Improving Students' Learning in Software Engineering Education through Multi-Level Assignments

CHRISTIAN KÖPPE
HAN University of Applied Sciences, Arnhem/Nijmegen, the Netherlands
and
LEO PRUIJT
HU University of Applied Sciences, Utrecht, the Netherlands

Assignments and exercises are an essential part of software engineering education. It usually requires a variety of these assignments to cover a desired wide range of educational objectives as defined in the revised Bloom's taxonomy. But such a variety has inherent problems, e.g. that students might not see the connections between the assignments and find it hard to generalize the covered concepts.

In this paper we present the educational design pattern MULTI-LEVEL ASSIGNMENT which addresses these problems. It enables the assignment designer to incorporate a variety of educational objectives into a single assignment by including the concepts on multiple knowledge and process levels. The description as educational design pattern and the provided three implementation examples make this approach directly applicable for other software engineering educators.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education —*Computer science education*

General Terms: Design, Education, Software Engineering Assignment

Additional Key Words and Phrases: Educational Patterns, Design Principles, Revised Bloom's Taxonomy

ACM Reference Format:

Christian Köppe and Leo Pruijt. 2014. Improving Students' Learning in Software Engineering Education through Multi-Level Assignments. Proceedings of Fourth Computer Science Education Research Conference, CSERC14, Berlin, Germany, 6 pages.

Author's addresses: christian.koppe@han.nl, leo.pruijt@hu.nl

CSERC '14, November 05 - 06 2014, Berlin, Germany
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3347-4/14/11 $15.00
http://dx.doi.org/10.1145/2691352.2691357

1. INTRODUCTION

Software Engineering (SE) comprises different disciplines and areas of knowledge like Software Requirements, Software Design, Software Architecture, Software Construction, or Quality Assurance. These are described in the Software Engineering Body of Knowledge (SWEBOK) [IEEE Computer Society 2004]. Software engineers need to have knowledge of these disciplines as this knowledge is essential for their professional performance. Academic SE programs intend to support students' acquisition of this knowledge. The desired level of knowledge hereby varies and is restricted by the educational setting of undergraduate or graduate programs due to limited available time for topic coverage, staff knowledge, institutional research areas, and other factors. Therefore, academic institutions have to make decisions on what to include in a curriculum and on what level to include it.

The coverage of the chosen knowledge areas is translated into a curriculum and the elements of the curriculum are mapped on educational objectives by curriculum- and course-designers. These objectives can be described according to the revised Bloom's taxonomy [Anderson and Krathwohl 2001], which is also applicable in computing education [Thompson et al. 2008]. The knowledge dimensions are *Factual*, *Conceptual*, *Procedural*, and *Metacognitive* and the cognitive process dimensions are *Remember*, *Understand*, *Apply*, *Analyze*, *Evaluate*, and *Create*. These knowledge and process dimensions form in combination a two-dimensional taxonomy table (see Table I for an example) that is helpful for designing and evaluating the content of courses and the included activities.

According to the curriculum objectives it is sometimes sufficient if the students just recall or recognize factual knowledge of some knowledge areas (rote or shallow learning). But as educators, we strive to engage the students in active cognitive processing of procedural or conceptual knowledge too (meaningful or deep learning) [Mayer 2002].

To incorporate a variety of cognitive processes in a course, different activities are used in the form of exercises and assignments. This "Learning Through Practice" [Laurillard 2012] often extends some more narrative approaches, where students have to acquire factual and basic conceptual knowledge through reading, listening, or watching. The assignments and exercises are related to specific educational objectives—the cells in the taxonomy table as described above. They also cover a variety of topics, e.g. a stopwatch, a music collection administration system, a board-game engine or an elevator system. In this paper we name these specific topics the *assignment domain*.

In our experience, there often is still a distinction between the factual, conceptual, and procedural knowledge and the assignments and exercises address varying but distinctive cognitive processes. Such assignments cover only sub-parts of the taxonomy and are therefore limited in their individual contribution to holistic meaningful learning. A mix of different assignments covering different aspects of learning helps with solving this problem, but requires many assignments. These would run in parallel or sequential, which makes it harder for the students to intuitively see the connections between the covered concepts. Another option for improving instruction is to raise the learning targets [Raths 2002], which could be realized by a broader coverage of cognitive processes and knowledge categories in a single assignment.

