A Laboratory for Teaching Object-Oriented Language and Design Concepts with Teachlets

Axel Schmolitzky
University of Hamburg, Germany
Vogt-Koelln-Str. 30
D-22527 Hamburg
+49.40.42883 2302
schmolitzky@acm.org

ABSTRACT
Teachlets are a new method originally developed to teach design patterns. Based on executable code, a problem is set that is to be solved collaboratively and interactively by all the participants of a teaching unit. A moderator is on hand to operate the computer, the development environment and video projector when called upon to do so. While deployable in themselves as an innovative teaching method, teachlets can also be used as a design object in seminar-like workshop events. In the course on advanced concepts of object-oriented programming described here, the participants themselves developed and used teachlets in a so-called teachlet laboratory.

Categories and Subject Descriptors
K.3.2 [Computers & Education]: Computer & Information Science Education - Computer Science Education
D.2.2 [Software Engineering]: Design Tools and Techniques - Object-oriented design methods

General Terms
Design

Keywords
Classroom Management, Courseware, Teachlet, Design Patterns, Pedagogy.

1. INTRODUCTION
Design patterns (see e.g. [7]) are an interesting challenge in computer science education. There are a variety of ways in which they can be taught, ranging from simple lectures without practical exercises to loosely supervised assignments suggesting the use of design patterns. Since design patterns are closely connected with practical design experience, teaching them in simple lecture form runs the risk of merely imparting a series of catchwords. However, using the lecture form can ensure that the most important design patterns are covered. On the other hand, loosely supervised assignments with a large practical element (i.e. practicals) offer the advantage of gaining individual practical experience. But they have their disadvantages, too: they usually provide too little scope for individual feedback on the respective design decisions, and some participants may not have the opportunity to familiarize themselves with all the relevant design patterns.

Teachlets adopt a middle course between teacher-centred instruction and individual learning by providing the opportunity for all participants to work collaboratively on a problem in the context of a specially prepared teaching unit. This approach also takes into account Vlissides’s emphatic reminder in [4] that design patterns should always be considered in conjunction with the problems they solve. While originally developed for teaching design patterns, the concept of teachlets would appear to have the potential for much broader application. The present article focuses mainly on the use of teachlets for teaching design patterns because most of the experience gained is in this area. The Outlook section considers other potential application areas.

The concept was presented for the first time in a project seminar held at the University of Hamburg in the summer semester of 2004. It was very well received by the participants. The seminar’s organizer initially conducted a number of teachlets himself in order to illustrate the concept. After this, each participant developed at least one teachlet and tried this out on the group as a whole. The seminar was repeated in the summer semester of 2005.

This article is primarily an experience report. Section 2 presents the basic concept of a teachlet. Section 3 gives an example of its use, while Section 4 summarizes the experience gained so far when conducting teachlets. Section 5 looks at the concept of a seminar-like teachlet laboratory in which the participants themselves design teachlets and thus have to make a detailed study of the material being taught. Section 6 looks at other possible application fields and examines the potential for further exploration of the approach.

2. THE TEACHLET CONCEPT
The following definition forms the programmatic basis of the teachlet concept:

A teachlet is an interactive teaching unit in which an executable piece of software is to be extended by a clearly defined functionality in order to illustrate a design pattern or a programming language concept. A moderator uses a computer and a video projector to present the initial system and the proposed extension and is then given directions by the participants on the necessary changes to be made to the source code.

The basic idea of a teachlet is thus to use this practical element to help participants achieve greater learning success than is possible through purely theoretical discussion. The emphasis here is on the
word *interactive* because a teachlet moderator does not simply demonstrate a practical application but also motivates the participants to actively collaborate in designing a solution.

2.1 Static Structure of a Teachlet

The following four elements are essential ingredients of a successful teachlet and should be given explicit consideration during its preparation:

- An *initial system*: an executable application that is as small as possible and that is available in source code.
- A *learning goal*: knowledge of a design pattern or a programming language concept that has been dealt with practically and understood.
- One or more assigned *problems*: they involve extending/modify the initial system; during problem solving, the learning goal should be achieved by collaborative programming.
- A *set of slides*: not only to present the initial system, the learning goal and the assignment and to explain the basic terminology, but also to discuss alternative solutions and generalizations; may be supplemented with handouts.

The initial system should be small enough to enable it to be explored by the participants in 15-20 minutes. Here a clear distinction should be made between (externally visible) functionality and (internal) architecture. The functionality should be clearly described and easy to explore. Initial systems with a GUI have proved to be highly motivating.

