Enhancing Learning Management Systems to Better Support Computer Science Education

Guido Rößling (co-chair)
TU Darmstadt
Dept of Computer Science
Germany
+49 157-72 52 4013
roessling@acm.org

Lauri Malmi (co-chair)
Helsinki Univ. of Technology
Dept of Computer Science and Engineering, Finland
+358 9-451-3236
lauri.malmi@tkk.fi

Michael Clancy
Computer Science Division
Univ. of California, Berkeley
USA
+1 510-642-7017
clancy@cs.berkeley.edu

Mike Joy
Univ. of Warwick
Dept of Computer Science
United Kingdom
+44 24-7652-3368
m.s.joy@warwick.ac.uk

Andreas Kerren
Växjö Univ., School of Mathematics and Systems Engineering, Sweden
+46 470-76-7102
andreas.kerren@vxu.se

Ari Korhonen
Helsinki Univ. of Technology
Dept of Computer Science and Engineering, Finland
+358 9-451-3387
ari.korhonen@tkk.fi

Andrés Moreno
Univ. of Joensuu
Dept of Computer Science
Finland
+358 1-325-17929
andres.moreno@cs.joensuu.fi

Thomas Naps
Univ. of Wisconsin - Oshkosh
Dept of Computer Science
USA
+1 920-424-1388
naps@uwosh.edu

Rainer Oechsle
Univ. of Applied Sciences Trier
Computer Science Dept
Germany
+49 651-8103-508
oechsle@fh-trier.de

Atanas Radenski
Dept of Mathematics and Computer Science
Chapman Univ.
USA
+1 714-744-7657
radenski@chapman.edu

Rockford J. Ross
Montana State Univ.
Computer Science Dept
USA
+1 406-994-4804
ross@cs.montana.edu

J. Ángel Velázquez Iturbide
Univ. Rey Juan Carlos
Dept de Lenguajes y Sistemas Informáticos I, Spain
+34 1-664-74-54
angel.velazquez@urjc.es

ABSTRACT

Many individual instructors — and, in some cases, entire universities — are gravitating towards the use of comprehensive learning management systems (LMSs), such as Blackboard and Moodle, for managing courses and enhancing student learning. As useful as LMSs are, they are short on features that meet certain needs specific to computer science education. On the other hand, computer science educators have developed—and continue to develop—computer-based software tools that aid in management, teaching, and/or learning in computer science courses. In this report we provide an overview of current CS specific on-line learning resources and guidance on how one might best go about extending an LMS to include such tools and resources. We refer to an LMS that is extended specifically for computer science education as a *Computing Augmented Learning Management System*, or *CALMS*. We also discuss sound pedagogical practices and some practical and technical principles for building a CALMS. However, we do not go into details of creating a plug-in for some specific LMS. Further, the report does not favor one LMS over another as the foundation for a CALMS.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science Education*

General Terms

Design, Human Factors

Keywords

Learning management system, LMS, CALMS, computing augmented learning management system, computer science education.

1. INTRODUCTION

Computer science education is inherently tied to the use of software for teaching, learning, and course management. Software systems in use range from narrowly-focused, instructor-developed models intended to help students learn a specific algorithm (e.g., Quicksort), to full-featured commercial or open-source integrated program development environments, complete with editors, compilers, debuggers, and version control capabilities. In between (or in parallel) lies a continuum of ever more ambitious systems for enhancing course management and student learning. Examples include systems that provide for

- online submission of assigned programming exercises with automatic assessment and feedback to the student and/or instructor (such as BOSS [65], Web-CAT [44], CourseMarker [55], and ASB [93]),
- online exercises with automatic assessment and feedback for abstract concepts such as data structures and algorithms, formal languages and grammars, real computer architectures, or theoretical machine models, such as TRAKLA2 [84], JHAVÉ [97], JFLAP [118], ACE, and SQL Tutor [90],
- algorithm or program visualizations systems, such as ANIMAL [127], Jeliot 3 [92], MatrixPro [65], and ViLLE [115],
- interactive compiling and debugging environments, such as BlueJ [13] and Eclipse [43], and
- program plagiarism detection systems, including Moss [3], JPlag [108] and Sherlock [64].

Despite the richness of these available CS specific resources, one interesting observation can be made about these sorts of systems. They are seldom used as widely beyond the institution at which they were created as one might expect, despite being perceived by other instructors as potentially quite useful. The reasons for this are traceable to certain issues: instructors are often too busy to locate, learn, teach students to learn, design exercises around, and integrate such software into the fabric of their courses, especially when the notation and methodology of the system do not precisely match those of the course, thereby creating problems for students [28, 130].

A Learning Management System (LMS) presents a potential avenue for both enriching the content of a course and for solving the course integration problem (for comparison, hypertextbooks offer another approach to solving the course integration problem [120]). In this paper, we do not make a distinction between an LMS and a Learning Content Management System (LCMS) as, for example, Wikipedia presently does. This is because, from the specific point of view of computer science, the difference between these two is not clear. Our basic premise is that *instructors should be able to tailor a course environment to their preferences over time by integrating computer science content into an LMS with plug-in modules for course management or learning software developed by themselves or others.* The intent is that the whole would represent a cohesive teaching, learning, and course management environment. We refer to a standard LMS so modified to support computer science education specifically as a Computing Augmented Learning Management System (CALMS).

In the rest of this report, we pursue this idea in depth and provide pedagogical, practical and technical guidelines for carrying out such integration work. We focus specifically on full-featured, extensible LMSs similar to Blackboard and Moodle, but believe that our recommendations will apply to other, less general environments as well. We also do not intend to prescribe how to write a plug-in for existing LMSs, nor to promote one LMS over another, but rather to describe the features and characteristics proposed plug-ins and extensions should have.

It is beyond the scope of this report, as well as beyond the means of a Working Group, to actually develop a full-fledged CALMS. We instead discuss the most relevant aspects of a CALMS. This shall enable future researchers and developers to extend a given LMS or other system to the concept of a CALMS as described in this paper.

We pursue our objectives by first describing the results of a survey conducted prior to the start of the working group. The survey asked members of the computer science education community about which LMS, if any, they utilized in support of their courses, how they used it, which features they found useful, and which features were missing with respect to computer science education.

For this end, we examine the state of the art with respect to existing teaching, learning, and course management software systems, which have the potential for enriching the teaching and learning experience as extensions to an existing LMS. This is followed by a look at pedagogical and technical "best practices" and considerations to be respected when integrating such systems into an LMS.

Next, to concretize our goals we provide a number of scenarios as examples of extending an LMS with selected CS specific teaching, learning, and course management resources. This section shall both motivate the use of a CALMS—once it is built—and raise the interest of potential developers to support the creation of a CALMS.

Finally, we provide a summary of the paper and some suggestions for future work.

2. SURVEY RESULTS

Before the working group convened at ITiCSE, we conducted an online survey to determine the nature of CS educators' use of and attitudes toward computing augmented learning management systems (CALMSs). Notice of the survey was posted to a variety of CS educator lists, but disappointingly only thirty-one responses were obtained; thus, the conclusions that we draw from those responses should not be viewed with statistical confidence. Nonetheless, those educators who did respond showed a great deal of interest in the topic, and from an anecdotal perspective, their responses offered our group some valuable insights. These insights include the following:

1. Defining exactly what we meant proved to be a difficult task. The definition used in the survey was as follows:

A CS-specific learning environment is regarded as a learning resource that works in the web or as a stand-alone application, has CS-specific content, may contain dynamic, CS-specific elements, such as algorithm or program animations or visualizations, automatic assessment, simulations, ... may be

designated for some specific purpose to support learning CS, should ideally be available to instructors or learners from other institutes, universities, or countries.

This definition caused confusion on the part of the respondents. As one respondent put it:

"I am confused by this survey's introduction. The first sentence mentions course management systems in juxtaposition with on-line learning environments. The given definition of 'CS-specific learning environments' reduces to 'an online learning resource that has CS-specific content,' and gives no examples. A Java applet that displays the powers of 2 from 1 to 256 satisfies this definition. With this broad a definition, it is impossible to answer the questions. I'm going to focus on course management systems."

This confusion led to a breakdown into two different types of responses to our survey—those responses from educators who were in fact using a general-purpose LMS, and responses from those who were using smaller online software tools specifically designed for computer science purposes, such as online code submission/evaluation, simulators for models of computation, or algorithm visualization. Out of the 31 respondents, 12 indicated they were only using a large-scale LMS, 13 indicated they were only using special-purpose learning software explicitly designed for a CS topic, 2 indicated they were using both, and 2 responses were unclear on this point. None of the respondents indicated that the software in the CS-specific category was seamlessly incorporated into a large general-purpose LMS.

2. Several respondents indicated in various ways that a seamless integration would greatly improve such learning environments. These views became apparent in the questions we asked regarding problem areas and obstacles to adoption for such learning environments. For example, one respondent said "Broadly, they (large-scale LMSs) are not flexible enough. They constrain the user to the developer's own idiosyncratic choices of course design, topic coverage, course management." Another indicated that "Support for integration of algorithm visualizations (into large-scale LMSs) is usually lacking". Another respondent complained that "With the generic approach (of large-scale LMSs), I can include visualizations, but I don't have the capacity for maintaining and displaying student grades securely, for example."