In this paper, we present a way for integrating a variety of cognitive processes and knowledge categories in a single assignment. The main idea is to include the concepts not only on the procedural level, important for the realization of the assignment or exercise, but to also include them on the conceptual level as assignment domain. Both authors have applied this approach multiple times, examples (which are described in more detail later) are a business rule generator, a software architecture compliance checking tool, and a design pattern selector.

For the description of the approach, we decided to use the format of an *educational design pattern*. This offers a few advantages compared to more traditional practice reports or theoretical descriptions [Laurillard 2012] and is a valid scientific format for formulating a theory about a concrete experience [Kohls and Panke 2009]. Christopher Alexander states that "Each pattern describes a problem which 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 twice" [Alexander et al. 1977]. Patterns have been adapted in the field of (CS) education for nearly 20 years now, and many educational design patterns have been documented and proven to be valuable (see e.g. [Goodyear and Retalis 2010; Köppe 2011; 2013; Larson et al. 2008; Laurillard 2012; Pedagogical Patterns Editorial Board 2012]). Our work extends this body of knowledge by proposing the MULTI-LEVEL ASSIGNMENT pattern.

The rest of the paper is organized as follows: In Section 2 we describe the background of the pattern and its mining. The pattern format and the pattern itself are presented in Section 3. This is followed by a discussion of the described approach and related work (Section 4). The final section concludes the paper.

2. PATTERN SOURCE AND MINING

Design patterns are usually discovered in the process of pattern mining, the main task hereby is to "expose the invariant structure and discriminate it from the surface structure (the non-essential features)" [Kohls and Panke 2009]. There are different approaches to pattern mining, whereby inductive inference—the observation and analysis of good examples [Alexander 1979]—is probably the most common one and typical for qualitative research [Kohls and Panke 2009]. This approach was also applied in this work.

Two student assignments were designed for 2nd year and 3rd year students of an undergraduate computer science program with a focus on software engineering at the HU University of Applied Sciences. They lasted 10 and 20 weeks, respectively. Both assignments will be shortly introduced now and described in more detail in the examples sections of the pattern description.

The first assignment was the design and realization of a *Business Rule Generator*, a tool for maintaining business rules and generating them towards different target languages, like Oracle PL/SQL (in the form of database triggers and functions). This tool also included business rules as part of the requirements.

The second assignment was an *Architecture Compliance Checking Tool*. This tool provides facilities to specify a logical architecture including elements like layers or components and rules related to them like "is not allowed to use". It then supports analyzing the source code of the system that is supposed to follow the defined architecture, mapping of the physical elements to the logical ones, and finally to check the compliance of the source code regarding the defined rules. The tool itself was developed by a group of 25 students, divided into 6 groups. This required a carefully designed software architecture, including elements, relations, and rules on these relations, many of these to be determined by the students themselves.

Both assignments had worked well according to the perception of the instructors and also the final student grades and student evaluations. Both had one aspect in common: the concepts the students had to learn and were required to apply in the assignments were also the domain of the application to be built—the students had to apply them on both procedural and conceptual level using different cognitive processes. This is the basic invariant structure—the good practice we wanted to document—and forms the base for the proposed pattern MULTI-LEVEL ASSIGNMENT.

To evaluate the quality of the pattern and to assess the degree of corroboration we applied the pattern for the design of a third assignment, a *Design Pattern Selector*. This assignment is described too as example in the pattern and the conscious application of MULTI-LEVEL ASSIGNMENT as educational *design* pattern is discussed in Section 4.

3. PATTERN DESCRIPTION

In the following sections we describe the identified pattern. We use an adapted version of the format as introduced by Christopher Alexander et al. [Alexander et al. 1977] and provide headings for all sections so that they can be easier identified. The description starts with the context, which explains in what conditions the described solution solves the problem. This is followed by the problem itself, whereby the core of the problem is highlighted in bold font. Next come the forces that shape and refine the problem. They help to deeper understand the nature of the problem and lead towards the solution, which is presented next. Again, bold font is used to highlight the core of the solution. The next section explains how the solution can be implemented, followed by detailed descriptions of three known uses/examples. The final section states the consequences of applying the pattern solution, both positive and negative.

