Serdica J. Computing **2** (2008), 277–294

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

PATTERNS FOR ACTIVE E-LEARNING IN CMS ENVIRONMENTS

Atanas Radenski

Abstract. The proliferation of course management systems (CMS) in the last decade stimulated educators in establishing novel active e-learning practices. Only a few of these practices, however, have been systematically described and published as *pedagogic patterns*. The lack of formal patterns is an obstacle to the systematic reuse of beneficial active e-learning experiences. This paper aims to partially fill the void by offering a collection of active e-learning patterns that are derived from our continuous course design experience in standard CMS environments, such as Moodle and Blackboard. Our technical focus is on active e-learning patterns that can boost student interest in computing-related fields and increase student enrolment in computing-related courses. Members of the international e-learning community can benefit from active e-learning patterns by applying them in the design of new CMS-based courses – in computing and other technical fields.

ACM Computing Classification System (1998): K.3.2.
Key words: Course management systems; active learning; pedagogical patterns.

1. Introduction.

1.1. Objective. In computer science, the purpose of a *pattern* is to capture, disseminate, and promote successful practice in some domain, such as software design, human-computer interaction, and pedagogy. Pedagogical patterns in particular capture successful experience in the practice of course design and delivery.

During the last four years, we have created digital study packs that promote computer-supported active learning through experimentation and self-discovery. These online study packs have been employed in core computer science courses taught at Chapman University in California USA.

Positive student feedback from students and increased enrolment are a testimony for the success of our active e-learning packs [20, 26]. The *technical objective* of this publication is to present a collection of active e-learning patterns in CMS environments. This pattern collection is founded on our online study pack design and delivery experience and the experience of instructors who have adopted and used the author's online study packs. While we recognize that active e-learning patterns can be based on a wide variety of online and offline software tools, our focus is exclusively on patterns that work in standard CMS environments, because of their wide proliferation and also because our long-lasting experience is mainly with CMS.

1.2. Background Work and Experience. Active e-learning in computing courses has been in the focus of our teaching and research since 2004. We have developed the following comprehensive online study packs to support various lower and upper level undergraduate courses: *Introduction to Computing with Python, Object-Oriented Computing with Java, Compiler Construction, Programming Languages, Java Generics*, and others. Our online study packs have been published online on a dedicated server, `studypack.com`, and already used in actual courses of study by over 600 hundred of learners in the US and abroad at Chapman University in California USA, Columbus Sate University and Berry College in Georgia USA, College of Southern Nevada USA, University of Wisconsin Colleges USA, Sofia University in Bulgaria, the National University of Lesotho, and elsewhere. The underlying active e-learning approaches have been reported at selective professional meetings in the USA and Europe [20, 21]. This approach has already attracted interested computing faculty to a SIGCSE 2008 workshop, an ITiCSE 2008 demo, and to three annual workshops at Chapman University. We have already accumulated considerable experience with the design and use of online study packs and this practical experience is the founda-

tion of our proposed collection of synthesized, general-purpose patterns for active e-learning.

1.3. Methodology Overview. Design patterns were originally introduced in the 1970s as an architectural qualitative method by Christopher Alexander and his colleagues [1]. In mid-1990s, Erich Gamma and others [11] successfully adapted Alexander's pattern methodology to the domain of computer software design. We follow other educators [2] in the use a modernized version of the qualitative Alexandrian method.

This publication presents the outcomes of two tasks: pattern identification and pattern specification. Our *pattern identification* has been rooted in successful - as testified by surveys and enrolment increase numbers – online study pack practices, such as interactive labs, non-interactive labs, online quizzes, e-texts, and more. Our *pattern specification* is in the form of texts with uniform structure; we use a modified Alexandrian pattern structure that has already been successfully used in pedagogical pattern design [6, 19].

In our pedagogical pattern design, we follow recent guidelines [23] to find a working balance between abstraction and concreteness and to design patterns that serve as working tools – rather than general abstractions – for active e-learning course design.