A respondent who is using the Web-CAT set of tools for evaluating student submissions of programs commented: "The tools we use are very nice for what they do, but together they do not cover the full spectrum of what is required. They also are not integrated. Being able to keep one unified gradebook and to be able to import/export with the university's registration and grades system would be nice. Also, the content management features are not as robust or flexible as in something designed specifically for that task (say, Drupal). While Web-CAT does a great job with programming assignments, we do not have the same support for other kinds of assignments (particularly written homework or exams)." Another respondent who used some of the CS-specific online tools noted that "Cooperation and data exchange with other tools is also less than good."

The conclusion that we drew from these kinds of anecdotal responses was that *there is a growing need to integrate the activities fostered by the CS-specific tools with the broader and more general capabilities of LMSs*.

3. In response to the question "What are the main features or functionalities in the resource(s) that you appreciate?", the two most frequently cited features were automatic feedback and automatic assessment. So the ability of a system, be it a large-scale LMS or a more specific tool, to provide students with feedback and instructors with ways of assessing the performance of their students are critical factors in determining whether or not these tools are successful. The ability of the tool to provide immediate feedback to students was seen as "more important" for those respondents who were using the smaller, special-purpose tools. Approximately two-thirds of them cited automatic feedback as an important feature, while only one-third of the users of large-scale LMSs cited it as important.

4. Some concern was expressed about the openness of the CALMS. Is it proprietary? Is the code for the tool open source? Is the learning content provided accessible to anyone, or only students who are enrolled in a particular course? One respondent said: "I avoid standard course management systems such as Blackboard or Moodle on philosophical grounds—I will never use any system that builds walls around access to my course materials. Any CMS I use must be capable of allowing standard Internet search access to those course pages that I want." (We parenthetically note that this response was not accurate with respect to Moodle. Moodle offers content creators the option of making course content searchable on the web.)

5. In response to a question that asked about CS educators' satisfaction levels with the system they used, only one of the thirty-one respondents indicated satisfaction at a level "Perfectly suited for the way it is used in your teaching", 12 indicated the system(s) they used were a "pretty good fit for the way it was used in their teaching", 13 indicated the system(s) were "OK, but wished there was something better". Only one indicated complete dissatisfaction with such systems.

6. In response to the question "What problems or issues would cause you NOT to adopt a given learning environment?", the issues that were cited by over half of the respondents were:

- The learning environment is difficult to learn how to use: 17 out of 31
- Cost: 17 out of 31
- Browser or operating system issues: 16 out of 31
- Missing, incorrect or insufficient content: 16 out of 31

Other problem areas with existing systems, though not at a level that would stop adoption, were anecdotally noted by respondents. We present those responses verbatim here:

- "A system that comes with pre-written exercises either doesn't quite match one's preferred teaching style, or requires a significant amount of work to select and arrange exercises that do. On the other hand, a system without pre-written exercises requires even more work to learn the coding and build those exercises."
- "Dynamic paths for student learning are missing."
- "The main problem is that my institute has had to put a lot of effort on developing the environments by ourselves because there were no environments that suited our situation exactly according to our needs."

- "Far too many clicks required to perform common tasks, with significant delays to perform useless computations thrown in for added aggravation."
- "Team interaction tools that assist both synchronous and asynchronous team work."

In total, the feedback from the survey indicated an interest in the development of a CALMS as outlined in Section 1. It is important to note that the actual definition of what constitutes a CALMS was only determined during the Working Group meeting during ITiCSE; thus, the survey did not simply confirm or modify our intentions, but rather played a large role in helping to define them.

3. LEARNING RESOURCES OVERVIEW

Given the perceived usefulness of a CALMS, the goal of the Working Group was to help shape the vision of the components that make up a CALMS. Instead of reinventing the wheel, a CALMS should incorporate aspects from the many very good aspects offered by existing LMSs or specific tools. We will therefore start by giving an overview of the existing learning resources. The reviewed systems, tools, and concepts can all act as candidate components for a CALMS. We note that the list is by no means comprehensive, but rather summarizes good examples of resources that are available. Moreover, we do not aim at making a deep comparison and analysis of the resources, but rather provide pointers to interested teachers and researchers who consider building CALMSs.

We start with a brief overview of the types of systems and tools covered in the following subsections. We will consider the following kind of systems that can be used for computer science education:

- tools relating to general pedagogy;
- augmented learning systems;
- specialized learning management systems,
- algorithm visualization tools;
- program visualization tools.

Most of the systems and tools described in the following subsections do not directly or indirectly export their contents to an established LMS. Existing LMSs provide facilities for collaboration, assessment, and other kinds of pedagogy. We begin by describing two kinds of extensions to these facilities:

- pedagogical tools, typically not yet found in the mainstream LMSs, that provide alternatives for collaboration, different types of system interaction, or different media for interaction, and
- augmented learning tools covering all systems that try to improve the learning process during the "presence teaching" (as opposed to e-learning or distance education) phase, mostly concerning the lecture, but also including exercise and (presence) lab sessions.

We call systems that are dedicated to providing CS specific exercises and giving feedback on the submissions as Specialized Learning Management Systems (SLMSs). Many of these are related to programming and generally they are called automatic assessment tools. However, there are other application areas as well. In addition, we consider tools that focus in particular on assessment, collaboration, and so forth, within a programming environment. Most of these rely on the coupling of an LMS with an Integrated Development Environment (IDE) for programming that allows extensions, or plug-ins, to add to the functionality of the IDE. Two such IDEs are BlueJ [13] and Eclipse [43].

Algorithm Visualization (AV) tools portray the (static and dynamic) behavior of an abstract description of software, for example, a given algorithm, an algorithm family such as sorting *int* values stored in an array, or a larger set of algorithms (e.g., by using a scripting language to allow the user to "program" the visual appearance).

Program Visualization (PV) tools show the behavior of a program written in a specific programming language by displaying the effect of individual operations, such as the program state. Each command or state change, such as the evaluation of an expression part, is visualized separately, so that users can directly see the effect of each operation.

Both AV and PV tools are subsumed by the field "Software Visualization" (SoftVis). The overall aim of SoftVis is to help the user understand, improve, debug and test software. Thus, education is only one aspect of this field. It is also useful for visualizing the structure of large systems or the evolution of software over time, as well as to find mistakes or inconsistencies. There are several anthologies that provide overviews on this field [37, 139].

3.1 Tools Relating to General Pedagogy

A number of learning environments include pedagogical features that go beyond those provided by the typical LMS. These features are briefly described below. They are not domain-specific, and are intended for use in a wide variety of contexts.

3.1.1 Collaboration

Research in the area of computer support for collaborative work is expanding, and some of the results are appropriate for use in education.

Vision Quest [148], a group decision support system, offered tools for brainstorming and for rating and reorganizing alternatives. (This system apparently is no longer available).

The family of tools referred to as W3 Interactive Talk (WIT) [152] generalizes the threaded discussion mechanism by allowing different types of threads.

WISE [79] includes a "gated collaboration" tool that presents a question to a student; after a response is provided, the student is allowed to review and rate the answers of his/her classmates and to amend his/her own answer if the review suggests a better alternative. This facility can also be used for formative assessment. For example, changing one's response to a better version is evidence of learning.

UC-WISE [27], a system built atop WISE to support semester-long courses built from WISE activities, allows a curriculum author to annotate WISE steps with metadata. It also provides a similar, more flexible tagging facility for students, which are intended to stimulate a folksonomic sort of collaboration.

3.1.2 Scaffolding

"Scaffolding" refers to support given to students in problem solving. Initially substantial help and guidance are provided; subsequently, the "scaffolds" are gradually removed, and students are required to do more and more on their own. Systems support scaffolding in a variety of ways.

WISE provides a hinting facility [33] with which a curriculum author may scaffold students.

CSILE (Computer-Supported Intentional Learning) [132] is a networked learning environment intended to foster higher-level processes of inquiry among elementary school students. It includes scaffolds to support students in areas such as text analysis, theory-building, and debating, along with a construct called a "Thinking Type" that guides students to engage in deepening inquiry.

Belvedere [78] is another system aiming to foster critical inquiry skills. It suggests candidate steps for proceeding further, and structures access to materials and activities.

3.1.3 Reflection

By “reflection”, we mean learning from experience. Different ways to encourage reflection are briefly described below.

A WISE curriculum author can include prompts to explicitly encourage reflection.

Vision Quest included a "reflective follow-up" step, to be completed after a decision has (provisionally) been made.

ALEL [76], which was developed to teach experimental research methodology and statistical inference, displays a tree that represents a student's activity path; this graphical representation can be used as a tool for collaborative reflection.

3.1.4 Multiple representations

Multiple representations of a concept—for example in diagrams, animations, or physical activities—contribute to richer understanding of the concept. ALEL's tree representation of actions has already been noted in support of the decision-making process. To foster better knowledge organization, WISE supports the creation and sharing of *concept maps* [101]. A concept map is a diagram showing relationships between concepts. The diagram consists of nodes, which represent concepts, and labeled edges connecting pairs of nodes, which represent how one of the pair of concepts is related to the other.

3.1.5 Support for "what if …?" activities

This problem solving process typically includes choice points where the solver tries out two or more solution approaches. Future Learning Environments (FLE) [75] is a web-based learning environment that focuses on creating and developing expressions of knowledge (i.e., knowledge artifacts) and design. FLE maintains a tree diagram of artifacts. When a choice is made to modify an artifact, the system extends the tree diagram, thus making clear the "inheritance structure" of artifact modifications.