2.2 Dynamics of a Teachlet

The quality of a teachlet only becomes evident when it is conducted in the context of a teaching unit. Though based on the same static structure, the teachlet can take different forms depending on the persons involved. A teachlet used in a teaching – or better learning – unit is henceforth called a *teachlet unit*. The relationship of a teachlet unit to the static description in the previous section is like that of an instance to its class in object orientation. A typical teachlet unit lasts 90 minutes.

Each teachlet unit has a number of *participants* and a *moderator*, who conducts the teachlet using a computer (the *teachlet computer*), which is connected to a *video projector*. Presentation software and at least one development environment are installed on the teachlet computer. Using the development environment, the moderator explains the initial system, in some cases with the help of slides, as well as outlining the learning goal and the problem to be solved. From this point onwards, the moderator plays a largely passive role, carrying out the directions given by the teachlet unit participants to solve the problem. The moderator can intervene at any time if the suggestions made overly distract from the desired learning goal, but should at all costs avoid simple “show programming” of the solution. Moderators usually have a mental “choreography” for conducting the teachlet. They should always have a complete solution in reserve that they can present should the need arise.

2.3 A Typical Choreography

To begin with, the initial system is usually embedded in a little story, especially in cases where – for didactic reasons – the description has been greatly simplified compared with a real system. Next, the moderator starts the application, carries out as brief as possible a demonstration and then encourages the participants to make suggestions as to how to further explore or re-explore the functionality. This enables the participants to familiarize themselves with the functionality interactively.

Only when the functionality has been clearly described should the architecture be studied. It is here in particular that the moderator takes a low profile (“Which class’s source code shall I show first?”), following the directions given by the participants (“Please start the application again. I want to see it executed.”, etc.).

Once all the participants are familiar with the internal and external views of the initial system, the moderator sets the (first) problem. Ideally, this is formulated on at least one slide. Mostly the problem is to extend the functionality (“we wish to create an *Undo* option”), but this is not necessarily the case. Design patterns in particular are often used where the internal architecture is not sufficiently maintainable. A typical problem might therefore be to minimize the number of classes or make it easier in the future to replace a portion of the source code.

At least 30 minutes should be planned for the highly interactive implementation phase. Here, the moderator must ensure that there is always enough time left to summarize. The summary, supported by suitable slides, should recapitulate on everything that has been demonstrated and discussed. It should also present alternative solutions and other application options for the concept being taught. The final phase of a teachlet unit thus reverts to the conventional form of presentation.

Ideally, at the end of a teachlet all participants should be provided with the source code of the collaboratively developed solution(s). This enables them, if needed, to go over again by themselves the material they have learned.

3. AN EXAMPLE

To better illustrate the theoretical ideas presented in the previous section, we now go on to look at a practical example: a teachlet for teaching the design pattern *Command*. It was developed in the teachlet laboratory described in Section 5 and has already been used in a number of other contexts.

3.1 The Exposition

The teachlet unit begins with a demonstration of the initial system. Figure 1 shows its graphical user interface. The system in question is a very simple system for practising the four basic arithmetical operations, such as might be suitable for use by elementary school children.

![Figure 1. The GUI of the initial system.](image)

The interface consists of a window, the top line of which contains five buttons: the symbols for the four basic arithmetical operations and the equals sign. The second line contains a so-called initial
value (to begin with: 1) and a randomly chosen operand. These are followed by two buttons that initially have no function: the first has an undo arrow, while the second is for showing a protocol, to be displayed in the empty list box on the bottom right. The actual mental arithmetic training is now done as follows: the user selects one of the four operators by pressing the appropriate button so that it is applied in the background to the number constituting the “initial value” (first operand) and the value constituting the “operand” (second operand). The resulting value forms the first operand in the next operation, while in the “operand” box a new randomly selected value between 1 and 9 appears, which in turn represents the second operand in the next operation. The original “initial value” is now greyed out to indicate that it is no longer directly involved in the following operations. Finally, by clicking on the equals sign, the current result is shown, in black again in the “initial value” box. In the next series of operations, it thus constitutes the new initial value.

The program is started and the moderator demonstrates the functionality the first time round. The moderator then encourages the participants to tell him/her what actions to take, this procedure being repeated until all the participants are familiar with the system’s functionality. This introductory phase should last approx. 5 minutes, 10 at the most. A slide is then shown summarizing the functionality again.

