolina at Asheville (USA), the LASD (***Lecture-Activity-Student Presentation-Discussion***) Pattern by Helen Sharp from the Open University (UK), the TSL (***Programming in the Tiny, Small, Large***) Pattern by Billy B. L. Lim from Illinois State University (USA), the EPIC (***Explore-Present-Interact-Critique***) Pattern by Jorgen Lindskov Knudsen from Aarhus University (Denmark), the DIRR (***Design-Implement-Redesign-Reimplement***) Pattern by Steve Houk and the two below by myself. Like with Design Patterns, every Pedagogical Pattern uses a common architecture including the following sections: intent, motivation, applicability, structure, consequences, implementation, related patterns and example instances. The “resources” section was added here and was not part of the original proposal.

3.1 PRCM (Peer Review and Corrective Maintenance) Pattern

Intent

Expose students to:
- software quality techniques
- group dynamics situations
- software maintenance problems

Motivation

Software developers often have to do maintenance (either corrective, adaptive or evolutive) on products or parts of products that they did not produce. This is enforced with OO technology because of the emphasis on reuse where you tend to adapt and extend existing components.
“Traditional” programming courses, however, often do not expose students to maintenance tasks. Systems produced by students are evaluated “as delivered” and as so graded. Even if they get back to students for correction, they will still work on their own products, not anybody else’s.
On the other hand, students when faced with group assignments are tempted to cluster around the same colleagues over and over again, the ones with whom they feel better because they share common interests and views and with whom they do not conflict. Besides not mimicking the real world, this situation reduces the knowledge and insight sharing with members of other groups.
Last but not the least, verification and validation techniques used by students are often restricted to testing (mostly black-box). Software revisions in general, and software inspection techniques in particular, although recognized as very effective and efficient (due to their possible early adoption on the life-cycle) are seldom taught in OO software development courses. Enforcement of standardization rules and conventions is also often forgotten in academic assignments, although being of capital importance in industrial settings.

Applicability

This pattern can be used whenever there are team assignments to produce any software deliverable such as a requirements or design specification, source code, test battery, executable system, installation or user manuals.

Structure

i) Initial work
Students are grouped in “n” teams (T1, T2, ..., Tn). Teams ranging from 2 to 4 elements are usually appropriate. All teams are given similar effort assignments but preferably distinct in scope. All assignments have well defined (as formally as possible) requirements and correspond to the production of a given software deliverable. These requirements will be used for deliverable validation. The rules and conventions for producing the required deliverable are also formally defined and will be used for deliverable verification.
After a stated period of time each team Ti must have produced a prototype Pi of the desired deliverable.

ii) Peer Review

The peer review session occurs for all groups at the same time in a special lab class during regular class hours. Each prototype Pi is reviewed by a team composed of one chosen (by the team mates) element of team Ti plus all minus one elements of team Ti+1 (or by T1 for Pn).

By other words, each team is like a node in a ring structure where all prototypes produced are shifted one position for review (and maintenance as we will see further on). One student representing the original producer is present in the review session, to contribute with his (her) special insight derived from having been involved in its conception.

The Fagan model of software inspections, or one of its variants, is a good choice for conducting the review session. In this formal approach to reviewing, a copy of the deliverable to be reviewed is distributed to each participant beforehand to allow an independent verification and validation process.

The meeting objective is identifying, classifying and registering defects, not proposing solutions for them. Each participant has a specific role (moderator, presenter, recorder, producer) which can be accumulated for review teams with less than 4 people. Moderator is compatible with any other role. If possible, producer should not accumulate with either presenter or recorder, not to bias the review results. The moderator is responsible for avoiding problem resolution (which extends the meeting duration) and conflicts. The presenter is responsible for the meeting pace. The recorder fills the review report. The producer role is conducted by the member representing the team which produced the prototype.

The lecturer (or several teaching assistants) offers assistance wandering around teams and guaranteeing that interaction between them is kept to a minimum to minimize chaos. However, (s)he must keep a relatively low profile during this phase, allowing students to learn from each other, but helping to reconciliate possible contradicting conclusions.

This peer review meeting should take between one to two hours. Each team may complete the review activity at a different time. As they do so, they are invited to leave the lab, to avoid disturbing other reviews still taking place.

A copy of the review report is given to the lecturer.

iii) Corrective Maintenance

After doing a causal analysis based on all the review reports the lecturer gives a class summarizing the most common defects and their possible workarounds. Each team is assigned the task of corrective (defects found) and perfective (requirements not met) maintenance of the product they reviewed, not the one they produced. A given schedule for this rework effort is established.

Consequences

This pattern:

- requires additional work on the part of the lecturer to prepare the detailed requirements checklists;
- forces students to record, and to reflect upon, what happened when they were involved in their work assignments;
- leads students to learn how to delegate and share responsibilities;
- exposes students to the problems and challenges of maintenance;
- allows the introduction of software quality principles and techniques and actual practice for students in exercising them;
- increases the learning potential by sharing other students' experiences and knowledge;
- allows students to give and receive critique to and from peers;
- avoids the frustration of egocentric students when they are required to redo their first solutions.

Implementation

Issues to consider:

- This pattern can be used repeatedly within the same course. An entire course can be organized as a sequence of instances of this pattern. These instances should not overlap in time.
- The material to be reviewed should not be tiny (see TSL pattern).
- The rating of a team is split in two parts, one due to the prototype they produced and other to the rework. A careful weight of each of the two parts is important. This avoids the reduction in the willingness of students to provide a wholehearted improved solution on both phases. Equal weights can be an appropriate approach.