3.1.6 Authoring

Authoring with the typical LMS is free-form, using a text editor or web form to specify each activity. In contrast, the Pattern-Annotated Course Tool (PACT) [24] allows a course to be authored based on *pedagogical patterns*, descriptions of outstanding teaching practices recorded in a format that facilitates a common vocabulary for pedagogical research and practice, is accessible to novice instructors, and encourages the repurposing and reuse of techniques that are solidly grounded in modern pedagogy. Its visual interface displays the association between curriculum activities and the corresponding pedagogical patterns; patterns may be easily instantiated and associated with corresponding activities.

PACT has been found to be useful in several scenarios, including the following [25].

- Annotation by expert course authors. Expert content developers have acquired substantial understanding of what works in a course and what does not. Often, however, this knowledge is difficult to uncover. The process of annotating a course with references to pedagogical patterns seems to help experienced instructors articulate their own understanding of their design and thereby make it more accessible to others.
- Content creation by novice instructors. PACT makes it easy to copy large chunks of an existing course to a newly created course. With typical courses, one instructor hands copies of homework, exams, lecture notes, and course syllabi to the next instructor. These artifacts, however, are probably insufficient to reveal the underlying *rationale* for the various course activities, precisely the information that PACT is designed to provide.
- Mediation for discussion. As a highly visual medium, the PACT interface makes an excellent visual aid for describing and discussing issues in pedagogy and curriculum design. It allows the pedagogical expert to review their own annotation with fellow teachers and researchers (both novice and experienced) to elicit ideas and stimulate discussion of improvements to content and structure.

3.2 Augmented Learning

Augmented Learning approaches try to support the "classical", lecture-based teaching style using computer support. They thus provide support for teaching that goes beyond simply plugging a laptop into a projector and running PowerPoint, without targeting distance education. The underlying rationale is that the lecture setting, whilst by no means perfect, plays an important social role in presence teaching: it provides both a focal point and a meeting place between lecturer and students and among students. It therefore also helps to build a sense of "community" and of "social presence" [131].

Applications in this area typically aid the presenter by providing features such as an annotation layer for writing on PowerPoint slides or whiteboards, as done in the Lecturnity [62] or E-Chalk products [46]. Apart from annotations, both systems also support recording the lecture materials, including annotations. Ubiquitous Presenter [50] adds another layer by proving a "teacher" and a "learner" perspective. In this mode, the teacher can see a different version of the learning materials, for example, annotated with the model answer. They can then simply use a pen on a tablet PC to draw the highlighted elements of the model answer and thereby "develop" the answer in-class [50, 151].

Many systems support interaction during lectures. The simplest application is to allow the students to "raise their hands" electronically—which can be useful for the large classes often found in European undergraduate courses, which may have more than 400 attendees. An additional application is participation in online quizzes [10, 12, 80, 133]. A more interesting aspect of interaction in the classroom is submitting questions to the teacher. Again, several systems address this concern, including Ubiquitous Presenter [50], which allows questions or content to be submitted to the teacher. Most systems do not export their content to a LMS.

The Digital Lecture Hall system is an integrated software platform that combines presentation, annotation, recording, interaction and annotation support. Its TVremote subcomponent provides student clients for all Java-enabled platforms ranging from cell phones to PCs [10]. The interaction content can be addressed directly during the lecture by the teacher, who is informed about new questions by a counter showing the number of "open" questions [12]. Alternatively, a co-pilot, typically a research assistant, can handle questions while the lecturer continues presenting. In this case, the teacher can continue teaching while the assistant intercepts and answers (most of) the incoming questions. Significant questions or important points are marked for the teacher to be presented to the audience. Finally, the assistant or teacher can use the same software client to answer questions outside the lecture.

In a 2005 survey of 369 CS1 students, 45% stated that it would be easier for them to ask questions if they could stay anonymous [11]. However, fully anonymous submissions are often not helpful, as they may incite some students to post "funny messages", and also remove the chance to reply. TVremote therefore uses a pseudo-anonymous address that a server can resolve back to a concrete email address. Thus, the teacher can reply to the students' questions, and still allow them to be anonymous in the sense that there is no connection between the automatically assigned pseudonym and the actual student's name—in fact, as names are assigned randomly (and then kept throughout the course), not even the gender of the name may match. Again, there is currently no direct export of the interactions taking place during the lecture to an LMS.

3.3 Specialized Learning Management Systems

There is a large number of learning management systems, some of which are freely available, including the popular system Moodle [29]. In addition to these general purpose LMSs, there exist a number of systems that are intended for CS education only. These special purpose systems (SLMSs) typically deliver assignments and exercises that are beyond the scope of general purpose systems. At least two main types of such systems can be identified: systems that check programming exercises (the deliverable is software) and systems that check conceptual knowledge in a specific topic.

Many SLMSs include some characteristics of general LMSs. They might lack some characteristics, but still expand the applicability in terms of automatic assessment and similar features. A 2003 ITiCSE Working Group [26] defined Computer Aided Assessment (CAA) as "any activity in which computers are involved in the assessment process as more than just an information storage or delivery medium." They identified five different types of CAA in CS Education: Multiple-choice questions, textual answers, programming assignments, visual answers, and peer assessment. In the following we are focusing on SLMSs that are intended to provide numerical marking and feedback in both textual and visual formats. Examples of such systems include Web-CAT [44], BOSS [65], JHAVÉ [97], CourseMarker [55], TRAKLA2 [84], SQL Tutor [90], and ACE.

Their general functionality is to provide on-line exercises for students and automatically assess and give feedback on their submissions. The exercises can deal with different topics, such as programming (BOSS, CourseMarker, Web-CAT), diagrams (CourseMarker), algorithmic exercises (TRAKLA2), SQL statements (SQL Tutor) or formal automata (ACE). In programming exercises, many different aspects can be assessed, such as correctness, programming style, use of required language structures, program efficiency, and so forth.

Facilities already exist for administering and correcting short programming exercises, for example, JavaBat [104] and JExercise [142]. Tools also exist [123] that allow an author to ask for code from the student, then to wrap that code with the rest of a checking program, to compile and run it, and to return and record the result, perhaps accompanied by some analysis.

Web-CAT is a submission and autograding tool. From the web site of its author, Stephen Edwards: "Web-CAT is a plug-in-based web application that supports electronic submission and automated grading of programming assignments. It supports fully customizable, scriptable grading actions and feedback generation for any assignment. The Web-CAT Grader supports traditional models of automated program grading, but also supports grading of assignments where students do their own testing. It helps encourage test-driven development (also called *test-first coding*), where students write small unit tests for each piece of code they add. Web-CAT allows a student to submit his or her test cases along with the solution, and grades on test validity and test completeness as well as code correctness." [http://people.cs.vt.edu/~edwards]

A tool that allows creation, sharing, and verification of box-and-pointer list structure diagrams would provide a non-text medium for assessing student understanding of data structures. TRAKLA2 [84] provides functionality close to this. Typically, students can submit their solutions several times, and use the received feedback to improve their solutions. Students graphically manipulate a given data structure representation to simulate the working of the algorithm, and the initial data structure is different for each student, even for each instance of the assignment for the same student.

The systems generally record the assessment results in a database, which allows the assignments to be used as part of the final grading of the course. This typically requires a connection to the database server. However, some tools also allow students an off-line practicing mode, in which case the results are not recorded.

A central functionality of all these systems is course management, for example for creating larger entities, such as units or rounds assembled from the pool of separate exercises. Thus the teacher can structure the whole course to match the lecture schedule, and set up deadlines for submissions. The

systems also allow online monitoring of student progress, and generation of summaries of the results.

Typically, model solutions can be published after the deadline for submissions is over. However, TRAKLA2 allows the presentation of model solutions—as dynamic algorithm visualizations—after each submission, because the learners get new initial data for the next time they solve the same exercise.

Many systems log additional information about student activities, such as date, time, and total number of submissions. TRAKLA2 also logs information about the use of model solutions, which allows teachers to gather data about problematic issues within the learning content.

3.4 Algorithm Visualization Tools

Many algorithm visualization tools are freely available. In the following, we will therefore focus on key aspects of such systems and refer to established systems, without presenting them in detail.

Rößling and Naps [129] have described a set of requirements to make AV tools effective learning tools.

- Users should be able to provide input to the algorithm. This is achieved in the content generators provided by ANIMAL [122] and JHAVÉ [98], as well as by the approach taken by Matrix Pro [66].
- It is very important to have an unlimited rewind facility, allowing students to step backwards and forwards as needed to gain a better understanding. This function is available at least in the ANIMAL, JHAVÉ, MatrixPro, and TRAKLA2 systems, but missing or limited in many other systems.
- The system should provide a structural view of the algorithm. This view allows learners and teachers to directly jump to key aspects, without having to search for them or navigate through less interesting steps. This feature is offered by ANIMAL, provided that the content author provides appropriate markup.
- The inclusion of activating elements such as “stop and think” questions [96, 99] or "visual algorithm simulation exercises" [84] can help raise the learner's engagement, thus helping to achieve better learning outcomes.

A previous working group report proposed the integration of algorithm visualization tools with a learning management system and a database for course management reasons, for example, to track student activities and points reached. The resulting system is called a Visualization-based Computer Science Hypertextbook [130]. Partial implementations of such systems are now available in TRAKLA2, and as a Moodle module for the AV systems JAWAA2, ANIMAL and JHAVÉ [125]. These systems, when they have been further refined, will be a good stepping stone towards a CALMS.