Pattern: MULTI-LEVEL ASSIGNMENT

Context. You want to design an assignment that supports meaningful learning of specific software engineering concepts. You already introduced some basic factual, conceptual, and procedural knowledge on the concepts.

Problem. **Student learning is suboptimal when assignments can be completed by merely applying techniques and concepts without requiring a deeper conceptual understanding of them. In that case the students might learn less than they could have.**

The deep understanding of a concept requires knowledge and skills at different levels, but different levels often require different educational methods. The procedural level—students can apply the concepts—is often implemented by giving the students work assignments, e.g. programming a certain tool, gathering and documenting requirements, or defining the architecture of a complex system.

Forces. Students should apply the techniques they have to learn, but often do that through copying and pasting from existing solu-

	Cognitive Process					
Knowledge	1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create
A. Factual	x	x		(x)		
B. Conceptual	x	x		(x)	x	x
C. Procedural	x	x	x			(x)
D. Metacognitive						

Table I. : Mapping of the educational objectives of MULTI-LEVEL ASSIGNMENT to the revised Bloom's taxonomy [Anderson and Krathwohl 2001]

tions and adjusting these so that they fulfill the requirements. Even though the resulting solutions might be sufficient, this often does not require the desired deeper understanding of the concepts we want the students to learn.

Trial and error, an approach often used by students, can lead to sufficient results too, but does not necessarily require conscious thinking about and awareness of the applied concepts. If students also experience this trial and error approach as sufficient, they are not supported in gaining a deeper understanding of the concepts.

Assignments often require one concrete application of concepts, which makes it hard to generalize these concepts for the students.

Extending assignments with exercises that also explicitly cover cognitive processes regarding the conceptual knowledge leads to *parallel or sequential activities*, which makes it harder for the students to see the connection between the procedural and conceptual parts of the concepts.

Solution. **Therefore: Make the concepts the students have to study (part of) the assignment domain itself and not only part of the techniques they have to apply in order to realize the assignment.**

This way the students have to apply the concepts when realizing the assignment and also have to understand the concepts on a higher level as part of the application domain. This could be even more improved by having the students also analyze (parts of) this domain as part of the requirements gathering and make them using some realistic data for testing and demonstrating purposes.

The following educational objectives (including their corresponding cells in Table I) are tackled by such an assignment ("the students are able to"):

—Remember previously taught essential knowledge of all categories (A1 to C1).
—(optional) Analyze the domain of the assignment and use this as basis for the domain model (B4; also A4 if known examples are used as basis for the analysis).
—Interpret and classify the concepts in the domain model (B2) and use specific instances as sample data for the application (A2).
—Evaluate the domain model before it's used as basis for the design of the application (B5).
—Apply the procedural knowledge for the realization of the concepts in the assignment (C3), which requires an understanding of how to apply them (C2).
—Plan and optionally think of alternative ways of implementing the product and applying the concepts to be learned (C6).
—Implement the assignment which certainly requires the creation of some larger product that contains the concepts (B6).

The broad coverage of educational objectives shows that indeed a higher learning target can be reached by using MULTI-LEVEL ASSIGNMENT. According to Raths [Raths 2002] this can also be interpreted as evidence for improved instruction. Please note: Metacognitive knowledge and its related cognitive processes are not covered with multi-level assignments.

Implementation. One important part is to identify upfront the concepts you want the students to learn on different levels. Setup the assignment so that (a) the concepts (or the most important parts of them) are (part of) the domain of the assignment and have to be described as domain model and (b) that the students also have to apply the concepts on a procedural level. Making the concepts part of the domain implies that the functional requirements of the assignment are in line with how the concepts would be applied in "real" projects. In most cases this could be an administration tool for specific instances of the concepts, like a software requirements administration (that is itself defined by requirements) or an architecture administration/visualization tool (that has itself an architecture). Another option is a generation tool, that generates concrete instances of previously defined implementation-independent instances of concepts, e.g. a business rule generator.

As described in the solution part, the learning target can be broadened by adding the cognitive process of analyzing to the assignment. In that case require the students to analyze (parts of) the assignment domain—including the concepts to be learned—mostly by themselves as part of the functional requirements. You may provide an initial version as starting point, mainly for triggering the thought process and making the first steps easier.