The next step is the code inspection. The system consists of three classes, which are looked at in detail interactively. All participants should be familiarized with the internal structure of the application. This phase too should take no more than 10 minutes.

### 3.2 Problem, Proposed Solution, Implementation

Once all the participants are conversant with the existing situation, the problem can be set. In this case it is to implement the still missing protocol, which shows the clicked operators, together with their respective operands, when the appropriate button (“show protocol”) is pressed. It should also be possible to undo already clicked operations. This problem is formulated on a separate slide.

After explaining the problem, the moderator suggests using the Command pattern to solve it. The calls to objects of class IntermediateResult along with their computation operations are to be quasi “objectified”. These command objects can then be stored, displayed in readable form and also deleted. The moderator shows a slide portraying a general class diagram of the command pattern and then, with the collaboration of the participants, maps this to a concrete design on the board. 10-15 minutes should be planned for this phase.

Once all the participants have agreed on the class names in the design, it is implemented collaboratively. Depending on the preferred programming culture, an incremental, iterative approach can be adopted or everything can be done in one big development step. A macro command can also be implemented, depending on the amount of time available. At least 30 minutes should be planned for this phase.

### 3.3 Summary, Conclusion

The design and the implementation are summarized using previously prepared slides. On one of the slides the generic-pattern participants and the concrete-implementation participants are juxtaposed (here the names can differ from those chosen by the group). Another slide shows the specific class diagram of the solution. Here, too, the names will very likely differ. Finally, slides are shown pointing out the advantages of the Command pattern and indicating other potential applications. All this should take no more than 10 minutes.

### 4. EXPERIENCE WITH TEACHLETS

Our experience so far with the use of teachlets has revealed different choreography patterns.

Some teachlets began directly with the presentation of the initial system, dispensing with introductory slides on the learning goal (no initial “dry run” presentation of a design pattern). The problem was addressed and it was subsequently “revealed” that what was being used was something that is normally termed XY (e.g. an Adapter). This generally made sense in cases where the design pattern itself was not very complicated and was primarily used for terminological purposes.

Other teachlets began with a description of the learning goal because the design pattern was too complex to be derived by the participants themselves during the learning unit. This was the case with a teachlet on the Interpreter pattern and would presumably also be the case with the Visitor pattern. Here the basic structure of the pattern is first presented, and then the pattern is applied to a concrete problem.

Contrary to what was said in the previous section, it may also be helpful to set the problem before the architecture of the initial system is studied. If the problem is confined exclusively to the functionality, it may even be a better idea for the users to familiarize themselves with the source code in a problem-oriented manner.

### 4.1 Advice for Moderators

The job of teachlet moderator is a demanding one – more demanding than, say, simply lecturing using slides. Basically, the usual advice given for lectures with slides also applies to teachlets, particularly for the slide element.

Moderators are often seated when moderating a teachlet, which may seem strange, especially when showing the slides. Moderators should be aware of this in advance. They should also be familiar with the teachlet computer to enable them to make routine use of shortcut keys (e.g. Ctrl-v).

From our experience so far with teachlets, we have been able to derive the following advice for teachlet moderators:

- **Make the problem as clear as possible.** If the problem is not properly understood by all the participants, this often holds up the teachlet. Ideally, the problem should be formulated on specially prepared slides. The moderator should ask explicitly if all the participants have understood the problem; conspicuously passive participants should be explicitly encouraged to ask questions if they have not properly understood everything.

- **Bear the silence.** If the participants of a teachlet are quiet, this is often because they are familiarizing themselves with the source code. This takes time, and they should be given the time they need.
• Keep control of the situation. There are frequently detailed discussions among the participants. These are desirable but should not get out of hand. If necessary, poor contributions should be dealt with briefly and prolonged discussions should be cut off. The moderator should not lose sight of the choreography. One piece of advice that has proved useful: don’t try to get participants to return to the basic thread of the discussion by jumping ahead to the next step but rather by summarizing what has been learned so far and by asking what comes next.

• Think out loud. All the things that happen in the development environment should be explained. A running commentary should be given on all the tools as they are used (“Now I’m translating this”, “This is where I use an Eclipse refactoring”, etc.). Unusual or rarely performed activities in particular must be explained.

• Conquer the solution space. There are often several solutions to a specific problem. Moderators should be prepared to deal with different proposed solutions. They should be ready to offer good reasons for rejecting solutions that distract attention from the learning goal. At the same time, they should be open to good suggestions they hadn’t previously thought of themselves.