From the perspective of CS-specific education, algorithm visualization tools should support a set of relevant data structures and/or algorithms. This includes at least arrays, matrices, graphs, and list elements, as offered in JAWAA2 [4], ANIMAL, JHAVÉ, and MatrixPro among other systems. There are other relevant data structures, such as queues, stacks, and trees, which are supported by JAWAA2, JHAVÉ, and MatrixPro.

ANIMAL allows content to be generated using drag and drop actions, with a scripting language [128], a Java API [124], or a set of built-in generators [122]. Each approach, except for the first, can be used to automate the generation of animation content. MatrixPro takes a different approach in terms of visual algorithm simulation [71] for the rapid creation of algorithm visualizations and animations. The same technique is utilized in TRAKLA2 exercises. The focus of MatrixPro lies in portraying and manipulating algorithms and data structures in a lecture "on-the-fly" (i.e., without prior preparation before the lecture).

The inclusion of source or pseudo code next to the actual visualization can help learners to understand the connection between what they see happening and the matching code lines. The ANIMAL system provides this, together with indentation and highlighting support, to indicate the current line of code under execution. A similar approach is taken in the model answers of TRAKLA2 exercises that show step by step how the visual algorithm simulation is supposed to be carried out.

For international users, one important aspect can be the internationalization of the graphical user interface and the content. ANIMAL currently supports English, German and Italian, with plans for Russian, Bulgarian and Finnish. The same underlying translation package [126] has been used to translate the Jeliot 3 program visualization system into English, Finnish, German, Spanish, and French. TRAKLA2 exercises are available in Finnish and English with plans for Spanish assignments.

3.5 Program Visualization Tools

The number of program visualization and animation tools for educational purposes is lower than the number of algorithm animation and visualization tools. The effort needed to write a simple program visualization tool is far higher than for a simple algorithm animation tool. For example, a program visualization tool needs to parse, analyze, and interpret or "render" the statements and operations of the underlying programming language. Even for a small programming language, this results in a large set of aspects that have to be covered, such as variable declarations and lookup, method declaration and invocation, parameter passing, expression evaluation, and so forth.

As the approaches taken by the different systems cover many different aspects, we will briefly describe the basic features of selected representative systems.

Jeliot 3 visualizes user-written Java programs and object-oriented features [92]. Method calls, variables, and operations are visualized as the animation progresses, allowing the student to follow the execution of a program step by step. A problem with this type of program visualization tool is inflexibility regarding the visual mapping from the program state to the visual representation. ViLLE [115] is a program visualization tool that addresses the same challenges, but in language-independent fashion—the visualizations can be viewed in any of the predefined languages. It has a parallel view that displays program execution in two languages simultaneously. As in many PV systems, it is possible to trace program execution line by line and monitor program outputs and changes in variable values. Moreover, in order to make visualizations easily

interpretable, ViLLE has an automatically generated textual description of each code line, including the role of variables. Other, non-automatic approaches used in Algorithm Visualization (for example, the so-called interesting event approach [39]) allow the user to freely change the resulting visualization [35].

Alice [31] allows the user to visually assemble programs by selecting statements from a set of possible operations and filling in placeholders, such as the condition for an *if* statement. The visualization of the program is done in a virtual 3D environment. A similar approach to programming is taken in the Greenfoot system [52].

A different kind of system produces animations at the border between algorithm animation and program visualization. Such program visualization systems aim at displaying high-level programming paradigms, specific language features, or specific classes of algorithms, and deliver animations very close to those rendered by traditional algorithm animation systems. For instance, WinHIPE IDE [103] graphically displays expressions for a functional programming language. The resulting animations exhibit a high level of abstraction and are therefore similar to algorithm animations. Another example is the SRec system [144] that automatically generates visualizations and animations of recursive methods in Java. SRec provides multiple views of recursion, including recursion trees, which are typical visualizations in many algorithms.

4. PEDAGOGY

Apart from the actual systems and approaches described in Section 3, the developers of a CALMS should also be aware of those aspects of pedagogy that apply to building and using a CALMS. This includes both a sound foundation of pedagogical theory that we discuss in Section 4.1, and a number of practical considerations of which a teacher should be aware (Section 4.2). After these we present "pedagogical patterns" useful in a CS education context (Section 4.3).

4.1 Pedagogical Theory

Effective design and construction of a CALMS must be underpinned by sound pedagogy. Not all aspects of education theory that we discuss here are necessarily relevant to each CALMS instance, but an awareness of these issues, and the application of pedagogical principles where appropriate, are fundamental.

4.1.1 Instructional process

Teaching and learning can be considered together as a single process, called the instructional process, which contains several phases—see Figure 1. The process starts with defining goals or learning objectives, which requires the listing of goals that are to be achieved during the process. The next phase is the mapping of a pathway, describing how the goals are to be achieved. This includes defining the content to be covered, the learning tasks to be carried out, the order of those tasks, how assessment will be carried out, and so on. This is the followed by the teaching/learning phase during which the tasks are carried out. The final phase is evaluation of learning outcomes where the initial learning objectives can be compared to what was actually achieved. An essential part of the process is feedback that informs future iterations of the process. For example, the learning goals may need refinement or up-/downgrading, the instructional methods may need to be changed to better match the set goals, or more time might be needed for a certain phase in teaching.

Note that the process model can be applied on several levels: such as course level or a single assignment level, or even on a curriculum level.

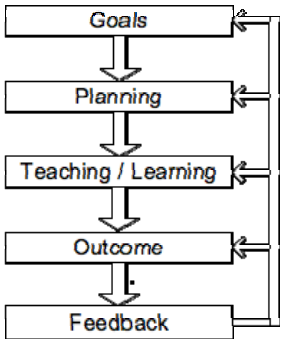

Figure 1. The instructional process [89]**.**

It is also important to note that the instructional process can be considered both from the teacher's perspective and the learner's perspective. A teacher selects teaching methods, prepares learning resources, defines course schedules in the planning phase, and interacts with learners in various ways during the process. Learners also set their own goals, which may be the same or different from those stated by the teacher. Learners also plan a personal schedule for working, which resources they are going to access or acquire, which tasks they are going to perform (or, perhaps, skip), and so forth.

The instructional process may also be regulated by the institution arranging the instruction. General objectives for courses and whole programs are set, specific teaching methods or instructional arrangements may be supported or required, and certain forms of assessment can be requested. The institution may also set limits for what resources, such as an LMS, are allowed to be used or should be used.

4.1.2 Learning theories

Learning theories provide behavioral or psychological models about the learning process. In particular, behaviorism, constructivism and cognitivism have gained special attention.

Behaviorism [136] is based on the assumption that human behavior can be explained just by publicly observable processes, without any internal mental processes. Education is organized according to external events and outcomes, such as classes, assignments and assessments. Behaviorism has been criticized for only supporting processes of lower intellectual level. However, these processes are necessary in most disciplines, including CS. For instance, coding requires extensive practice and repetition in using the different programming constructs.

The cognitivist approach [6] supposes that a learner has clear and discrete mental states and thought processes that can be regarded as changing, or being changed, in an algorithmic fashion. This is a richer view than that of behaviorism, since the internal mind states of a learner are perceived as complex, rather than the result of the learner responding in a simple and controllable way to stimuli.

Constructivism [21, 109] suggests that individuals construct new knowledge from their experiences through processes of accommodation and assimilation. Students build their own meanings of knowledge in interaction with their environment and opinions of other people. Thus, the instructor must design learning experiences that are oriented to the use of previous knowledge and construction of new knowledge. For example, a constructivist approach is taken in collaborative learning, which emphasizes individual and collective experiences and collaborative processes that lead to knowledge acquisition.

4.1.3 Learning taxonomies

Taxonomies of educational objectives describe and categorize the stages that an individual may reach during a learning process. They greatly vary in aim and structure. Some taxonomies divide educational objectives into three domains—cognitive, affective and psychomotor—whereas others try to integrate them. On the other hand, some taxonomies, such as Bloom's taxonomy [20], categorize student performance in a linear hierarchy, whereas others, like the revised Bloom's taxonomy [7], use a matrix.

Learning taxonomies can be seen as a language which can be used in a variety of educational contexts. One common use is the definition of the curriculum objectives of a course, so that the desired level of understanding for each topic is stated. Taxonomies are also used to assess students' performances. Another application is checking curriculum alignment, which is the matching of course learning goals, activities planned to achieve these goals and assessment of student performance. Finally, learning taxonomies have been used in other contexts, such as a framework to specify educational applications or materials, structuring exercises in computer-based and computer-assisted instruction, or introducing students to a learning taxonomy to raise their awareness and improve their level of understanding and their studying techniques. Each of these uses is relevant to CALMSs.

Commonly used taxonomies utilized within CS education include Bloom's taxonomy [20] and SOLO [19]. Their merits are discussed in the literature, but they also exhibit problems, and the design or adoption of a taxonomy which is adequate for CS is still an open issue. A past ITiCSE working group report contains a good survey of the topic [47].

4.1.4 Learning styles

Students engage with their individual learning processes in different ways. For example, some students may want to actively discuss learning materials with their colleagues, whereas others may prefer to read material on their own. These differences are generically known as *learning styles* and much work has been done to identify both how to articulate what styles might be, and how engaging with different styles might positively affect a student's learning.

