Teaching Pattern-Based Development

Andreas Seitz
Technical University of Munich (TUM)
Department of Informatics
Munich, Germany
seitz@in.tum.de

Bernd Bruegge
Technical University of Munich (TUM)
Department of Informatics
Munich, Germany
bruegge@in.tum.de

***Abstract*—The use of patterns in software engineering is an important and widespread concept. However, teaching patterns to students is challenging because it requires practical knowledge. To evaluate the usefulness of patterns, students need to remember, and understand them as well as apply and analyze them. Pattern-based development (PBD) is a model-based development approach that focuses on the reuse and extensive use of patterns throughout the software lifecycle. In this study, we describe the foundation for teaching PBD to large classes using an inductive and formative approach. We encourage students to apply patterns in any stage of the software lifecycle. We apply agile methodologies, particularly Scrum, to structure lectures and give students the opportunity to apply patterns in each iteration. An increment (in our case a simple game) is reviewed and assessed by the instructors after each iteration. In a case study, we demonstrate the use of this approach to teach PBD in two university courses with 500 and 1400 students. From our experience, we conclude that teaching PBD in large courses works well and discuss the best practices for other instructors.**

I. Introduction

The application and use of patterns in software engineering are essential. However, teaching these concepts is challenging. Patterns serve as a reusable solution for recurring problems; students should be able to internalize this solution and recall an applicable pattern by name. Software engineering requires collaboration and practical application of knowledge [1], [2]. This is particularly true for applying patterns in software engineering.

While the theoretical delivery of the structure and composition of patterns is easy, the challenge lies in practically imparting the knowledge. Students should be able to effectively apply patterns in their university projects or later in their professional careers. We aim to impart theoretical and practical knowledge about patterns while also trying to establish patterns as a language for the students. The approach we are pursuing must also be applicable to an increasing number of students. The number of students in our department has risen by 67% between 2013 and 2016. Since teaching in private lessons or small groups is not feasible, a method for teaching pattern-based development (PBD) in large classes is needed.

By developing a new teaching methodology for PBD, we have introduced a foundation that includes the delivery of theoretical knowledge about patterns and the skill of practically applying them. We apply Bloom's framework to classify the expectations of what students should learn as the result of an instruction session [3]. Bloom classified six major categories of cognitive processes ordered by complexity from lowest to highest: knowledge, comprehension, application, analysis, synthesis and evaluation. The goal of teaching PBD is that students are able to understand and remember the learned concepts. In our courses, we try to teach patterns as interactively as possible. Our goal is to encourage students to achieve cognitive processes at higher levels of the pyramid and to teach students to analyze, synthesize, and evaluate the imparted knowledge. Using this technique for teaching PBD we can attain all of Bloom's categories of cognitive processes and allow students to appreciate the beauty of PBD. We aim to ensure that each element (class, component, or source code) of a system is covered by a pattern. A coverage or *pattern traceability* of 100% is desirable, yet difficult to achieve.

The remainder of this paper is structured as follows. Section II explains the foundations on which we have based the proposed approach for teaching PBD. Teaching PBD is described in Section III. In Section IV, an application of this concept in two large university courses is presented. In Section V, we discuss our experiences and share our best practices with other instructors interested in teaching PBD. In conclusion, Section VI summarizes the paper and offers an outlook into our future work.

II. Foundations

The following foundations serve as a basis for the development of the proposed approach for teaching PBD.

A. Active, Interactive & Chaordic Learning

Interactive learning involves the combination of lectures and exercises into interactive classes with multiple iterations of theory, examples, exercises, solutions and reflection [4]. It is based on the concepts of active, computer-based and experiential learning and focuses on providing immediate feedback to improve the learning experience of a large group of students. In this approach, educators teach and exercise small chunks of knowledge in short cycles and students receive immediate feedback regarding the exercises so that they may reflect and gradually increase their knowledge. This approach requires the active participation of students and the use of computers (laptops, tablets, or smartphones) in the classroom.

Chaordic learning is an educational approach that combines theoretical and experimental learning and includes aspects of both order and chaos (structured courses with detailed

instructions represent order while experimental learning and educational innovation represent chaos) [5], [6].

Interactive and chaordic learning serve as the foundations on which teaching PBD is based.

B. Pattern-Oriented Analysis and Design & Pattern-Based Engineering

Yacoub said, "design patterns are immensely powerful, but to build large-scale robust systems, you need more." [7] Yacoub introduced pattern-oriented analysis and design (POAD), a methodology for composing proven design patterns into reliable and robust large-scale software systems. POAD can be used to quickly create systems that are robust, scalable and maintenance-friendly thanks to the use of UML class diagrams as building blocks. Yacoub found that the most difficult part of software development is not programming but rather the decision making required in the design phase. These design decisions are integrated into the system for its entire lifecycle [7]. Based on this concept, Ackerman developed pattern-based engineering (PBE); a systematic, disciplined, and quantifiable approach to software development. PBE involves the use of pattern specifications and implementations throughout the software development and delivery process [8].