Our design is intentionally restricted to active e-learning patterns that can be used in standard CMS environments – such as Moodle and Blackboard – because of the growing use of such standard environments as opposed to more exotic experimental systems, and because our own design and teaching experience is predominantly with such standard CMS environments.

2. Related Work.

2.1.Pedagogical Patterns Overview. Since the mid 1990's, *patterns* have become a central theme in the computing community.

As of the time of writing of this publication, a Google Scholar advanced search for publications on patterns and pattern languages in computer science, engineering, and mathematics produces an impressive list of 9110 publications which refer to pattern language(s) in their titles. Patterns have been in the focus of well-established international conferences, such as the prestigious OOPSLA and ECOOP conferences, the specialized PLoP conferences (organized since 1994 and published in comprehensive volumes [15]), and a number of regional conferences, such as EuroPLoP, KoalaPLoP, and VikingPLoP. Patterns related to education are frequently in the program of ACM SIGCSE and ACM ITiCSE conferences.

The Hillside Group's website [13] offers a useful collection of references on software patterns, including hundreds of articles, papers, and a list of nearly 70 books on software patterns that are currently available on the book market.

The Pedagogical Patterns Project [19] has produced a broad collection of pedagogical patterns [4, 5, 6, 7]. These general pedagogy patterns are focused on more traditional classroom environments and are not particularly dedicated to e-learning environments and techniques neither they address computer science learning specifics. Except for some patterns that address the use of email in course delivery, the PPP patterns do not particularly cover the e-learning domain.

Dearden and Finlay [10] offer a beneficial analysis of patterns and pattern languages in general and also focus on patterns for human-computer interaction in particular. A state of the art review of pedagogic patterns in computer science is compiled in the recent technical report of the ITiCSE 2008 WG1 on *Design Patterns for Online Learning Environments in CS* [25].

2.2. Active Learning Ideas and Practices. *"Active learning* styles deviate from traditional lectures and reading and involve learning by doing (physical action) and by thinking about what has been done (mental action). Active learning techniques are well supported by technology and are successfully applied in both core and in advanced computing courses" [20]. *Cooperative learning* is active learning in a group [16].

Roschelle et al. [24] summarize the benefits of the ancient active learning method offered by Socrates to his students. Active learning is characterized by (1) learners' active engagement, (2) a focus on knowledge construction, (3) timely feedback and adaptive instruction, and (4) learning community formation. The authors highlight the strengths of tablet PCs as active learning media, particularly through the use of platforms such as Classroom Presenter and Group Scribbles.

Razmov and Anderson [22] describe positive active learning experiences with tablet PCs in software engineering classes. In particular, the authors promote *student submissions*, "a style of interaction whereby the instructor poses a question ... on a tablet in front of each student" then students write in digital ink their answers back to the instructor, possibly with immediate feedback from the instructor. Student submissions help "engage all students, not only the vocal ones".

Budd [8] recalls another ancient active learning method, the one promoted by Plato in his book *Meno*. In active learning, the role of the teacher is "not only to present new information, but also to lead the student to understand what it is they already know". Budd describes a specific implementation, the

daily worksheets, of this idea in his data structures course. The author reduces traditional daily lectures to not more than 30 minutes in order to introduce a new concept. Students then spend the rest of the class analyze code that implements the concept, search textbook and other resources in order to fill answers into a worksheet of questions and problems.

Radenski [20] claims that "students who grew up browsing the Web are skilled in what is usually referred to as abduction, a reasoning process that starts with a set of specific observations and then generates the best possible explanation of those observations". In order to exploit the abduction skills of contemporary students, the author has developed digital CS1/2 study packs that promote and support active learning through abduction, i.e., *abductive learning*.