There are several models of learning style, which we only briefly mention here. The reader is advised to look at the references to find more information. Perhaps the most frequently cited is Felder and Silverman's Learning Style Theory [45] which classifies a student on four scales: active vs. reflective, sensing vs. intuitive, visual vs. verbal and sequential vs. global. Gardner's Multiple Intelligences [49] uses even more dimensions: linguistic, logical-mathematical, spatial, bodily-kinesthetic, musical, interpersonal, intrapersonal and naturalist.

Kolb's Learning Style Theory [70] employs a four-stage cycle, as depicted in Figure 2. He proposes four learning styles—diverging, assimilating, converging and accommodating—based on the stages in the cycle with which the learners principally engages.

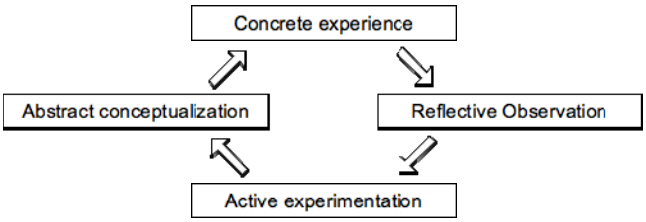

Figure 2. Kolb's learning cycle.

The Dunn and Dunn Learning Styles Model [42] contains five components that influence a student's learning—their personality, their emotional and physical state, the physical environment, and social factors (such as their teachers and peers).

In the computing context, recent work suggests that active engagement [9, 112, 119] is an effective strategy that suits many learners' styles.

There are two important issues to remember when considering learning styles. Obviously, differences between individual students will affect the efficacy of a CALMS, especially if the material contained within it focuses on a particular genre. However, learning styles are not static properties of people. The same people may use different learning styles depending on the context and their motivation. They can also develop their skills in the dimensions where they are weaker—the CALMS might even be designed to support this. As an overall observation we conclude that presenting content in many different ways is an advantage because it supports many different learning styles.

4.1.5 Motivation

Motivation positively affects a student's engagement with their learning experience, which is highly relevant for computing students. Active engagement in the student's learning process [58] and a rich source of material to support their self-guided study [114] are both important motivating factors, and the availability of such resources online is valuable [53].

Motivation can be classified in two ways. *Internal motivation* grows from the learner's own interest in mastering content or skills, whereas *external motivation* is based on reaching some goal, such as getting a grade or high mark, or avoiding a punishment.

4.1.6 Collaborative learning

Collaborative learning has been evidenced as effective in promoting students' higher-level cognitive skills [108]. Srinivas [142] summarizes 44 benefits of collaborative learning. Lehtinen *et al.* provide a review of Computer Supported Collaborative Learning [77].

Vygotsky [147] argued that the basic mechanism of cognitive growth is communicative in nature. In the *zone of proximal development*, the contradictions, inconsistencies and limitations

within a learner's explanation become apparent when the learners need to communicate their thoughts to others—the learners need to perceive their conceptualizations from a different point of view. When several learners face the same situation, communication forces them to articulate their conceptions, which they can then compare with their peers', and therefore groups of learners tend to perform better than those working alone.

Learning environments should support deeper cognitive actions, but this is not always the case. Sometimes students manage and even succeed without thinking deeply. For example, some students use automatic assessment tools to support a trial-and-error approach to solving programming problems. They do not think through the debugging process, but use feedback only as an indication of correctness/incorrectness (thus being driven only by external motivation).

Kitcher [69] and Dunbar [41] have shown that cognitive division of labor is an important prerequisite for scientific advance. Distribution of cognitive effort supports group flexibility, thus achieving better results. Moreover, when the members' expertise is different but overlapping, the results are better than in homogeneous groups, suggesting that learning environments should support collaboration and should be used as parts of collaborative working methods. Knowledge-seeking inquiry, such as setting goals, identifying research questions, looking for new information, and providing explanations are actions that could be included in the learning process and supported by the environment. Thus, sharing documents, and support for joint reflection and discussion, are valuable attributes of an advanced learning environment. Note that even in the absence of these features, the learning environment can effectively be used collaboratively by enabling pair or small group investigative and reflective activities [96]. For example, it has been argued that LOGO is an important pedagogical tool because it can encourage and facilitate student collaboration [59, 60]. More modern approaches such as those taken by Alice [31] and Greenfoot [52] support similar processes. Few programming education tools have been specifically designed to support collaboration; AlgoArena [140] is an exception in introductory programming.

Even though collaborative working has many advantages, it is certainly not a silver bullet. Groupware is successful only if there is a real need for collaboration among the learners. Moreover, learners should clearly understand how the software can support collaboration. Finally, the collaborative paradigm must be embedded within the whole educational culture, and any conflict between support for collaborative learning and rules for avoiding it (to prevent plagiarism, for example) has to be dealt with.

4.1.7 Organizing learning content

Learning environments can broadly be classified as *open* or *closed*. In closed learning environments, the learning content is more or less prescribed by the teacher, and often there is a detailed path through which the learner must proceed in order to achieve the learning goals set by the instructor. The learner should therefore be aware of the goals. Such environments are useful for practicing specific skills or assimilating certain material, such as learning to use some piece of software or gadget. A "guided tour" of an item of software is a typical example of a closed learning environment. Closed environments may emphasize external motivation and often restrict learner's initiatives.

In contrast, open learning environments allow the learner to select the content and methods that they find appropriate at each stage of their learning. Consequently there is no specific path to follow, but different learners may use the environment in different ways. It is important that the learning content not be restricted, rather that the learner can spontaneously start to follow some path to find more information about a topic they find relevant. Thus, we could say that the environment supports internal motivation, which emerges from the learner's own needs and learning preferences. Jeliot 3, for example, allows the learner to explore the working of different Java programs, which the user is free to modify.

Open and closed environments set different expectations both for learners and instructors. When designing a closed environment, the teacher defines the relevant contents and learning paths, and the student's role is to more or less passively follow the instructions given by the teacher. In open learning environments, the learner is free to explore what they consider to be interesting and relevant, requiring the learner to be responsible for their own progress. The teacher's role is thus more like a coach and facilitator who provides relevant content, helps the learner to make decisions about how to proceed, and provides additional resources when needed.

Open learning environments support creative problem solving and individual learning better than closed environments, and are firmly grounded in the constructivist approach.

Designing closed learning environments is closely tied to defining learning goals and a detailed instructional process. In contrast, in open learning environments the teacher cannot set up fixed paths for browsing content and carrying out learning tasks—the environment can be organized in many different ways. There are, however, several common models that are widely used in both structuring the content and directing the learner interaction when using the environment. These models work as metaphors.

In the *market* model the metaphor is the market square. The learner is walking on the square and explores what is available in small shops, but they can visit them in any order. In practice, the learner is provided with an environment in which they are free to select among many different actions and content to explore whatever they find interesting. For example, MatrixPro allows the learner to interact with different data structures in any order with no preferences—the learner can explore the behavior of data structures freely or solve exercises and get feedback. Many educational games designed for children work using the same metaphor. In one such game, the learner is walking through a house in which the entrance room provides access to many different rooms with different contents and tasks. When such a learning environment also allows the student to add new material into the environment, it follows an *open market* model [88].

The *theater* model emphasizes that it is not primarily a tool but a medium in which the user modifies concepts that may not correspond with external reality [74]. The emphasis is on actions the user is carrying out, not on objects and their relations. The environment stimulates imagination and

experimental activity, and the learner can "direct the show" by considering different alternatives. For example, in Jeliot3 the visualization acts as a scene, the Java program is the manuscript, and the learner can modify the manuscript to see what kind of effects it causes in program execution.

The *story* model is based on the fact that humans have a long history of learning by telling and listening to stories. The environment is designed based on what the learner would experience in real life, and thus learning content could include imitated authentic materials and tasks—not dissimilar to problem based learning. Visualization and comparison of structures often have an important role in this model, supporting reflection and allowing the learner to actively relate new information to their previous knowledge.

Finally, the *object* model originates from a more practical issue of re-usability. The learning content is organized as relatively independent chunks, called learning objects (LOs), which can be reused in different contexts. This emphasizes the need for meta-data to describe the LO contents. Current research in educational technology has put much effort into developing standards for LO metadata, such as SCORM [2].

4.2 Practical Considerations

The theoretical discussion above should be complemented by alerting the reader to a number of practical issues.

4.2.1 Legal, social, ethical and professional issues

These issues are frequently overlooked, and in many straightforward educational activities may not be a central concern. However, the requirements of the professional accrediting bodies (such as the ACM/IEEE-CS and the BCS) include consideration for legal, social, ethical and professional issues. This is relevant for the construction of CALMSs in many ways.

First and foremost of all, the content of a CALMS must be legal. There are, of course, many laws that may relate to a CALMS, but some of the most important include the following.

- Discrimination, accessibility, and social exclusion—the material presented must not disadvantage any student who is likely to use it, and must not contain any content that might be perceived as such. Discrimination may relate to gender, sexual orientation, ethnic or racial background, age, or disability. In the EU, human rights legislation may also be relevant.
- Data protection and privacy—access to the data within the CALMS must be restricted to appropriate persons, since much of it is "personal data" within the defining framework of EU legislation.
- Export and import of technologies with security implications—a sensitive issue within the US. Hence a CALMS containing security related content may require scrutiny.
- Intellectual property—if material from a third party is used, appropriate permission must be sought and obtained.