III. Pattern-Based Development & Pattern Traceability

PBD is a model-based development technique that focuses on the reuse and extensive use of patterns during analysis, system design, object design, testing, and build-and-release management. For each phase of the software lifecycle, different patterns are available and can be applied. For example, architectural patterns are used during system design (cf. Figure 1). Antipatterns can occur in any phase and are therefore also included when teaching PBD. This approach to teaching PBD follows an inductive and formative approach. Traditionally, a university course is divided into 12-14 lectures. In each lecture, a set of patterns that can be applied in a certain phase of the software lifecycle are taught. In an interactive learning approach, these patterns can be applied in small exercises after the delivery of the theory.

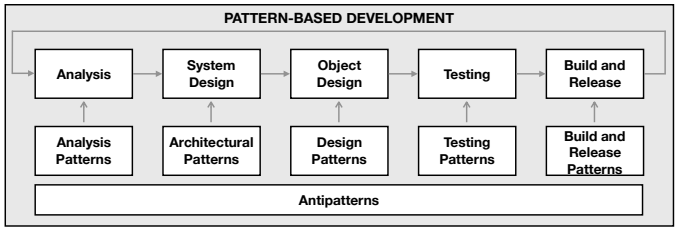

Fig. 1. Pattern catalog for PBD: Patterns can be applied and used throughout the software lifecycle: analysis, system design, object design, testing, and build-and-release.

As the course progresses, students learn and apply a variety of patterns. To further deepen the students' understanding of the patterns and their application, a PBD-specific lecture is held at a specific point in time (cf. Figure 2). In this lecture, students apply already learned patterns using a coherent example. The PBD lecture is comparable to a hackathon: the lecture begins with a problem statement and students work on the problem throughout the lecture and must achieve and deliver the results by a defined deadline. Instructors choose a subset of the patterns already learned and skillfully formulate them in the problem statement. The structure of the PBD lecture is important and is therefore explained in more detail in the following section.

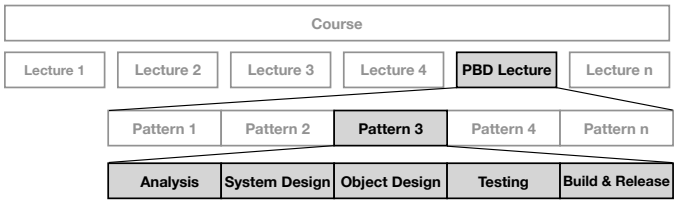

Fig. 2. Structure of a course in which PBD is applied: In lecture 1 through n, patterns are taught and these patterns are then applied in a PBD lecture that involves several iterations of applying individual patterns to a certain phase of the software lifecycle.

A. Structure of the PBD Lecture

The PBD lecture is based on the Scrum[1] framework. Similar to Scrum, the lecture is divided into several short iterations. Based on a problem statement, product backlog items (PBIs) are derived and iteratively implemented. The PBIs are created by the instructors and are rather generic and intended to stimulate discussion. When formulating the problem statement, hints to specific patterns should be given. For example, the phrase "the application must be compatible with the existing system" indicates the application of the adapter pattern [9]. The amount of work for the PBIs should be feasible to complete in the given timeframe, which is a challenge for instructors. Instructors may provide code skeletons and parts of the implementation needed for specific components that can be integrated into the increment. These components can be made available to the students for each iteration (for example as a git repository).

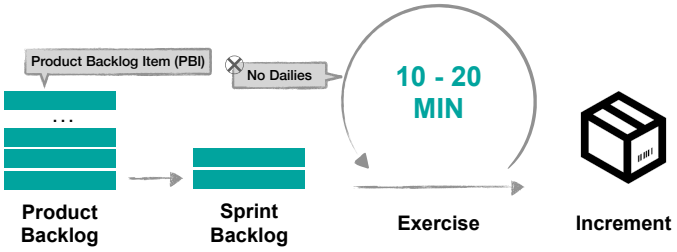

Fig. 3. Exercise iteration for teaching PBD—adopted from Scrum.

Each iteration is either an in-class exercise (IC), a homework (HW), or a tutorial (T) according to the definitions of different types of exercises given in [4].