Bailey and Forbes [3] stress the importance of interaction in active learning courses. The authors promote the *feedback loop*, a communication method to create synergy between (1) web-enhanced out-of-class activities and (2) active learning classroom activities. In brief, students complete web-based warm-up assignments that prepare them for the upcoming lecture and lab. Warm-up submissions guide the instructor for best in-class delivery. Finally, the feedback loop is closed by online submissions of graded assessment exercises on material covered in class.

Gonzalez [12] advocates active and cooperative learning style that goes beyond mere course delivery to address management and assessment. The author describes successful experience with cooperative CS1 learning that is characterized by positive interdependence and face to face interaction between students and also by individual accountability.

According to Pollard and Duvall [18], it is beneficial to incorporate games and prizes in undergraduate computer science courses. The authors advocate – and present experience with – teaching techniques that are not computerized but are in fact "reminiscent of kindergarten: games, toys, stories, and play".

Valino [29] introduces active and cooperative learning techniques in software design patterns course. The study of a particular group of related patterns commences with a detailed list of learning objectives, continues with a set of questions related to the patterns, and ends with a design and implementation exercise.

2.3. Active E-Learning Patterns Overview. Patterns for active learning have been published as early as in 1995. Anthony [2] offers simulation games pattern and quiz games pattern as part of his general purpose patterns. Later, Bergin and others [5, 6] have collected patterns for experimental learning and patterns for active learning at various levels in various disciplines. These

general patterns are focused on more traditional classroom environment and are not particularly dedicated to e-learning environments and techniques neither they address computer science learning specifics.

Warren [30] employs several active learning ideas to teach software patterns in his software design course. A key idea is that "students should be able to *do* design by making effective use of design patterns". The author's *pattern for teaching software patterns* consists of these steps" (1) immerse students in a real problem, (2) present code that solves the problem, (3) introduce a software patterns that can in fact has been used in the solution design, (4) engage students in applying the patterns on a new, small problem, and (5) reflect on the pattern. This interesting pattern is not directly oriented to e-learning setting.

When it comes to patterns for (a) active learning in (b) e-learning settings, we feel that there is a misfortunate void in the literature. To help at least partially fill this void, this publication offers a collection of active e-learning patterns that is specifically oriented towards digital course design and delivery – as supported by standard CMS environments.

3. Collection of Active E-Learning Patterns.

3.1. Pattern Identification. Since 2001 we have used Blackboard to develop, use, and experiment with eight digital courses at Chapman University. Since 2004, we have developed an independent Moodle [9] installation for the same purpose and hosted at `studypack.com`. This independent Moodle installation has permitted the use of five new online courses by students and instructors from outside Chapman University as well (see Section 1.2 for a partial list of users).

Our Blackboard and Moodle experiences have revealed a number of beneficial e-learning practices that potentially can be formalized as pedagogical patterns. These practices include the use of integrated *online study packs*, *self-guided interactive labs*, *self-guided programming labs*, *honor reports*, *multiple attempt quizzes*, digital textbooks – referred to as *e-texts*, *email* for assignment distribution and evaluation, *forums* for individualized assignment submission and discussion, *glossaries* for Q & A types of documents, and *messaging* for student assistance and consultation. While all these are beneficial e-learning practices, we chose to systematically specify only the first five as pedagogical patterns. The five selected practices, such as self-guided labs for example, directly support abductive learning, which is a beneficial active e-learning method [20]. Useful practices that we have left outside of our pattern collection – such as email and forum uses – are well known and their publication as patterns would not be a significant contribution.

3.2. Pattern Specification. This section offers patters that we have discovered in our online study pack design and utilization (see Section 1.2 for details). While our experience involves study packs for lower and upper level courses, these particular patterns apply primarily in the context of introductory computing courses. The patterns also apply, we believe, to introductory courses outside of the computing field, particularly courses in which computer-based experimentation is plausible. These patterns are meant to be used in either hybrid courses that combine structured class meetings with asynchronous web-based learning, or in purely online courses.