The legal requirements vary between countries; if in doubt advice should be sought.

Social issues, beyond those prescribed by law, are also important and, depending on the student cohort likely to be using the CALMS, might include consideration of the economic and cultural background of the students. Content must also be ethical, and might include, for example, an awareness of environmental issues.

Finally, material must follow "good practice" as defined by the professional bodies. The most often encountered aspect of this is plagiarism prevention.

4.2.2 Plagiarism

Plagiarism—the unauthorised and uncited use of another person's work—is a persistent problem in academia. Much effort has been invested on work into the detection and prevention of such activity, both in free-text and program source code [64, 150]. Substantial resources are available online to assist teachers in this task [100].

This becomes an issue when use of a CALMS includes assessment (whether formative or summative). However, careful design and authoring of material can minimize the problem. For example, tasks can be individualized to each student. Re-use of material that exists elsewhere (on the web or in textbooks) should be avoided.

4.2.3 Tracking student activities

A feature of LMSs is the ability to track, in detail, all interactions with the system. This is both a blessing and a curse.

In principle, student tracking data should enable the teacher to profile how the system is being used, for example, by identifying sections of the environment infrequently or inappropriately being engaged by students. Prediction of student performance or, at least, of their learning profiles is another application of student tracking. Student tracking is also used in collaborative learning systems to log and analyze students' contributions and interactions.

Students need to be aware of how the data they have generated by interacting with the LMS are used. If the data is used to assess students, then that needs to be transparent, since there is a risk of alienating students if they do not clearly understand the process. Furthermore, the use of that data within the EU is also regulated by data protection legislation.

Use of student data will also affect how students interact with the system, notwithstanding any assurances that may have been given by the instructor. Conscientious students will try to please, even if they are not formally assessed.

A further problem with gathered data is that it is often unreadable (e.g., as log files) or too simple (e.g., statistics charts).

4.2.4 Documentation requirements

In a CALMS, documentation contains either technical information required by system developers and/or content managers , or pedagogical information.

Although a teacher may have developed activities for their students, which are reasonable and worthwhile, they may be required to justify those decisions to third parties. A primary focus for such justifications would be mapping the learning outcomes of a course to the detailed activities, particularly to the assessment components, within the course. These may be

required by the institution or by external agencies (in the UK, the Quality Assurance Agency requires such data to be accurately maintained).

4.2.5 Mobile learning

Mobile learning is e-learning with additional capabilities and limitations, delivered through devices such as mobile phones, Smartphones and Personal Digital Assistants (PDAs), Pocket PCs, or Palmtop devices [83]. More formally, mobile learning is "Any sort of learning that happens when the learner is not in a fixed, pre-determined location, or learning that happens when the learner takes advantage of the learning opportunities offered by mobile technologies" [102].

A CALMS may involve delivery on mobile devices. The principal technical advantages of mobile devices include their portability, small size, connectivity, rich functionality, and low cost. These advantages may be offset by the small screen and keyboard, possibly limited functionality, lack of robustness, high risk of being lost, difficulty of upgrade and expansion, and expense of connectivity [81].

Pedagogically, a mobile approach may be beneficial since it can maintain student motivation [95], adapt to students with visual or physical handicaps or particular learning styles [94], and be used in a variety of learning situations, including collaborative [57] and independent learning [23, 107]. However, there are a number of concerns that need to be addressed [72]. Some of these concerns concern personalization, since the small size of mobile devices requires material to be targeted as much as possible [105].Another concern is the functionality and limitations of the target devices, since general e-learning design requirements may not scale [106].

O'Malley *et al.* [102] have produced a set of guidelines for m-learning and m-teaching.

4.3 Introduction to Pedagogical Patterns

Design patterns were originally introduced in the 1970s as an architectural qualitative method by Christopher Alexander and his colleagues [5]. In the mid-1990s, Erich Gamma and others successfully adapted Alexander's pattern methodology to the domain of computer software design [48]. In the broad computing field, many authors continue to use modified versions of the qualitative Alexandrian method [34]. In particular, modernized Alexandrian pattern structures have been successfully adopted in pedagogical pattern design [14, 110]. Recent guidelines [117] stress the need to find a working balance between abstraction and concreteness. Recent guidelines also stress design patterns that serve as working tools—rather than general abstractions.

Since the mid 1990s, patterns have become a central theme in the computing community. In mid–to–late 2008, when this report was being written, a Google Scholar advanced search produced an impressive list of 9,140 publications on pattern language(s) in computer science, engineering, and mathematics. Patterns have been the focus of well-established international conferences, such as the prestigious OOPSLA and ECOOP conferences, the specialized PLoP conferences [85], and a number of regional conferences, such as EuroPLoP, KoalaPLoP, and VikingPLoP. Patterns related to education are frequently in the program of the SIGCSE and ITiCSE conferences. The Hillside Group's website [56] 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 market.

4.3.1 Pedagogical pattern languages

A collection of related patterns is called a pattern language. In the past, many such pedagogical pattern languages have been proposed. We review some of them.

The Pedagogical Patterns Project PPP [110] has produced a broad collection of pedagogical patterns [15-17] that are oriented towards the traditional classroom setting. The patterns are categorized in different groups, such as "Patterns for Gaining Different Perspectives", "Active Learning", "Experiential Learning", and "Feedback Patterns". These patterns are mainly "tips and tricks" (e.g., "Teach the most important topics first!", or "Find a complex and consistent metaphor for the topic being taught!") and are not related to electronic learning environments. For some patterns, however, the pattern can best be used if it is technically supported. So, for example, the pattern "Grade It Again, Sam" ("Provide an environment in which students can safely make errors and learn from them by permitting them to resubmit previous assignments for reassessment") is easier to implement when an automatic assessment system is available.

Another pedagogical pattern language has been developed by Vogel and Wippermann called "Didactic Design Patterns for the Documentation of the Modes of Teaching and Learning in Universities" (in German) [146]. In contrast to the pedagogical pattern project, this pattern language is especially focused on e-learning settings. The pedagogical patterns are grouped into the following categories:

- pre- and postprocessing of course sessions,
- presentation (e.g., tele-teaching or the use of hypertexts and animations in a course session),
- communication and cooperation (video conference group discussion, moderated expert chat, teamwork supported by a learning management system, forum, e-mail, chat, audio/video conference, and wiki) ,
- evaluation (feedback discussion by using an electronic questionnaire and/or a forum).

In contrast to the patterns of Vogel and Wippermann, which are supposed to help the teacher use different teaching settings, the pedagogical pattern language by Schümmer and Lukosch [82] is meant for supporting the communication between users and developers of learning management systems. The patterns are mainly features of a learning management system and use cases. They are arranged in the following way:

- creation of the learning group (e.g., objects or places for collaboration are locked by a password, users comment on the quality of contributions of others),
- base technology (e.g., a server notifies about status changes of shared data, changes data optimistically and undoes the operations if users performed conflicting changes),
- building the community (e.g., enroll in a course, the teacher takes the supervising role in a learning environment), and

- group support (e.g., several users edit an object in parallel, quick check of a group's opinion on a question, conference with audio and/or video support, teacher spreads news on the course, embedded e-mail, students work on assignments and submit them to the teacher for grading, online tests and quiz, students review other students' assignments).

4.3.2 Patterns for active and cooperative learning

Recently there has been increased interest in alternative learning styles, such as active and collaborative learning (while at the same time raising questions about the suitability of traditional lecture and classroom pedagogy). In general, active and cooperative learning are umbrella terms that refer to a number of instructional models, most of which can be effectively supported by CALMSs.

Qiu, L. and Riesbeck [112] wrote "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 advanced computing courses". Cooperative learning is active learning in a group [86]. Active and cooperative learning patterns are of special interest in the light of the increasing popularity of CALMSs.

Patterns for active and cooperative learning have been published as early as 1995. Anthony [8] offers patterns for simulation games and quiz games as part of his general purpose patterns. Most notably, Bergin and others have collected patterns for experiential learning [17] and patterns for active learning [16] at various levels in various disciplines. These general patterns are focused on a more traditional classroom environment and are not particularly dedicated to e-learning environments and techniques, nor do they address computer science learning specifics.

Recently, CS educators have experimented with—and published—a number of active/cooperative learning models that are exclusively based on the use of CALMSs. It will be beneficial to formalize the following models as pedagogical learning patterns.

- The *student submissions* model [116]. The instructor poses a question that all students answer online.
- The *abductive learning* model [112]. Abduction is a reasoning process that starts with a set of specific observations and then generates the best possible explanation of those observations. Online "study packs" stimulate students to learn abductively by browsing, searching, and performing self-guided lab experiments.
- The *feedback loop* model [9]. Just-in-Time Teaching (JiTT) is a teaching and learning strategy based on the interaction between web-based study assignments and an active learner classroom. The essence of JiTT is the feedback loop formed by the students' preparation outside the classroom that shapes their in-class experience.
- The *daily worksheet* model [22]. The instructor administers ungraded paper quizzes each day on which students may collaborate.
- The *gated collaboration* model (described in Section 3.1). The LMS poses a question; students answer, and then review their classmates' responses.

Active e-learning patterns are of special interest in the light of the increasing popularity of e-learning environments.

For CALMS designers and implementers, active and cooperative learning patterns will be a resource of promising new CALMS features. For CALMS authors, these patterns will provide formal background for sound active learning course design.