[1] https://www.scrum.org/

B. Artifacts

PBD involves three artifacts that correspond to the artifacts defined in Scrum: the product backlog, the sprint backlog and the increment. Instructors can change and add PBIs in the product backlog during the lecture and introduce new requirements or make changes. The sprint backlog is a subset of the product backlog. Before an iteration, instructors select PBIs to be implemented and thereby determine which patterns the students should apply. As soon as the exercise iteration is started, the sprint backlog cannot be changed. At the end of the iteration, students must deliver an increment to the instructors (via continuous integration and delivery), which serves as a basis for evaluating the students' results.

C. Roles

During the PBD lecture, instructors serve as a proxy customer and take the role of a product owner in Scrum. The instructors are responsible for the product backlog and the sprint backlog. The students can interact with the instructors to clarify PBIs. The Scrum framework is adapted to our needs. The Scrum master (as defined in Scrum) is not required for a PBD lecture. In this lecture, student collaboration is encouraged to provide the environment of a development team in Scrum, XP or pair programming. During an iteration, students can choose to work alone or as a team; there is no prescribed team size.

D. Events

A PBD lecture consists of three events that iteratively repeat during the lecture. An exercise iteration starts with sprint planning, followed by the actual development work and a review of the increment. During planning, the selected PBIs for one iteration are discussed. After all of the students' questions have been answered, the exercise time starts: the timeframe for one iteration is 10-20 min depending on the number and difficulty of selected PBIs. At the end of the given time period, the instructors present a possible solution; this corresponds to the review meeting in Scrum. The instructors indicate which pattern they intend to use to complete the PBIs and students may reflect on their own results. In addition to this solution inspection, students receive feedback from teaching assistants on their individual solutions. After the end of an iteration, the next iteration starts immediately with a new selection of PBIs.

IV. Case Study

We applied this approach for teaching PBD in two large university courses:

1) **Introduction to Software Engineering (EIST)**: mandatory course with 1400 bachelor students
2) **Patterns in Software Engineering (PSE)**: elective course with 500 master students

In both the courses a set of patterns has been introduced, taught and applied prior to the PBD lecture. We describe the two courses in more detail and then describe how the PBD lecture is conducted.

A. EIST

EIST is an introductory course to software engineering in which students learn to apply relevant concepts and methods in each phase of a software engineering project. The students have university-level knowledge of the most important terms and concepts in the software engineering domain. They are also aware of the problems and issues that generally must be considered in software engineering. A non-negligible part of the course is related to patterns. Several design, architecture and testing patterns are introduced over four lectures followed by a PBD lecture with the following iterations:

- **Iteration 1**: No Pattern Applied (IC)
- **Iteration 2**: Dealing with Generalization (IC)
- **Iteration 3**: Strategy Pattern (IC)
- **Iteration 4**: Observer Pattern (IC)
- **Iteration 5**: Adapter Pattern (HW)

B. PSE

We have been teaching PSE since 2008. Typically, 500 students register for the course even though it is an elective course. Students learn about the principles of patterns in software development and the structure of a pattern-based software system. We teach the students how to apply patterns to a variety of problems and how to deal with the patterns in concrete applications. The course covers patterns that can be applied throughout the software lifecycle: design, architectural, testing, and organizational patterns, as well as anti-patterns are taught. The PBD lecture was held after nine lectures and comprised five iterations that involved the following patterns:

- **Iteration 1**: No Pattern Applied (IC)
- **Iteration 2**: Observer Pattern (IC)
- **Iteration 3**: Abstract Factory Pattern (IC)
- **Iteration 4**: Adapter Pattern (IC)
- **Iteration 5**: Strategy Pattern (HW)

C. Lecture Structure & Problem Statement

We use Java as the programming language and use UML class diagrams for modeling. To stay within the scope of a lecture, we adapt and tailor the software lifecycle based on our needs and focus on the phase in which we want to apply an appropriate pattern. In the PBD lectures for both courses, no pattern is applied in the first iteration as we found that this gives the students an opportunity to familiarize themselves with the existing source code. Simple PBIs are chosen to help the students get started. The last iteration is assigned as a homework (HW) assignment rather than an in-class (IC) exercise and the results of this iteration are reviewed in the following lecture. For the distribution and assessment of the source code we use ArTEMiS, a platform for automated assessment of programming exercises in large classes that makes use of a version control system to track students' progress [10]. ArTEMiS enables the execution of structural, behavioral, runtime, performance, and functional tests to test the desired patterns.