Our patterns specifications consist of six sections: *Context, Constraints and Forces, Solution, Resulting Context, Utilization,* and *Related Patterns*. Each pattern begins with a brief *Context* section which formulates a pedagogic problem (or problems) that the pattern can resolve. The *Constraints and Forces* section of a pattern supplies detailed restatement of the problem(s) and helps determine when to apply the pattern. The *Solution* section defines the pattern's solution. The *Resulting Context* section points to possible benefits of the pattern's utilization. The *Possible Uses* section points to plausible areas of application for this pattern. The *Related Patterns* section, if present, may include references to other patterns in this paper and/or patterns published by others.

3.2.1. Online Study Pack.

Problem:

How can you actively engage students who are restricted by their extremely busy work and study schedules?

Constraints and Forces:

In developed and developing countries alike, more and more full-time undergraduate students now rely on paid work as their main or sole source of income. Australian full-time undergraduate students, for example, work an average of around 15 hours per week, according to McInnis and Hartley [17], and 18% work 21 hours or more per week. Students engage in paid employment not only to provide needed or desired income, but also to accumulate competitive professional experience before their graduation. Apart from paid employment, undergraduate students engage in overloaded course schedules for the purpose of timely graduation, especially at university with high tuition and other costs.

The need to balance busy work and course schedules puts significant strains on most undergraduate students and can be detrimental to their academic performance and their well-being. To help alleviate this problem, e-learning technology in general, and CMS environments in particular, can be used to liberate

students from the time and location constraints by actively engaging student in asynchronous learning activities

Solution:

Use a standard CMS environment to develop a comprehensive online study pack which integrates all possible resources and activities that are needed to teach and study an undergraduate course.

A study pack is a complete collection of digital resources (such as e-texts/digital books/tutorials, lecture slides, lab manuals, sample programs) and activities (such as quizzes, lab assignments, homework, exams, forums, chats, and messaging). An online study pack is pre-programmed with all deadlines and is made available to students in its entirety in the very beginning of any course of study. A study pack template can be instantiated to support various course sections at different schools.

In short, a study pack is an all-in-one e-learning solution that is constantly available to students and students, independently of time and location.

Resulting Context:

An online study pack permits busy students to actively engage in learning activities not when they are told by the instructor to do so, but when they have the time and the desire to do so (for example, some students choose to work at night while others chose to work early morning). In addition, students chose what exactly to do in a particular work session (for example, one student may choose to do a late-night quiz while another student may prefer working on a programming lab at the same time).

An instructor who manages an online study pack directs the learning process largely behind the scenes programming the study pack before the course and then facilitating students during the course. Throughout the course, the study pack offers students substantial freedom of what exactly to do and when to do it. Doing by own choice motivates students to actively engage in the learning process and do more.

Choice of alternative pathways in has been widely recognized as an indispensable e-learning feature and has therefore been placed in the focus of major research and development funded projects, such as EU's TEN Competence [28].

Possible Uses:

The development of a comprehensive study pack is time consuming and therefore is most justified for mass undergraduate courses. A well-designed study pack can be used as a template which is instantiated for easy reuse by various instructors.

Related Patterns:

Self-Guided Programming Lab, Interactive Lab, Honor Report, Multiple-Attempt Quiz

3.2.2. Self-Guided Interactive Lab.

Problem:

How can you engage students to actively learn a large number of lower-level technical concepts that are considered boring-to-read about?

Constraints and Forces:

Contemporary technical disciplines – even those of introductory nature – require learning about a voluminous set of low-level technical terms. Consider for example introductory computing courses. Typically, such courses evolve around the study of mainstream programming language, such as Java, C++, Visual Basic, Python, and others. These languages require acquaintance with and understanding of a very large set of lower-level technical concepts. Consider, for example the Java 2 platform which contains around 200 packages of classes in its standard edition [27]. The *java.lang* package alone incorporates 36 classes (it also includes 8 interfaces and relates to 36 exception classes, and 22 error classes). The String class, arguably on of the must-know classes from the *java.lang* package, contains over 60 methods and 16 constructors. Getting to know this fundamental class alone can be a daunting experience to any learner. It is certainly understandable reading about such a vast number of technical concepts – as in the case of Java methods and classes – can seem uninspiring and even dreary. It comes to no surprise that students often tend to evade technical reading that involves numerous lower-level concepts.

