Pedagogical Patterns from the Teachlet Laboratory

Axel Schmolitzky
Software Engineering Group
University of Hamburg, Germany
schmolitzky@acm.org

Introduction to Teachlets

Teachlets are an innovative teaching method that builds on executable code in a teaching unit. It encourages highly interactive classroom settings through introducing a running piece of software in its source code and setting a task to extend this software; the participants then have to find a solution collaboratively and to tell the moderator how to implement this solution in front of the audience. The method can be used in courses on any software-specific subject but is especially suited for teaching design patterns.

Teachlets can also be used in a seminar-like workshop, where students develop new teachlets, conduct these and get feedback on their work in a so called *Teachlet Laboratory*. By being in the teaching position, students get an even better understanding of the subject to teach.

The Teachlet concept and the Teachlet Laboratory have been introduced at the OOPSLA 2005 Educators' Symposium [9]. At the time writing this paper, the third Teachlet Laboratory is being conducted, with further to come. Another course at the University of Hamburg has adopted and extended the concept to produce learning material for advanced computer graphics topics [2].

This paper dissects the experiences from the laboratories into several pedagogical patterns [1] for the general context of *teaching about software*. They are presented below in order of increasing involvement of the students. The pattern language follows the one chosen by Bergin in [4]: All patterns are written in the you-form, talking to the teacher. In addition to the pattern name, each pattern is divided into four sections. The sections are separated by ***. The first section sets the context. The second describes the forces and the key problem. The third section outlines the solution, the consequences, limitations and disadvantages. The fourth section complements the discussion of the solution, by providing further information and examples.

SHOW IT RUNNING

You are teaching about a software tool or framework you want the students to use. You have slides that describe the properties (features, advantages, disadvantages, etc.) of the software well, maybe supported by some screenshots that illustrate the usage of the software.

***

Students tend to forget easily if they just hear about the functionality of a software; hearing somebody talk about using a software can be boring. You feel uncomfortable about the slides being too theoretical on their own, catching not enough interest. But the slides form a good base for learning for the exam at the end of the semester, so you want to keep them.

***

Use the software during your presentation. Students remember better if they have seen it working. Use some simple scenario that makes use of the software; the more the scenario shows the particular strength or weakness of the software, the better.

Make sure that the time you are investing is paying off. It can be quite time-consuming to work with running software; start-up time can be long, the firewall might need reconfiguration, the web server might not start, the database can be slow on your presentation machine.

***

If you are discussing layout management of components in a GUI framework, some well prepared resizable example GUIs will be far more instructive than any slide set.

If you want to discuss unit testing with JUnit, a running demonstration producing a red and a green bar is more impressive than pure slides.

Finding and preparing a good scenario can be time consuming; you have to weigh this against the improved learning effect.

Think aloud while using the software. Make sure that you explain everything you are doing with the software; it is new for the students and they are not as fluent as you might be.

SHOW PROGRAMMING

You are teaching about a programming language, a particular programming language concept or a programming technique (such as refactoring, unit testing, or a programming idiom). You are using well-prepared slides that discuss the subject with good source code examples.

***

If students ask about variations of the examples on your slides, you can only tell, not show (if you know the answer); if you don't know the answer, you and the students will feel uncomfortable and unsatisfied after the teaching unit. Even the best slides can be too inflexible for you to react on students' questions. Quite often students ask about variations of the examples shown. If you know the answer and tell it, things are good for you but not for the students; they just hear the answer, they do not see it working. Things are worse if you are not sure about the answer; so you answer "probably" and "try it yourself at home", which implies that students have to refocus on the question some time later, alone. Most will not do this, either due to time constraints or due to lack of interest.

***

Do not just show slides about programming, show programming as well. Start an integrated development environment (IDE) during your presentation. Explain the source code you provide, then show how you apply the concept you are trying to explain. Show how you make use of useful features of the IDE. When students ask about variations, you answer the question and then show the answer in action. This way, you provide a simple kind of TEST TUBE [5] during your presentation.