Bumpers, a simple 2D game, in which a certain number of cars drive on a rectangular game board is chosen as the

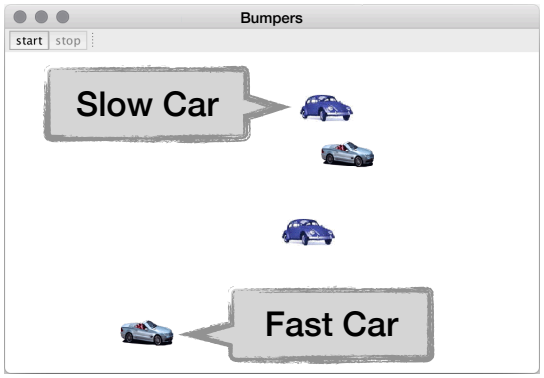

Fig. 4. Screenshot of the Bumpers game: the user interface after iteration 1. Fast cars are depicted as sports cars and slow cars are represented as classic cars.

problem. In this game, a car can be controlled by the player using a mouse. Some cars are fast while others are slow and the initial driving direction of each car is randomly determined. Cars collide with each other and a winner and a loser are determined in each crash; when a car collides with the border of the game board, it bounces back depending on its speed and direction according to the laws of physics. Figure 4 shows the initial user interface of Bumpers. The goal is to continuously improve the game and incrementally extend its functionality. To ensure that the timeframe of the lecture is not exceeded, a source code is provided for the students to work on.

V. Discussion

As we have applied this approach for teaching PBD several times, we discuss our experiences and share our findings and best practices with other instructors. By combining theoretical knowledge delivery with the immediate practical application of this knowledge, it is possible to attract students to attend the lectures. As with other major lectures, the number of students attending our lectures is declining but not as much as for traditional lectures that are not interactive. Students seem to be more motivated and recognize the added value of the PBD-based interactive courses. As additional motivation, we reward students who actively participate in the exercises. Students who submit submissions quickly or provide particularly good submissions are rewarded with small gifts such as gummy bears or gadgets (for example, bluetooth speakers and power banks). In our experience, this encourages the students to put their newly acquired knowledge into practice and deliver fast and high-quality results. Exercises must be carefully planned, set up and assessed. It should be noted that it takes more time and personnel to prepare and execute a PBD lecture than a traditional lecture. Another positive impact of applying PBD is that the students become familiar with the methods and processes of agile software development; specifically, students learn an adaption of Scrum and learn how to apply it. Changes do not pose a major obstacle and can be instantly integrated. When applying PBD in large-scale lectures, we can deal with students with different levels of apriori knowledge of students by designing exercises such that experienced students usually have to solve a challenging additional task in addition to the standard exercise and thereby ensure that all students can increase their knowledge during the course. In the lectures, we demand and promote cooperation between students and encourage students to use patterns as a language. The goal is that, e.g., when a student mentions the term “bridge”, all other students immediately know what the term means and have a common understanding of a possible solution to the design problem.

VI. Conclusion

In this paper, we describe the foundation for our approach to teaching PBD. We introduce patterns step-by-step in different phases and emphasize their application in a dedicated PBD lecture that is based on an adaptation of Scrum. This lecture is divided into short, recurring iterations each comprising planning, development work and review. We have applied this teaching approach in two different courses to demonstrate that it is feasible for students at different levels. Based on our experience, we have developed a best practice catalog for other instructors so that they can incorporate our method of teaching PBD into their own courses. In the future, we aim to conduct qualitative and quantitative evaluations. Further, we plan to integrate the teaching method into an online course. We expect to face and overcome various challenges in our work to make teaching patterns in software development more interactive and exciting.

References

[1] D. W. Shaffer, “Pedagogical praxis: The professions as models for postindustrial education,” *Teachers College Record*, vol. 106, no. 7, pp. 1401–1421, 2004.

[2] J. Whitehead, “Collaboration in software engineering: A roadmap,” in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 214–225.

[3] B. Bloom, M. Engelhart, E. Furst, W. Hill, and D. Krathwohl, “Taxonomy of Educational Objectives: The Classification of Educational Goals,” 1956.

[4] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge, “Interactive Learning: Increasing Student Participation Through Shorter Exercise Cycles,” in *Proceedings of the Nineteenth Australasian Computing Education Conference*. New York, NY, USA: ACM, 2017, pp. 17–26.

[5] S. Krusche, B. Bruegge, I. Camilleri, K. Krinkin, A. Seitz, and C. Wöbker, “Chaordic Learning: A Case Study,” in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, ser. ICSE-SEET ’17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 87–96.

[6] D. Hock, “The chaordic organization: Out of control and into order,” *World Business Academy Perspectives*, vol. 9, no. 1, pp. 5–18, 1995.

[7] S. Yacoub and H. Ammar, *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[8] L. Ackerman and C. Gonzalez, *Patterns-Based Engineering: Successfully Delivering Solutions via Patterns*, 1st ed. Addison-Wesley Professional, 2010.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[10] S. Krusche and A. Seitz, “ArTEMiS - An Automatic Assessment Management System for Interactive Learning,” in *49th Technical Symposium on Computer Science Education (SIGCSE)*. ACM, 2018.