To alleviate this problem, passive reading can and should be partly replaced by active learning techniques. In particular, students can be engaged in technical concept study through systematically designed self-guided lab experiments.

Solution:

Use detailed interactive labs that systematically guide students through experimentation with various technical concepts.

An interactive lab is a detailed collection of step-by-step instructions that guide students through systematic experiments, then for observation, and for analysis of the observed results. For example, a self-guided computing lab may tell the student to type individual commands. As the student types each command, she observes and analyses the result, and discovers plausible explanations of he

semantics of the command. In the process, the student may consult appropriate reading resources that facilitate the student's comprehension.

A self-guided interactive lab should be embedded within a comprehensive study pack. While working on the lab, students gain straightforward access to supportive pack resources, such as tutorials and e-texts. Upon lab completion, students receive instant credit through an honor lab report.

In short, a self-guided interactive lab guides students in active self-discovery through experimentation, observation; such lab can also trigger reading and research if and when more information is needed by the learner.

Resulting Context:

An interactive lab stimulates students to actively engage in learning trough experimentation and self-discovery instead of monotonous reading of voluminous technical texts. Yet student may still read – not when they are told by the instructor to do so, but when they are curious to find better and clearer explanations of their interactive lab observations.

A properly designed self-guided interactive lab is carried out by students independently from the instructor. Students feel that they are the active participant in the lab, not the instructor. The feeling of self-discovery and the feeling of being in control stimulate students to do more and learn more.

Possible Use:

A self-guided interactive lab can be used in introductory courses in computing, mathematics, sciences, and possibly other disciplines that permit software-based interactive experiments.

Related Patterns:

Online Study Pack, Honor Report

3.2.3. Self-Guided Programming Lab.

Problem:

How can you actively engage beginner programmers in learning general and somewhat abstract programming methodology?

Constraints and Forces:

Programming methodology deals with the analysis, design and implementation of programs. This involves a variety of general methods, such as stepwise refinement, top-down design and implementation, bottom-up design and implementation, and agile programming, to mention a few. While these methods are valuable for any advanced professional, they are difficult to comprehend by beginners. For example, stepwise refinement seems to be a method that most

beginners seem to accept on theory but can never actually apply in their early practice. Standard software development methods are too abstract and without specific meaning to novices who lack experience.

To alleviate this problem, theoretical study of introductory programming methodology needs to be partly replaced by active learning techniques. In particular, students can be engaged in intuitive hands-on study of core programming methods through systematically designed self-guided programming labs.

Solution:

Use detailed self-guided programming labs that systematically guide students in the development of concrete program by means of a chosen programming methodology. For example, a particular lab may guide students of how to use stepwise refinement in the development of a GUI.

A self-guided lab supports three programming modes that suit students with different backgrounds. First, inexperienced students may follow complete and detailed prescriptions of what to do and how to do it in or. Second, experienced and motivated students may acquaint themselves with the lab specification and then develop the required software by themselves. A third category of students may try to find a solution independently while peeking into detailed instructions when help is needed.

A self-guided programming lab does not need to contain excessive explanations of its underlying methodology; it has to simply lead students the methodology for the development of a specific program. The methodology itself can be explained din e-texts and lectures and practically comprehended in labs.

A self-guided programming lab should be embedded within a comprehensive study pack. While working on the lab, students gain straightforward access to supportive pack resources, such as programming tutorials and reference sources. Upon lab completion, students receive instant credit through an honor lab report.

A self-guided programming lab guides students in active hands-on acquaintance with abstract programming methodologies - through experimentation and observation, and in supplement to reading and lectures.