The assessment of such an assignment is similar to that of assignments with other domains. Of main importance is that the domain model has to be correct and consistent, which means for a MULTI-LEVEL ASSIGNMENT that the conceptual knowledge is much more emphasized than in typical software engineering assignments.

Consequences. Even though the students still can copy&paste and adjust an existing solution when realizing the assignment, they are also forced to have to work on and with the concepts on a different—the conceptual—level.

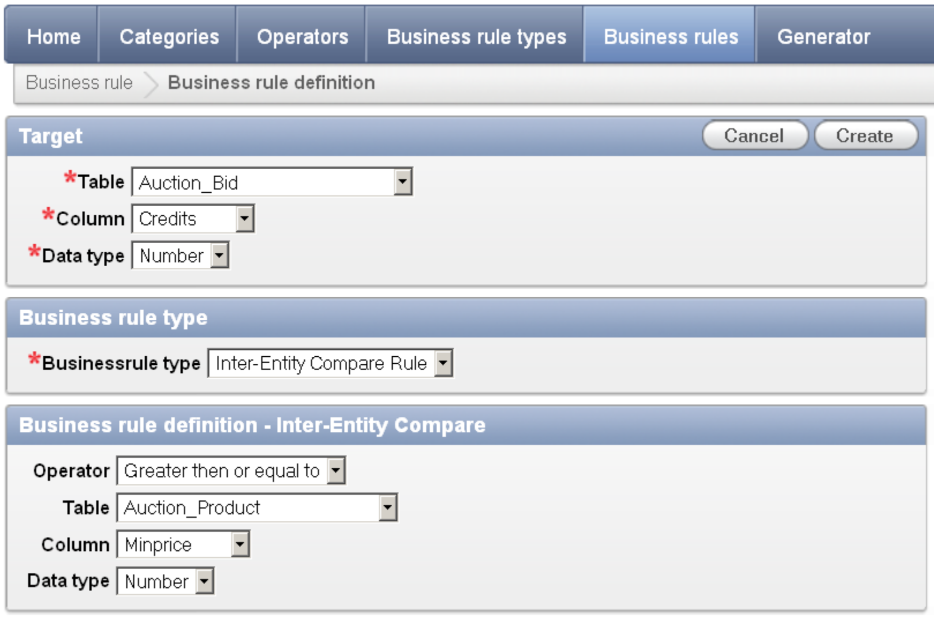
Applying trial and error might still work for the technical realization of the assignment, but it does not help with the conceptual component of the assignment: the definition and realization of the domain does not allow trial and error.

As the students are exposed to the concepts they are applying on a conceptual level too, it probably will be easier for them to recognize the generic aspects of them.

In some cases, like interdisciplinary assignments or when involving industry partners, pre-determining the assignment domain does not work as it does not fit the needs of the involved stakeholders. In such cases the pattern is not applicable.

There is also the chance that the students would favour another assignment domain which is closer to their interests, like e.g. a specific game or a gym administration.

Example: Business Rule Generator. In the second year of the bachelor program computer science at the HU University of Ap-



Target

*Table	Auction_Bid
*Column	Credits
*Data type	Number

Business rule type

*Businessrule type	Inter-Entity Compare Rule

Business rule definition - Inter-Entity Compare

Operator	Greater then or equal to
Table	Auction_Product
Column	Minprice
Data type	Number

Fig. 1: Student system example: Business rule definition (the selection of a specific business rule type implies specific business rules to be checked for this type, e.g. a limited set of available operators and allowed selections of attributes)

plied Sciences, we teach the students to recognize, specify, classify and implement business rules as part of a course on Software Requirements. The Business Rules Group [The Business Rules Group 2013] defines a business rule (from the information system perspective) as “a statement that defines and constraints some aspect of a business. It is intended to assert business structure, or to control or influence the behavior of the business.” In our program, we introduce the basics of business rules, but focus thereafter on data rules and how they are classified according to the BCDM framework [Boyd 2001].

The project assignment was developed in cooperation with a software development company specialized in Oracle Technology. It stimulates the students to think and work with business rules at both procedural and conceptual level. The students are instructed to design and build a business rule generator. Their resulting generator should allow a software developer to enter the functional details of a business rule via the GUI component of their application, which means that business rules are the assignment domain. On demand the rule should be transformed automatically, by the generator component, to a database constraint or database trigger in the SQL-variant of a pre-selected database system.