• Deliberately repeat certain actions. Repetition helps get participants who have been temporarily distracted “back on track”. At regular intervals moderators should briefly recapitulate on the material handled up to this point. A good way to introduce such recapitulations is by asking: “What have we done so far?”

• Use class diagrams for visualization. If a standard pattern from [7] is presented, the slides should at least show the generic class diagram of the pattern. In addition, a specific class diagram is often desirable for the concrete system. Depending on the teachlet, it is helpful to show either a diagram of the initial system or a diagram of the final system.

• Have a good command of necessary fundamentals. If the inner classes of Java are a good choice for, say, implementing the design pattern Iterator, the moderator should both be conversant with them and have slides in reserve to help explain the necessary fundamentals if required.

5. A TEACHLET LABORATORY: LEARNING BY TEACHING

“I hear, and I forget.
I see, and I remember.
I do, and I understand.” (Confucius)

This saying reflects the insight – especially valid in computer science – that doing is the best way to achieve a lasting understanding of a problem (see e.g. [6]). By way of a motivation for the above-mentioned course on advanced concepts of object-oriented programming, we added the following phrase to the saying:

“I teach, and I really understand”.

The basic idea of the course thus rests on the assumption that someone who is required to teach others a concept must necessarily reflect more intensively on the material than someone who is merely performing actions associated with the concept.

The following distinction can be drawn between a teachlet laboratory and a teachlet:

A teachlet laboratory is a workshop-like course in which the participants develop new teachlets. To this end, each participant must pursue at least one learning goal and design a teachlet for this purpose. The new teachlets are tried out on the laboratory participants and are then analyzed and evaluated by them.

A teachlet laboratory is therefore something quite different from a fictitious course on, say, design patterns, in which the lecturer uses only proven teachlets. The – already large – interactive component of a teachlet is thus further enhanced in a teachlet laboratory.

5.1 First-Time Deployment

The definition of a teachlet given in Section 2 was presented to the participants at the first meeting. At the next two meetings, two teachlets were conducted by the organizer of the course to illustrate the concept. At subsequent meetings, the participants conducted the teachlets they had developed themselves. During the preparation of these teachlets, they received the same sort of close supervision as is often given in conventional seminars: a rough outline of the teachlet had to be submitted to the organizer at least two weeks before the meeting, and all the static elements of the teachlet had to be presented at least a week before the event.

All moderations were conducted without outside help. Each teachlet unit was followed by intensive feedback at various levels: on the learning goal itself, on the teachlet and on the moderation. Feedback was given by the participants themselves and the organizer.

5.2 Observations

The attempt to conduct a teaching unit on a design pattern in the form of a teachlet does actually involve addressing the learning goal in depth. All the students that acted as teachlet moderators had a good command of their material.

Not everyone is well suited to the role of teachlet moderator. One of the basic requirements is the ability to talk fluently and coherently on a subject using slides. Not all participants were equally well equipped in this respect. Other soft skills are also needed: the ability to constantly reflect on which – and to what extent – interventions by participants can promote understanding of the material, and to repeatedly summarize the material already covered. It is, after all, the moderator’s job to constantly monitor how the teachlet is progressing. Moderators must also make their presence strongly felt in discussions, for they must remain in control at all times. These are just the sort of skills required of software engineers. When people are moderating a teachlet, it quickly becomes apparent where their weaknesses lie and what they need to do to improve their skills.

At the meta level, there were a number of discussions on programming styles as well as on the respective integrated development environment (IDE) used: Eclipse [5], IntelliJ IDEA [8] or BlueJ [3]. In particular the professionally oriented environments Eclipse and IDEA were frequently the subject of brief discussions. The organizer felt that these discussions were
extremely fruitful because they underlined just how important the IDE tool is for software engineers. There were lively exchanges on the support options offered by the IDE (especially for refactorings), which was quite appropriate when dealing with object-oriented design.

6. ASSESSMENT AND OUTLOOK

6.1 The Teachlet Concept

6.1.1 Assessment
The teachlet concept has proved successful for teaching design patterns in groups of up to 16 participants. Feedback from the participants has confirmed that a problem-based teaching method (f61) is highly motivating, especially for learning about design patterns. The advantages of closely integrating theoretical instruction with practical work when teaching programming have also been recently pointed out by Kolling and Barnes in [9]. The teachlet concept represents a further step towards breaking down the traditional division between lecture and practical. By focusing on handling software artefacts, it constitutes a highly interactive form of teaching and learning. Teachlets thus offer a way to increase the effectiveness of course teaching.