Resulting Context:

A self-guided programming lab permits students to acquaint themselves with advanced programming methodology, through hands-on activities and self-discovery. Such labs work in concert with other forms of learning, such as lectures and textbook reading.

A self-guided programming lab is designed to be carried out by some students independently from the instructor. Like with interactive labs, the feeling of

being in control motivates students to do and learn more. However, programming labs are more complex than interactive labs and help in should be available in various lab activities, such as debugging and testing.

Possible Use:

A self-guided programming lab can be used in introductory courses in computing.

Related Patterns:

Online Study Pack, Honor Report

3.2.4. Honor Report.

Problem:

How can you grade a very large number of required interactive and programming labs with a reasonable effort and within a short time span?

Constraints and Forces:

The importance of hands-on lab work in computing and the natural sciences has been widely recognized. In introductory computing courses in particular, programming lab assignments are usually present throughout the entire course of study. While lab assignments support active learning they pose significant challenges to the instructor with the amount of effort and time that are needed to grade them. Consider for example an introductory class in computing which consists of 30 students who are required to submit 25 programming exercises each; if the instructor spends 10 minutes per exercise, she will spend 125 hours (more than 15 full workdays) of her time for grading alone.

Extensive grading of labs consumes a significant portion of instructor's professional time and also deprives students from the timely receipt of deserved lab credit. The whole process can be quite detrimental to both instructor's and student's motivation.

E-learning technology in general and CMS environments in particular can be used to (1) reduce instructors' grading burden and at the same (2) provide stimulating instant credit to students upon completion of required lab work.

Solution:

Use a CMS environment to implement the following honor lab report system. Upon the completion of each lab assignments, students are required (1) to upload their solution and (2) to file an online lab report. In the report, students specify all completed and partially completed labs. Provisional lab credit is awarded automatically once the report is filed. Submitted labs and reports are subject to audit by the instructor and provisional lab credits can be reduced by

the instructor. Credit reduction penalties apply when students report lab work that actually had not been performed adequately.

Such honor report system can be implemented with standard CMS functionality, such as file upload facilities (for completed work) and multiple-choice single answer questions (for the honor report itself).

Honor reports are intended to be integral parts of comprehensive study packs, together with their corresponding labs.

Resulting Context:

Honor reports significantly reduce the time and effort spent by instructors to grade labs who can now focus on more productive activities, such as novel course design, individual work with students, professional development, and so on. At the same time, honor reports provide students instant reward – in the form of provisional self-claimed lab credit – which stimulates students to actively engage in next lab assignments and other learning activities (all available in the study pack).

Possible Uses:

Honor reports can be used to grade programming labs in introductory computing courses. Beyond computing labs, honor reports can be applied to any kinds of activities that require student submissions: review questions, exercises, and problems. Lastly, honor reports may apply to required activities that require no student work submission – such as reading activities for example.

Related Patterns:

Online Study Pack, Self-Guided Programming Lab, Interactive Lab

3.2.5. Multiple-Attempt Quiz.

Problem:

How can you actively engage students in reading the textbook, if they are unwilling and possibly incapable of reading voluminous textbook chapters?

Constraints and Forces:

There is substantial anecdotal evidence on the increasing unwillingness of the net generation of students to read textbooks. In fact, some surveys show that students – and even instructors – consider textbooks to be one of the least important learning resources [14, 20]. These attitudes are misfortunate because textbooks are usually the highest-quality resources for the study academic disciplines. Students who are not willing to use textbooks deprive themselves from a systematic and complete coverage of their field of study. The unwillingness to use textbooks can be quite detrimental to the quality of one's learning.

It is possible to implement and conduct online quizzes that not only test and grade student knowledge, but also stimulate students to make use of their textbook and possibly other reading resources

Solution:

Use standard quiz functionality of CMS environments to implement online quizzes that are open to enrolled students for unlimited number of attempts for an extended period of time. To make sure students are dedicated to serious learning, conduct single-attempt closed-book comprehensive exams that are based on multiple-attempt quizzes.