You need to be quite fluent in the language you are showing, but you need not be an expert. If you know every little detail about a language, you impress the students with your deep knowledge; but the students do not necessarily learn better by this. If you have to try the solution yourself to be sure about the answer, students feel closer to what you are doing; so sometimes it can help if you fake to not know the solution.

Limitation: You need more preparation time for the lecture. You have to check that the software is running on your presentation machine and you have to program the examples from your slides.

Limitation: It can be quite time-consuming to work with running software; start-up time can be too long, online documentation can be clumsy to use, the web server might not start, the database can be slow.

Limitation: This pattern starts to become bulky as soon as you try to compare language mechanisms in different languages; starting two or even more IDEs can be too much for one presentation.

***

If you want to discuss unit testing with JUnit, a running demonstration allows a better exploration of variations.

If you want to teach 'test first', doing it in front of the audience will be more instructive than a dry recipe.

The teacher, if an experienced programmer, can become a role model for the students, as she can show tips and tricks and can demonstrate best practices in the IDE and/or the programming language (see e.g. [8] for a timely discussion of apprentice-based learning).

Make sure that the source code is readable for the audience (font size, window arrangements, etc.).

TEACHLET

You are teaching a fundamental design pattern or an important programming language concept. You want to make sure that all students have a thorough understanding of the subject by letting them TRY IT YOURSELF [6].

***

Students will not understand well without applying the imparted knowledge; if they apply it on their own they do not get immediate and qualified feedback on their work which can manifest wrong understandings. Typically there is not enough time to set a task that students can solve offline and then to give each student individual feedback on the solution. A pure slide presentation, on the other extreme, is the most time-effective way of imparting knowledge, but feedback about students' individual understanding is typically sparse. You can improve the learning effect by applying SHOW PROGRAMMING, but you still feel uncomfortable about the engagement of the students; you want them to become ACTIVE STUDENTS [4].

***

Set up an environment where all participants of a teaching unit get to know a running system, then set a problem the audience is supposed and able to solve; let the audience agree on a solution and let them direct you to realize this solution, visible to all. Finally reflect thoroughly on the way the general solution was applied to the specific problem and on other ways or contexts where the general solution can be helpful.

Try to make sure that the initial system and the problem lead to the demonstration of a "killer example"; a killer example for a design pattern is one which "provides clear and overwhelmingly compelling motivation for the use of the pattern" (according to the "Killer Examples for Design Patterns" workshop series at OOPSLA).

Limitation: This pattern is oversized for teaching simple design patterns, such as Singleton or Factory Method, or for simple programming language concepts, such as conditionals or loops.

***

You can achieve that everybody can see the initial system for example by using a single presentation computer connected to a projector.

Introduce the running system both in its functionality (using SHOW IT RUNNING) and its internal structure (source code). Engage the audience by asking which clicks to perform or which class definition to show next. Make sure that everybody has a good understanding of the initial system and feels confident to extend the system; thus the initial system should be as small as possible, but not smaller. Even more as in SHOW PROGRAMMING, you should provide a TEST TUBE [5] for experimentation.

Provide information describing a general solution that can be helpful for the specific solution. You can do this before you set the problem or afterwards, depending on the difficulty of finding a solution for the problem.

Do not fall into SHOW PROGRAMMING, i.e. you being the main person in control of programming. The orders for the next programming step should always come from the audience. Ideally, during the design and implementation part of the Teachlet, the moderator becomes an INVISIBLE TEACHER [4], while the participants have a lively discussion about different alternatives in the form of a STUDENT DESIGN SPRINT [7] with a lot of REFLECTION [5].

Always have a programmed solution up the sleeve that you can show in case you run out of time.

TEACHLET LABORATORY

You are teaching a course on advanced software concepts (e.g. design patterns, advanced concepts of object-oriented programming, advanced computer graphics).

***

You want to engage the students as much as possible and use the time of the course as effectively as possible, but time constraints do not allow you to prepare teachlets for the whole course. You want to make sure that certain topics are covered and well understood. A pure lecture would be too much of a one-way street, with only little engagement of the students. Because the subject of the course is covering a field with rapid change (e.g. advanced computer graphics), the overhead for keeping slides or teachlets up to date is too much on your side.