Functional requirements are provided to the students, including the details of nine types of rules and examples of the rule types with related code examples to implement the rule as database constraint or trigger. Before the students start building the generator, they have to design a conceptual model that answers the requirements. Furthermore, they need to specify and implement an accompanying set of business rules for their generator, constraining the data to be entered into their system—the procedural part of the pattern solution. By doing so, the students improve their analytical skills and develop a meta model of the business rule domain, which adds to their knowledge and is the result of the pattern application.

Figure 1 shows a screenshot of one of the student systems. All metadata relevant for specific business rule types have to be defined through the tool including business rules constraining them. These rules then have to be checked when entering specific business rules for later generation.

Example: Architecture Compliance Checking Tool. Software architecture is an important topic within our bachelor program computer science. At the end of the second year, preceded by several analysis and design modules, an introduction in software architecture is provided, and in the third year we offer a module Advanced Software Architecture and a related, large project assignment. The basics of software architecture, modular design, layers, components and architectural patterns are discussed at first, with the focus on the design of semantically rich modular architectures [Pruijt et al. 2013]. Next, to develop the practical skills, assignments are made in which the students classify different types of responsibility, design layers and components at the logical and physical level, and design application scenarios in accordance to the architectural models and/or in accordance to architectural patterns.

The third-year project assignment invites the students to think and work with software architecture at different levels. The assignment focuses on tools to support checks on architecture compliance; “a measure to which degree the implemented architecture in the source code conforms to the planned software architecture” [Knodel and Popescu 2007]. Previous to the assignment, the students have to identify requirements to this type of tools and use and test different tools. Next, the students work on the design and development of a new open source software architecture compliance checking tool. A domain model is used to identify and relate the most important concepts of architecture compliance including different software architecture elements, their relations and rules applying on these relations. Furthermore, a component model is used to identify and relate the functional parts of the tools, to allocate the domain concepts to these parts, and to divide the work over (up to six) student teams. Each team is responsible for one component and the teams have to negotiate the requirements with regard to the provided services, as well as the implementation details. That way the students had to apply the same software architecture concepts for building the tool that also are part of the domain of the tool, which is the core of the pattern.

At the end of the semester, one-day architecture compliance checks were organized at the sites of software development organizations. On such a day, a professional application was analyzed with the help of the tool, and checks were performed on the conformance to the software architecture. At the end the students had to present suggestions for improvement regarding the actual realization of the software architecture in the analyzed applications. These suggestions were well perceived by the present software architects of the companies and demonstrated a high conceptual understanding of the architectural concepts *and* their realizations in applications by the students. We interpret this as indication that the pattern application indeed was successful.

Example: Design Pattern Selector. We have applied the pattern for the design of an assignment for the course "Patterns & Frameworks", part of the second year of an undergraduate computer science program with a focus on software engineering. The main requirement for the assignment was to implement a pattern selector and a pattern editor in Java. The students therefore had to apply design patterns for the design of the tool while the (same) design patterns also formed the domain of the tool. This reflects the core of the pattern solution.

In that example we went through all educational objectives as described in the pattern solution and how they were included in the assignment. This helped us to consciously check the coverage of objectives and demonstrates the practical support the pattern solution offers for assignment design. Based on this check we decided to add some more focus on specific educational objectives, namely understanding and analyzing conceptual knowledge. We did so by requiring that the tool should be usable for all domains of design patterns (instead of only the GoF-patterns as presented in [Gamma et al. 1994]) through the usage of generalized pattern description sections (context, problem, forces, etc. instead of intent, motivation, etc.). The assignment furthermore included that the students had to fill their repository with at least ten GoF-patterns. The students hereby had to interpret and classify the pattern parts by themselves and to organize these parts in the given structure.

A set of extra functional and non-functional requirements was given that was related to typical problems addressed by specific GoF design patterns. An example is that one could add patterns on-the-fly while leaving the selection user interface open, and that the selection lists automatically are updated after adding a new pattern *or* a new context (solvable with the OBSERVER pattern). Another example is that the repository of patterns that one builds up should be exportable and importable in different formats, and these formats should be easy to add (solvable with the FACTORY METHOD and STRATEGY patterns).