The review report is composed of:

- a section for the identification of the deliverable being reviewed, preparation times, date, review duration, etc;
- a section for the identification of the review members (and their role if formal reviews are used);
- a section for registering and characterizing the defects found (type, location, description, possible cause);
- a verification and validation (V&V) checklist.

The V&V checklist contains:
- identification of each elementary or atomic requirement (specific to each assignment) - this allows the detection of missing forward traceability (stated requirements not implemented) or reverse traceability (implementation features not reflected in design).
- identification of each rule or convention to produce (common to all groups) - this allows the detection of non adherence to the adopted standards (naming conventions, documentation guidelines, indentation, use of headers, etc).

Related patterns

- EPIC pattern, since it also advocates peer work.
- DIRR pattern, since it also advocates a rework phase, although in a different framework.

Example instances

This pattern has been used to teach OMT (Object Modeling Technique) and Object Pascal programming concepts (using Delphi). However, since it is defined in a general way, it can be used with any kind of software deliverable, either OO or not.

Resources

The instantiation of this pattern requires the availability of a lab or classroom where several groups of students can work with as few interaction as possible from other groups under the supervision of one or more lecturers (that have to circulate among the groups). The preferred option is to have an independent table for each group around which its members can sit.

3.2 PIPR (Preparation, Industrial Presentation and Roundtable) Pattern

Intent

Bridging the gap between academia and the "real world".

Motivation

It is often the case, mostly in countries where university staff spend their lifetime careers in academic campus, that software industry reality is quite different from the academic one. By the time students end their graduations and start working in industry, they feel completely unadjusted, either because their knowledge is not applicable or because they lack some basic understanding of what is day to day life in industry. This includes dealing with several constraints, human resources allocation, motivation, leadership, team dynamics, client pressure, marketing pressure, conflict resolution, technology migration problems, past experiences and selected tradeoffs and so on!
On the other hand, industry profits largely from having informal networks with university staff and students themselves because they can somehow indirectly shape the academic curricula to fit their real needs (by making the staff aware) and by establishing means of picking “eagles” among the students.

Applicability

This pattern can be used whenever the concepts being introduced in the classroom are being (or should be) used in production activities going on in the software industry.

Structure

i) Preparation
It is the author’s experience that much more insights are gained if students are taught what they are going to see or hear beforehand. This includes introducing, in the classroom, all the theoretical concepts related to the industrial presentation to be carried out at a later moment (eventually on another day). Students gain the ability to criticize and ask questions, that otherwise would only occur too late. Industrial presenter should also be “prepared” (informal meeting with one academic lecturer) in the sense that (s)he must be aware of which is the basic background of the intended audience (the students) in order to shape its presentation.

ii) Industrial Presentation
The theme of the presentation can be varied: tools adoption, migrating legacy systems, dealing with human resources, calendar and budget constraints, enforcing reuse adoption, configuration management procedures, subcontracting policies, verification and validation mechanisms in use and so on. These presentations can be carried out in class

(invited speaker approach). However, it would have much more impact if done at the industry premises (offices, labs, ...). Presentations should be carried out in normal working days when “real” activity is taking place. Weekend visits in empty premises are not a thrilling and enriching experience. If presentation involves a site tour, students should be split in small groups and taught to minimize the risk of being intrusive.

iii) Roundtable
After the presentation is over, all students, respective teachers and industrial presenter(s) gather in one room where they can ask and share with others whatever doubts, questions and comments they feel appropriate. The teacher may start with some preliminary questions, although (s)he may already know the answers, in order to stimulate the discussion session. This meeting should not take more than one hour.

Consequences

The PIPR pattern:
- provides the students (and in fact the lecturers themselves) with a more realistic view of what are the problems faced by real world projects in the software industry;
- enables lecturers to give more concrete, less abstract, lectures;
- allows students to realize "why" rather than on "how" things are done;
- invites students to reflect on what they have learned;
- makes the learning process more fun by introducing external agents and probably external visits (multinationals have long understood the benefit of conducting kickoffs).

Implementation

Issues to consider:
- this pattern can be instantiated several times in the same term;
- distant visits should be avoided because some students may not afford the costs involved and because it can take too much time.

Related Patterns

(none so far)

Example Instances

This pattern has been used to teach Software Engineering Project Management and Software Quality Principles and Techniques. Its generality is believed to allow it to be equally applied in other teaching areas.

Resources

The instantiation of this method requires that the lecturer have contacts in industry or that (s)he is allowed the opportunity and budget to visit a few industries in order to select, negotiate and plan the interaction events.

4. CONCLUDING REMARKS

Pedagogical patterns are an emerging approach for condensing the knowledge on proven and reusable solutions for teaching. Two pedagogical patterns were proposed here using a common skeleton. Although originally conceived to help teaching object technology there is no reason whatsoever why this same approach can not be applied successfully to teaching in many other areas of knowledge, either technological or not. Time, as usual, will be the final judge.

BIBLIOGRAPHY

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

[Lilly96] Lilly, Susan, “Patterns for Pedagogy”, Object Magazine, January 1996, pp.93-96.

[Manns96] Manns, Mary Lynn et al., workshop notes, Workshop on Pedagogical Patterns, OOPSLA’96, San Jose, California, October 1996.