***

Let students design teachlets; let them conduct these teachlets and organize intense FEEDBACK [6] after each. Having to prepare and conduct a teaching unit with slides and running software implies a deeper involvement of the preparing student with the subject; she becomes an even more ACTIVE STUDENT [4] than an active participant of a TEACHLET.

If a teachlet is ill-prepared or the student is not a good presenter, an important topic might not get the appropriate coverage. You have to be prepared to give additional background on the topic in the feedback phase. Typically the feedback phase should also include PEER FEEDBACK [6].

Limitation: Designing a teachlet requires some creativity. If students do not like to be creative, this pattern can be too demanding.

Limitation: Conducting a teachlet requires several soft skills (the self-assured handling of an IDE, slides and a video projector in front of a group of people, the moderating of design discussions). If students are not self-confident or experienced enough, this pattern can be too demanding.

***

If you have some flexibility for your course content, you can let the STUDENTS DECIDE [4] for which learning goals they should build their teachlets.

If students do not want to be creative, they can do a *teachlet replay*: they take a teachlet from a previous laboratory, work it over and conduct it again. This is an implementation of ADOPT-AN-ARTIFACT [4].

If students are not self-confident enough on their own, you can let them prepare and conduct their teachlets in pairs, as GROUPS WORK [4] often better; this worked well for one pair in the last teachlet laboratory conducted by the author.

In a postgraduate course on advanced computer graphics at the University of Hamburg, students developed teaching material that could be used in a teaching unit as well as for individual offline learning (in a format that could be submitted to CGEMS, a computer graphics educational materials server). Especially here, a teachlet laboratory can provide lasting material as STUDENT EXTENDS [3].

In the last Teachlet Laboratory conducted by the author, students anonymously graded different aspects of each others teachlets after the FEEDBACK of each unit on prepared ballots. The results were presented at the end of the laboratory as one form of PEER GRADING [6].

Acknowledgements

My thanks go to the participants of the first three teachlet laboratories for their great commitment, it really endorsed the concept. I also thank my colleagues in the Software Engineering Group at the University of Hamburg for several fruitful discussions on teachlets and possible variations. Steffi Beckhaus had the patience to listen to my ideas and adopted them for her course on advanced computer graphics. Christian Späh supported me with his enthusiasm about teachlets and contributed the peer grading ballots to the latest teachlet laboratory.

References

[1] *The Pedagogical Patterns Project*, http://www.pedagogicalpatterns.org (last visited July 21, 2006).
[2] Beckhaus, S., Blom, K. J.: “Teaching, Exploring, Learning - Developing Tutorials for In-Class Teaching and Self-Learning”, *EUROGRAPHICS '06 (Education Papers)*, Vienna, 2006.
[3] Bergin, J.: “Active Learning and Feedback Patterns”, *PLoP '06*, Portland, Oregon, 2006.
[4] Bergin, J., Eckstein, J., Manns, M. L., Sharp, H.: “Patterns for Active Learning”, *PLoP '02*, Monticello, Illinois, 2002.
[5] Bergin, J., Eckstein, J., Manns, M. L., Wallingford, E.: “Patterns for Gaining Different Perspectives”, *PLoP '01*, Monticello, Illinois, 2001.
[6] Eckstein, J., Bergin, J., Sharp, H.: “Feedback Patterns”, *EuroPLoP '02*, Irsee, Germany, 2002.
[7] Eckstein, J., Manns, M. L., Wallingford, E., Marquardt, K.: “Patterns for Experiental Learning”, *EuroPLoP '01*, Irsee, Germany, 2001.
[8] Kölling, M., Barnes, D. J.: “Enhancing Apprentice-Based Learning of Java”, *SIGCSE 36*, Norfolk, Virginia, 2004.
[9] Schmolitzky, A.: “A Laboratory for Teaching Object-Oriented Language and Design Concepts with Teachlets”, *OOPSLA '05 (Companion: Educators' Symposium)*, San Diego, CA; ACM Press, 2005.