Finally, the requirements included that the tool explicitly has to support the method of how patterns should be selected prior to applying their solution, which is an application of the educational pattern CONTEXT, PROBLEM, AND CONSEQUENCES FIRST [Köppe 2013]. One should first check if the context the pattern describes and the design context are sufficiently matching. Then the students have to analyze what the actual design problem is that they want to solve. Next, the problem that needs to be solved should be matched with the problems that can occur in the selected context (a list generated from the pattern descriptions). Finally, *before* presenting the pattern solution, one should first carefully look at the consequences the application of the pattern solution has to trade off benefits against liabilities.

4. DISCUSSION

An important part of being "Reflective Practitioners" comprises the search for improvements of our own teaching, and the form of educational design patterns offers a valid way of doing so [Laurillard 2012]. Describing our approach as educational design pattern helped us to consciously reflect on and think of the different aspects of the approach in a structured and detailed way.

The described pattern was interpreted as hypothesis and we evaluated the pattern by applying it a for a third time (see the third example in the pattern description). This conscious application raised the understanding of the concepts, in that example the understanding of design patterns and how to use them correctly, by addressing the concepts on different knowledge levels and requiring a variety of cognitive processes as described in the pattern solution. The good students' results of the assignments (not presented in this paper), both for the domain model parts and the realization parts of the assignments, indicate that the proposed pattern works. This is also supported by the students' evaluations of the courses in general and specific feedback on the assignments

There are other implementation instances possible, like gathering requirements for a requirements gathering tool, following a certain software process to develop a process support tool (where one can define roles, responsibilities, tasks, etc.), or to build and test a testing support tool.

Related Work

Guided Exploration [Köppe and Rodin 2013] is another way of addressing the issue that students use procedural knowledge without understanding the underlying concepts. The assignment description hereby consists of descriptions of the concepts the students have to apply in order to successfully finish the assignment instead of focusing on the required functionality of the product.

Some already described pedagogical patterns are related to MULTI-LEVEL ASSIGNMENT. ABSTRACTION GRAVITY - FROM HIGH TO LOW [Pedagogical Patterns Editorial Board 2012] addresses the issue of presenting concepts at different abstraction levels. It could be used for presenting the factual knowledge required for MULTI-LEVEL ASSIGNMENTS. EXPERIENCING IN THE TINY, SMALL, AND LARGE [Pedagogical Patterns Editorial Board 2012] adresses learning of concepts on different levels, but requires multiple steps (assignments).

Choosing exercises that cover simultaneously multiple ideas or topics—as opposed to multiple abstraction levels of the same concepts—is the core of a MULTI-PRONGED ATTACK [Pedagogical Patterns Editorial Board 2012]. Similar to this is ONE CONCEPT - SEVERAL IMPLEMENTATIONS [Pedagogical Patterns Editorial Board 2012] which emphasizes the use of different implementations of an abstract concept as examples.

5. CONCLUSION

In this paper we reported on a common issue in the design of assignments: the distinction between acquisition and application of procedural and conceptual knowledge. To address this issue, we proposed the educational design pattern MULTI-LEVEL ASSIGNMENT, which was mined from two successful assignment designs. The variety of educational objectives—as shown in the mapping to the revised Bloom's taxonomy—addressed in one assignment, increases the number of learning targets and improves students' learning.

Using the pattern format helped us to explore all relevant parts of the assignment design in more detail and describing it in a reusable way. We—as reflective practitioners—experienced this as helpful. We applied this pattern for designing a third assignment and evaluated its applicability and added value through conscious application as *design* pattern. The result was satisfying: the pattern provided good guidance for the design and observations of discussions between students who worked on the assignment showed that the students were indeed thinking and talking about the concepts at both conceptual *and* procedural level. This was similar to the results of the first two assignments where this pattern was applied. This observation is based on the teachers perceptions and still lacks a sufficient evaluation.

As stated earlier, we plan to apply MULTI-LEVEL ASSIGNMENTS in other courses too. This promotes our striving for supporting meaningful learning of our students and will be part of future work. We also encourage other educators to apply this pattern and to share their experiences with us.