Multiple-attempt online quizzes are intended to be integral parts of comprehensive study packs, together with their corresponding digital texts, tutorials, and other reading resources.

Resulting Context:

Led by the common desire for full credit, students usually make a number of consecutive attempts. In the process, students consult their texts to find correct answers – or to explain answers they have accidentally guessed. Therefore, multiple attempt quizzes trigger active reading. Students do not simply aim at guessing correct quiz answers but also do their best to understand and learn, in preparation for comprehensive quiz-based exams.

Multiple attempt online quizzes permit students to achieve highest possible credit and also truly learn in the process.

Possible Uses:

Multiple-attempt quizzes can be beneficially used in virtually any subject.

Related Patterns:

Online Study Pack

4. Conclusions. This paper presents five pedagogic patterns for active e-learning: the *study pack* pattern, the *guided interactive lab* pattern, the *guided programming lab* pattern, the *honor report* pattern, and the *multiple attempt quiz* pattern. These patterns formalize active e-learning practices in CMS environments as developed and used by the author and also adopted by other instructors from various schools (see Section 1.2).

From the whole collection, the *study pack* pattern is most important for it can be interpreted as an implementation framework for all other patterns. These patterns can be implemented by mainstream CMS environments, such as Moodle and Blackboard. With Moodle for example, we have implemented

study packs as Moodle course entities in which labs are realized by means of the assignment module, while quizzes and reports are implemented with the quiz module. (Before Moodle, we have used Blackboard in similar implementation schemes.) All implemented study packs incorporate significant original contents, such as hundreds of pages of e-texts and labs, and also significant slide and quiz collections.

Several student surveys testify that our active e-learning patters increase student interest [20, 21, 26]. Most notably, surveyed students agree that the use of integrated online study packs increase students' desire to learn more with an average evaluation of 4.4 on a 5-point scale [26]. Such study packs encapsulate self-guided labs, online quizzes, honor reports, and other online resources and activities.

REFERENCES

[1] Alexander C., S. Ishikawa, M. Silverstein. A Pattern Language. Oxford University Press, 1997.

[2] Anthony D. Patterns for Classroom Education. PLoP 1995, Monticelli, Illinois, 1995.

[3] Bailey T., J. Forbes. Just-in-Time Teaching for CS0. SIGCSE'05, St. Louis, Missouri, USA, 2005, 366–370.

[4] Bergin J., J. Eckstein, M. L. Manns, H. Sharp, eds. Feedback Patterns.
`http://www.jeckstein.com/pedagogicalPatterns/feedback.pdf`, October 2008.

[5] Bergin J., J. Eckstein, M. L. Manns, H. Sharp, eds. Patterns for Active Learning.
`http://www.jeckstein.com/pedagogicalPatterns/activelearning.pdf`, October 2008.

[6] Bergin J., M. L. Manns., K. Marquardt, J. Eckstein, H. Sharp, E. Wallingford, eds. Patterns for Experiental [sic] Learning.
`http://www.jeckstein.com/pedagogicalPatterns/experientiallearning.pdf`, October 2008.

[7] Bergin J. A Pattern Language for Initial Course Design. Proceedings of the 32nd ACM Technical Symposium on Computer Science Education, SIGCSE 01, ACM Press, Charlotte, North Carolina, USA, February 21–25, 2001, 282–286.

[8] Budd T. An Active Learning Approach to Teaching the Data Structures Course. SIGCSE'06, Houston, Texas, USA, 2006, 143–147.

[9] Cole J., H. Foster. Using Moodle: Teaching with the Popular Open Source Course Management System. O'Reilly, 2007.

[10] Dearden A., J. Finlay. Pattern Languages in HCI: A Critical Review. Human-Computer Interaction, Vol. **21**, No. 1, Lawrence Earlbaum Associates, Inc., (2006), 49–102.