In [9], Kolling and Barnes address another interesting point that is also highly relevant to teachlets: the teachlet moderator can serve as a role model demonstrating best practices in software engineering.

Some of the pedagogical patterns (see [2]) can be recognized in the teachlet concept: Clearly Try It Yourself from the Feedback Patterns catalog and Active Student from the Patterns for Active Learning, but also some variation of Student Design Sprint as a pattern for active learning.

One drawback of teachlets is that they can only be evaluated in real-life teaching situations. This means that they take a long time to “mature”.

6.1.2 Outlook
Future work in this area will need to explore the potential for varying a teachlet’s content and number of participants:

- So far, little experience has been acquired with teaching programming language concepts, though this was originally planned for the teachlet laboratory. Because of the options available to participants, there were no teachlets on multiple (implementation) inheritance, genericity or multi-methods, for example. More experience must be gained in this area. The use of teachlets could also make the teaching of algorithms more interesting because embedding an algorithm in a problem context is likely to prove highly motivating for students.

- Another point that needs looking into is the scalability of the method, i.e. the maximum group size for which the concept is implementable. With very large groups it is not possible to actively involve all participants. Even so, the use of teachlets could prove fruitful even for large groups because it would probably enable the passive participants to share in the interactive problem-solving process, which is driven by the more active participants, to a greater extent than in a purely teacher-centred situation. Further experiments are required in this area.

For the development of existing teachlets, the following maturity levels have been proven useful:

- Preliminary teachlets are teachlets that have only been conducted once after their creation (in a teachlet laboratory for instance). For such teachlets, typically only a set of slides and the initial system exist.

- For a documented teachlet, there must also be a documentation that describes the teachlet and gives a possible choreography (ideally complemented by time measurements of the sections from the first teachlet unit). Further a target system should be provided that illustrates one possible solution. Since most teachlets have so far only been conducted by their developers, the aim here should be to enable sufficiently competent persons to conduct teachlets with which they are unfamiliar with a minimum of preparation.

- An approved teachlet is a documented teachlet that has not just been conducted by its developer, but has been conducted successfully by at least one other person using the documentation. The second person should have complemented the documentation with her own experiences.

Teachlets that have proved fruitful and are sufficiently mature should be made available to other teachers. Another conceivable option is to make a coordinated collection of approved teachlets covering a wide variety of learning goals. These could be made accessible via a dedicated website and integrated in, say, MuSoT [1], a German portal offering access to multimedia teaching material for software engineering courses.

6.2 The Teachlet Laboratory

6.2.1 Assessment
The first laboratory focused specifically on the development of new teachlets. The fact that the teachlet concept is largely formalized made the students’ job easier.

Developing a good teachlet is highly motivating for students. It requires creativity for elaborating the basic idea and practical skills for developing the initial system. Conducting the teachlet also requires the self-assured handling of an IDE, slides and a video projector in front of a group of people, which is a good way of training various soft skills. And finally there is the prospect of the results being made available to a larger public. All these factors contributed to the high motivation of the teachlet laboratory’s participants.

During the second laboratory in the summer semester of 2005, several of the teachlets from the first laboratory were used again. Especially students that were not too happy about having to be creative embraced the notion of a teachlet replay, i.e., conducting a teachlet unit with a documented teachlet.

6.2.2 Outlook
It is planned to conduct the laboratory again in the summer semester of 2006. Some of the teachlets from the first two laboratories will be used again – in revised form – at this event.

Operating a computer with slides and an IDE requires a great deal of attention, as does moderating a technical discussion in a large group. One conceivable option would therefore be to have two moderators. This is probably not feasible when using teachlets on
a large scale in university courses, but it is a reasonable option in the context of a workshop-like laboratory. The students could initially develop and moderate new teachlets in pairs, and then from the second teachlet onwards do the moderation individually.

7. ACKNOWLEDGEMENTS
I wish to thank all the participants of the first two teachlet laboratories for their help and commitment. Their feedback has confirmed the viability of the concept. My thanks also go to my colleagues in the University of Hamburg’s Software Engineering group for the fruitful discussions we had on teachlets and the helpful suggestions they made regarding modifications. I am indebted to Heinz Züllighoven for the original idea to develop the teachlet concept. It was he who suggested conducting a small interactive design pattern workshop two years ago to fill an unexpectedly vacant seminar slot. Finally my thanks go to the anonymous reviewers, especially those of the German predecessor [10] of this paper.

8. REFERENCES