5. TECHNOLOGICAL GUIDELINES

So far, we have reviewed existing learning resources generated for, or well suited to computer science education, followed by a review of basic pedagogical aspects for creating a well-designed learning system. In this section, we will complete the analysis of required aspects for a CALMS by examining technological aspects to be considered for a CALMS or a simpler computer science education support system.

A software developer who has written a system designed for one of the particular CS-specific instructional needs described in the "Learning Resources Overview" section of this report may be able to encourage the use of your system by other CS educators by making it interoperable with one of the commonly used LMSs. Although the system will still focus on its particular instructional goal, it will also benefit by increasing its awareness of the larger instructional context in which the LMS is using it. It is beyond the scope of this report to develop details for standards and APIs to which such plug-ins should adhere. Nonetheless, there are general guidelines that we envision will facilitate the efforts to turn a particular application into such a plug-in. These include guidelines for platform independence, licensing, dissemination, data exchange, security, internationalization and customization, and making the system compatible with other resources that are often used in instruction on that topic.

5.1 Platform Independence

Platform independence is an important issue when developing tools that will be distributed to institutions other than the developer's own institution. Moreover, computer science departments use a wider set of platforms than other departments. Thus, it is advisable to extend an LMS using technologies such as Java and open web technologies such as HTML and Javascript.

Most current CS specific learning tools have been developed in Java. Java's only requirement is that the user's computer has the Java virtual machine installed. Java technology also includes Java Web Start, which allows starting Java applications from a web page.

5.2 Licensing Issues

Just as platform independence can be a factor in determining whether instructors are able to integrate the resource into an LMS they use for their class, the licensing terms under which the software is distributed and the additional media resources it

uses are important. We saw in our discussion of the survey results in Section 2 that some respondents expressed distaste over using proprietary software systems and resources in which content would not be open to global web-based search tools. Open source development is worth considering, as it fosters external collaboration, often resulting in more comprehensive software developed by educators who have neither the time nor inclination to become involved in proprietary software development. This is precisely the reason why so many widespread CS tools are distributed for free.

If that software then accesses and manipulates hypermedia materials, there may well be copyright concerns when the software moves from being a special-use application to being a plug-in for a large-scale, widely used LMS. The authors of those hypermedia materials need to explore their options for copyrighting them in a fashion that suits their needs and desires. The Creative Commons project [30] provides a wealth of information on copyrights.

5.3 Dissemination

If nobody knows about a tool, it will not be used. However, the issue of making a tool known can include a variety of issues. Registering it in appropriate repositories of learning materials, and publishing results and other information in scientific and/or educational forums is one thing to consider. Providing a supporting web presence in which the authors include metadata so that increasingly sophisticated search engines can gather information about your tool, however, is also of increasing importance. The latter is a more technical issue as it requires not only ontologies to describe the content, but also metadata formats to deliver the descriptions. Putting the material in an XML format should help to address these issues.

5.4 Data Exchange Between the LMS and Specialized Systems

The range of data that various LMSs could provide to your system upon start-up is large. It could be as simple as a mere token that identifies the learner using the system. However, more likely it will include information about the learner's past activities. Those past activities have, through the lesson structure authored by the instructor in the LMS, ultimately led to this particular learning exercise, which is now calling upon the specialized resource. The degree to which a resource can adapt itself to the contextual information it receives from the LMS will become increasingly important. For example, a resource may be given information indicating that the learner has already demonstrated, from past activities, a "pretty good" understanding of this topic, or it may be told that past activities have indicated this learner is "completely lost." In such situations, the system may be able to direct these learners to different content that is better suited to their individual needs.

Conversely, when the learner has completed an activity with a given system, there will probably be information that the system must convey back to the LMS. For example, in our analysis of the results collected from our survey, we noted that automatic assessment is one of the features of LMSs most valued by the educators. Consequently, if the system cannot carry out an assessment of how well the learner fared in the activity and then report the results of that assessment back to the LMS, its value to the instructor is diminished. In some instances, the data interchange involved in this assessment of the completed activity may be relatively simple, for example, how many points the learner scored out of a perfect score for the activity. However, for other activities, this assessment data could be quite complex, perhaps including a complete log of the learner's interaction with the system during the activity.

Because the data interchanged between the LMS and the system will become increasingly complex and important, standards are emerging – and will continue to do so – regarding how the data interchange should be done. For example, the Sharable Content Object Reference Model (SCORM) [2] is a collection of standards and specifications for web-based learning that has begun to address these issues. Whether one should use a standard as comprehensive as SCORM or develop a simpler convention for data exchange protocols that are more uniquely suited to the particular needs of an application is a factor that system developers will have to consider. As such protocols for interchanging this information mature, they will often be represented in XML, and new programming APIs will emerge to facilitate software development efforts in this regard.

5.5 Security

One particular challenge for LMSs is the development and maintenance of an infrastructure for user authentication and authorization. When the LMS and the corresponding integrated modules are disseminated to other institutions, inter-organizational authentication and authorization become an issue. Fortunately, some local projects exist to solve this issue. For example, the HAKA infrastructure (Federation) is a group of organizations founded by Finnish universities and polytechnics which cooperate in this area. The purpose of the Federation is to support higher education and research institutions by developing and maintaining an infrastructure for user authentication and authorization. The TRAKLA2 system [84] is one of the services that can be accessed through the HAKA authentication mechanism. Any student in any Finnish university can use his or her local account and password, provided by the home university, to log into the system (as well as services of other institutions that have joined the federation). The HAKA infrastructure is entitled to collaborate also with cooperating foreign federations, but we believe new initiatives are needed to make the infrastructure available worldwide.

5.6 Internationalization and Customization

Developed tools should be designed to accept and produce content in different languages. The graphical user interface and user messages should be implemented in a way that different languages can be used to display textual information. Java property files or GetText tools can be used for that purpose. If textual information from the user is processed, the tool should be aware of different text encodings.

The survey results from Section 2 indicated that one of the adoption blocks of current CALMSs is "missing, incorrect or insufficient content". Thus, tools should provide ways for teachers to create and customize material in the tool that will match their other teaching materials, such as text books.

5.7 Compatibility Issues

The contents of the tool and the tool itself should be mapped to existing teaching and learning resources, such as the ACM

curricular guidelines, Bloom's taxonomy, the SOLO learning.For example, ACM curricula can be used to break down the content into meaningful topics that are universally known. An instructor looking for new materials can then better judge whether or not the material covers his or her needs. Taxonomies can provide a way to rate the difficulty of the exercises. This could help to construct adaptive learning materials in which an exercise can be selected based on the student's previous knowledge. For example, the difficulty of the exercise can be lowered in terms of Bloom's taxonomy until the student is able to solve the exercise. In addition, both the break-down and the taxonomies can be used to label the metadata for the material (see the other item below). The bottom line is that the material should be in a form that attracts the instructors. This also includes uniform terminology with existing text books that the instructors already use. Customization is one issue that needs to be taken into account, but if the tool already uses the same conventions that the course text book does, it will be easier to use.

5.8 Use Best Practices and Guides to Develop Tools and Materials

Many products developed for CS education are potentially valuable and technically sophisticated. However, they must also be grounded in sound principles. Some of these principles come from CS, but some other principles may also come from other disciplines.

An example can be found in algorithm animations. There are a number of sophisticated and powerful tools and systems available, but they do not give hints about how to produce educationally effective animations. Fortunately, we can find several kinds of aids.

In the first place, we find "best practices"—recommendations distilled from past experience, including successful and unsuccessful experiences. Pedagogical visualization draws on many related disciplines, including typography, psychology, and algorithms. Consequently, some general recommendations can be inherited from typography. However, most best practices have been developed within the field of algorithm animation. Naps *et al.* [99] summarize eleven general suggestions.

We may also find guidelines not only grounded in experience, but also on theory. Velázquez-Iturbide *et al.* [145] warn that recommendations can hardly be found about how to structure an algorithm animation. Consequently, they develop a "pedagogical guide" that gives advice on the following key elements of algorithm animations: number of animations, size and structure of each animation (including important steps of the animation and level of detail in different stages of an animation), and size and value of input data. The guide is inspired by two different fields: computer science (mainly, program testing) and human factors. Program testing principles provide a basis for identifying the main elements of an animation, whereas human factors are taken into account to decide whether these elements are worth keeping or discarding.

6. EXAMPLE CALMS SCENARIOS

As discussed in previous sections, current general-purpose LMSs do not adequately support some specific needs of CS education. In particular, they do not incorporate CS-specific learning activities that are successfully supported by specialized CS education tools like those included in the Section 3. After reviewing the aspects to be considered in building a CALMS in Sections 4 and 5, we now give some scenarios of demonstrating the kind of learning experiences we envision a CALMS should support. These shall provide the final motivation for actually building a CALMS, by outlining learning and teaching possibilities that are usually difficult to achieve in most current systems. Each Section will briefly describe the scenario, provide a motivation and a proposed solution, followed by a discussion.

6.1 Systematic Assessment

Scenario: Incorporate a systematic assessment capability within an LMS, which allows assessment by the student, the instructor, peers, and automatic tools, to be applied to all assessment activities for a learning unit.

Motivation: Assessment of student performance is one of the most critical functions of LMSs according to the survey conducted by this working group (see Section 2).