[11] Gamma E., R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., 1995.

[12] Gonzales G. A Systematic Approach to Active and Cooperative Learning in CS1 and Its Effects on CS2. SIGCSE'06, Houston, Texas, USA, 2006, 133–137.

[13] Hillside Group. `http://hillside.net/`, October 2008.

[14] Lahtinen E., K. Ala-Mutka, H.-M. Järvinen. A study of the difficulties of novice programmers. ITiCSE'05, Caparica, Portugal, ACM Press, 2005, 14–18.

[15] Manolescu D., M. Voelter, J. Noble. Pattern Languages of Program Design 5 (Software Patterns Series). Addison-Wesley, 2006.

[16] McConnell J. Active and Cooperative Learning: Tips and Tricks (Part I). *Inroads – The SIGCSE Bulletin*, **37**, No. 2, (2005), 27–30.

[17] McInnis C., R. Hartley. Managing Study and Work. Commonwealth of Australia, Department of Education, Science and Training, 2002. `www.dest.gov.au/Highered/Eippubs/Eip02_6/Eip02_6.Pdf`, October 2008.

[18] Pollard S., R. Duvall. Everything I needed to Know about Teaching I Learned in Kindergarten: Bringing Elementary Education Techniques to Undergraduate Computer Science Classes. SIGCSE'06, Houston, Texas, USA, ACM Press, 2006, 224–228.

[19] PPP: The Pedagogical Patterns Project. Pedagogical Patterns. Retrieved from http://pedagogicalpatterns.org/ in October 2008.

[20] Radenski A. Digital Support for Abductive Learning in Introductory Computing Courses. Proceedings of the 38th ACM Technical Symposium on Computer Science Education, SIGCSE 07, Covington, Kentucky, USA, March 7–10, ACM Press, 2007, 14–18.

[21] Radenski, A. Python First: A Lab-Based Digital Introduction to Computer Science. Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 06, University of Bologna, Italy, June 26–28, 2006, ACM Press, 197–201.

[22] Razmov V., R. Anderson. Pedagogical Techniques Supported by the Use of Student Devices in Teaching Software Engineering. SIGCSE'06, Houston, Texas, USA, ACM Press, 2006, 344–348.

[23] Rising L. Understanding the Power of Abstraction in Patterns. IEEE Software, Special Issue on Software Patterns, July-August 2007, 2–7.

[24] Roschelle J., D. Tatar, S. R. Chaudhury, Y. Dimitriadis, C. Patton, C. DiGiano. Improvisation, and Interactive Engagement: Learning with Tablets. *IEEE Computer*, **40**, Issue 9 (2007), 42–48.

[25] Rössling G., M. Joy, A. Moreno, A. Radenski, L. Malmi, A. Kerren, T. Naps, R. Ross, M. Clancy, A. Korhonen, R. Oechsle, J. Á. Velázquez Iturbide. Enhancing Learning Management Systems to Better Support Computer Science Education. SIGCSE Bulletin Inroads, 2008 (peer reviewed and accepted).

[26] Study Pack. Student Surveys. `http://studypack.com`, October 2008.

[27] Sun Microsystems, Inc. Java^{TM} Platform, Standard Edition 6 API Specification.
`http://java.sun.com/javase/6/docs/api/`, October 2008.

[28] TEN Competence. The European Network for Competence Project. `http://www.tencompetence.org/`, October 2008.

[29] Vallino J. Design Patterns: Evolving From Passive to Active Learning. 33rd ASEE/IEEE Frontiers in Education Conference, Boulder, Colorado, USA, 2003, S2C19–24.

[30] Warren I. Teaching Patterns and Software Design. 7th Australasian conference on Computing education – Vol. **42**, Newcastle, New South Wales, Australia, 2005, 39–49.

Atanas Radenski
Chapman University
Orange, CA 92869, USA
e-mail: `radenski@chapman.edu`

Received November 18, 2008
Final Accepted November 25, 2008