Acknowledgements

We mostly want to thank all students who took the assignments based on the described design pattern and helped us to improve it. We also thank all reviewers of earlier versions of this work for their valuable feedback.

REFERENCES

Christopher Alexander. 1979. *The Timeless Way of Building* (later prin ed.). Oxford University Press, New York.

Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Lorin W Anderson and David R Krathwohl. 2001. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman, Inc, New York. 352 pages. DOI:http://dx.doi.org/10.1207/s15430421tip4104_2

L.L. Boyd. 2001. BCDM RuleFrame - the business rule implementation that saves you work. In *ODTUG Business Rules Symposium*. Oracle Corporation, iDevelopment Center of Excellence.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, Boston, MA.

Peter Goodyear and Simeon Retalis (Eds.). 2010. *Technology-Enhanced Learning: Design Patterns and Pattern Languages*. Sense Publishers. 330 pages. http://www.amazon.com/Technology-Enhanced-Learning-Patterns-Pattern-Languages/dp/9460910602

IEEE Computer Society. 2004. *Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society. http://www.swebok.org/

Jens Knodel and Daniel Popescu. 2007. A Comparison of Static Architecture Compliance Checking Approaches. In *Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. IEEE, 12–21. DOI:http://dx.doi.org/10.1109/WICSA.2007.1

Christian Kohls and Stefanie Panke. 2009. Is that true...? - Thoughts on the epistemology of patterns. In *Proceedings of the 16th Conference on Pattern Languages of Programs - PLoP '09*. ACM Press, New York, New York, USA. DOI:http://dx.doi.org/10.1145/1943226.1943237

Christian Köppe. 2011. Continuous Activity - A Pedagogical Pattern for Active Learning. In *Proceedings of the 16th European Conference on Pattern Languages of Programs - EuroPLoP '11*, Vol. 2011. ACM Press, Irsee, Germany. DOI:http://dx.doi.org/10.1145/2396716.2396719

Christian Köppe. 2013. A Pattern Language for Teaching Design Patterns. *Transactions on Pattern Languages of Programs* 3 (2013), 24–54. DOI:http://dx.doi.org/10.1007/978-3-642-38676-3_2

Christian Köppe and Rick Rodin. 2013. Guided Exploration: An Inductive Minimalist Approach for Teaching Tool-related Concepts and Techniques. In *Proceedings of the 3rd Computer Science Education Research Conference, CSERC'13*. ACM, Arnhem, Netherlands. http://koeppe.nl/publications/CSERC2013_Koppe.pdf

Kathleen A. Larson, Frances P. Trees, and Scott D. Weaver. 2008. Continuous feedback pedagogical patterns. In *Proceedings of the 15th Conference on Pattern Languages of Programs - PLoP '08*. ACM Press, New York, New York, USA. DOI:http://dx.doi.org/10.1145/1753196.1753211

Diana Laurillard. 2012. *Teaching as a Design Science: Building Pedagogical Patterns for Learning and Technology.* Routledge, Oxon, UK. http://www.eric.ed.gov/ERICWebPortal/recordDetail?accno=ED529967

Richard E. Mayer. 2002. Rote Versus Meaningful Learning. *Theory Into Practice* 41, 4 (2002), 226–232.

Pedagogical Patterns Editorial Board. 2012. *Pedagogical Patterns: Advice for Educators*. Joseph Bergin Software Tools, New York, NY, USA. 230 pages.

Leo Pruijt, Christian Köppe, and Sjaak Brinkkemper. 2013. Architecture Compliance Checking of Semantically Rich Modular Architectures: A Comparison of Tool Support. In *Proceedings of the 29th International Conference on Software Maintenance, ICSM'13*. IEEE Computer Society Press, 220–229. DOI:http://dx.doi.org/10.1109/ICSM.2013.33

James Raths. 2002. Improving Instruction. *Theory Into Practice* 41, 4 (2002).

The Business Rules Group. 2013. Defining Business Rules. (2013). http://www.businessrulesgroup.org

Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. 2008. Bloom's taxonomy for CS assessment. In *Proceedings of the tenth conference on Australasian computing education*. Australian Computer Society, Inc., 155–161. http://dl.acm.org/citation.cfm?id=1379249.1379265