Popular LMSs offer several forms of assessment, including automatic assessment, instructor assessment, peer assessment, and self-assessment [7, 84, 87]. Alas, assessment functionality is usually developed on an *ad-hoc* basis and varies from activity to activity. In Moodle, for example, quizzes permit only automatic and instructor assessment, while forums allow only peer assessment (see the extended Moodle example in Table 1).

This is a problem because it makes it difficult, if at all possible, to implement systematic assessment across all activities in experimental or production courses.

Proposed Solution: Integrate systematic assessment functionality into each CALMS activity (see Table 2). Note that automatic assessment can incorporate plagiarism detection in all activities, and that all forms of assessment can provide grading.

Table 1. Existing assessment functionality in Moodle

Activity / Assessment	Automatic	Instructor	Peer	Self
Quiz	✓	✓		
Assignment		✓		
Forum			✓	
Workshop		✓	✓	✓
Database			✓	

Table 2. Proposed systematic assessment functionality

Activity / Assessment	Automatic	Instructor	Peer	Self
Quiz	✓	✓	Possibly add	Add
Assignment	Add	✓	Possibly add	Add
Forum	Add	Add	✓	Add
Workshop	Possibly add	✓	✓	✓
Database	Add	Add	✓	Add

Discussion: Self-assessment of quizzes can be particularly beneficial for essay and file-submission questions. Peer assessment of quizzes and assignments can be beneficial, but it may create logistic problems in practical courses with students who are late with their submissions. Automatic assessment of Moodle workshops may be beneficial, but its design and implementation can be a serious challenge due to the inherent complexity of workshops.

For e-learning researchers, systematic assessment functionality will permit experimental investigation of the pedagogical potential of innovative assessment techniques, such as self-assessment of quizzes or automatic assessment of assignments. For course designers, systematic assessment will open opportunities for implementing assessment procedures that save instructor time and increase student motivation.

6.2 Automatic Assessment of Programming Exercises

Scenario: Combine LMS, peer reviewing, and automatic assessment of programming exercises.

Motivation: When doing a peer review of programming assignments, the reviewers should be able to view some of the results of automatic assessments, so that they do not have to comment on breaches of prescribed programming styles or conventions. They can then use feedback from the automatic assessments and focus a deeper discussion on other aspects, such as design issues.

Students only have to use a single system (instead of multiple tools)—they have a "one-stop-shop" for accessing the learning material, submitting their solutions to programming assignments and performing the peer review. A similar statement holds for the teachers. The specialized learning management systems described in Section 3.3 only provide automatic assessment, but not peer reviews of the submitted systems.

User management only has to be done for a single system, and students do not have to register multiple times.

Proposed Solution: Incorporate an assignment management system as part of the CALMS, which allows for the creation of assignments with defined automatic assessment procedures and/or peer review.

Discussion: Superficially, this may appear to be a simple problem. However, development of LMSs, automatic assessment tools (such as CourseMarker [55] or BOSS [65]) and peer assessment tools (such as WebPA [149], Bess [18] and CAP [32]) have progressed in parallel, and integration has not taken place. Each such tool duplicates, in part, the content management functionality of an LMS. Principal reasons for this include differences in the underlying data storage requirements and in the technologies used to develop the tools, and the closed nature of some LMSs, which inhibits detailed development work from taking place. Moreover, tools which work with a given programming language, such as Java, may not be modified easily to work with other languages, preventing wide adoption of the tool. A further issue is the uptake of tools within the academic community—although various assessment tools are in use, there are different approaches to automatic assessment, and the pedagogies have not yet been evaluated in any detail.

The lack of integration has a further consequence. Since the code base for each of these assessment tools is large, the effort required to extend the tool to include further pedagogical benefits, such as support for different learning styles, is also high.

A well-designed integrated environment will provide a framework for developing educationally rich tools that will also allow for easy deployment and effective evaluation. In the context of Computer Science, a further benefit would be the simplification of support for multiple programming languages and paradigms.

A good realization of this scenario can also touch upon several of the pedagogical aspects discussed in Section 4: it concerns the instructional process (see Figure 1), can raise the students' motivation (see Section 4.1.5), and offers collaborative learning (Section 4.1.6).

6.3 Collection of Assessment Results

Scenario: Support the incorporation of assessment materials, and the results of the interactions of students with these materials, into the LMS data store. This requires an API that enables the creation of new tools or the integration of established tools into the LMS.

Motivation: There are numerous tools that provide assessments of various kinds. Examples include JHAVÉ [97], which allows the author of a program visualization to ask questions of the student at each step of the visualization, and TRAKLA2 [84], which administers exercises relating to data structures and keeps track of how many exercises the student has correctly completed. The problem is that an LMS is not able to provide information to the tool to guide the authoring of exercises for students.

Proposed Solution: Produce an API (application programmer interface) for each pedagogical tool that provides some metadata for the LMS and describes its input and output, and provide a plug-in for the LMS that supplies appropriate input and also records the resulting output consistently with the other data the LMS maintains about each student. Input to a tool might consist of some authoring information. Output would describe results of the assessments.

Discussion: There would be a substantial one-time effort required, since tool builders and LMS developers would need to collaboratively design and implement APIs.

6.4 Adapting to Teaching and Learning Styles

Scenario: Include a facility for content delivered within an LMS to adapt to students' learning styles.

Motivation: According to learning style theory (Section 4.1.4), students can be classified on a variety of scales, for example active vs. reflective, sensing vs. intuitive, visual vs. verbal and sequential vs. global. In practice, most learners' preferred styles do not fall exclusively into just one of these categories; rather, each learner adheres to a combination of the characteristics of the different learning styles. On the other end, instructors can

strive to choose a mixture of teaching styles that provide the best possible match to their students' learning styles.

According to the survey conducted by this working group (see Section 2), current mainstream LMSs are rigid and difficult to adapt to one's preferred teaching style. In addition, LMSs lack the capability to flexibly adapt to each student's learning preferences.

Proposed Solution: Systematically extend LMSs with learning and teaching style capabilities.

Discussion: Recent research on adaptive hypermedia has been addressing the need for adaptation in learning environments. One proposed system [143] is able to rearrange the order of pages (potentially also dropping individual pages, which are determined to be inappropriate for the given learner). However, experimental use of this approach has shown that the reshuffling of pages, while comparatively easy to do mechanically, becomes highly difficult to support for the content author. The reason for this is that in a simplified model, the author will not know which pages(s) the learner has read when they access a given page, making it very difficult to ensure that all relevant previous knowledge has been presented to the learner, or that the learner is spared from tedious repetitions of already known materials.

Instead of this simple reshuffling, more complex reordering may include pedagogical links between pages, for example to specify that page A explains the details of page B, or that page C has to come before page D. In this case, the system can infer a partial ordering of the materials to better ensure that what the given student can see makes sense.

The L4 system [1] provides a strategy editor, which allows the content author of a given course to specify such partial orderings and relationships between course entries. Additionally, it also provides a strategy template editor, with which the author can specify a template of which elements a course using the concrete template should have. For example, a template for visual learners could prescribe that the materials always start out with a (hopefully inspiring or thought-provoking) image or video about the topic under discussion, followed by a definition and an example with another visual element.

Another system, called SHALEX (Structured Hypermedia Algorithm Explanation) [68, 135], addresses several of the aforementioned problems. It provides novel features, such as reflection of the high-level structure of an algorithm and support for programming the algorithm in any procedural programming language. By defining the structure of an algorithm as a directed acyclic graph of abstractions, algorithms may be studied top-down, bottom-up, or using a mix of the two. It is also possible to support several levels of abstractions, which help the learner understand basic properties of an algorithm as well as to recognize good implementation strategies. Moreover, SHALEX supports many algorithms by using a taxonomy of explanations, which has a tree-like structure. Non-leaf nodes of the taxonomy represent concepts, such as "iterative algorithms" (the root of the tree represents the set of all algorithms). Leaves represent explanations of specific algorithms created by specific authors.

An alternative approach is to provide less flexibility in the pages themselves, and simply write multiple views of the materials for different target audiences. For example, Ross' hypertextbook *Snapshots on the Theory of Computing* [121] offers three different routes through the learning materials, geared for beginning, intermediate, and advanced students. A similar approach could be taken for more visually-oriented students or students who learn better from examples than from definitions.

6.5 Tools That Specialize Generic Tools to Programming

Scenario: Connect an integrated development environment (IDE) to an LMS via a plug-in.

Motivation: One principle for choosing the tools to be used in a course is to pick tools that are used in the real world. Currently, in programming, such tools include integrated program development environments (IDEs). Generic LMSs suffer from the lack of access to such programming environments.

Proposed Solution: Ensure integration of IDEs into LMSs that enable communication flow in both directions (from the LMS to the IDE and vice versa). Several IDEs, BlueJ [13] and Eclipse [43] in particular, allow *plug-ins*, extensions that increase the IDE's functionality.

Discussion: A scenario like this would enhance several of the tools found in a typical LMS:

- Tracking of student activity. At present, students typically switch out of an LMS to do their programming; activity done in an independent IDE is invisible to the LMS and thereby inaccessible for the purposes of gathering data about student learning. Important information to be recorded would include movement from one program component to another (e.g., to gauge student strategies for program understanding) and program edits and compiles (e.g., to detect episodes of counterproductive programming behavior [63] or to monitor application of test-driven development).
- Collaboration and peer review. LMS events can structure a pair-programming activity, reminding students of their designated roles. A student reviewing a classmate's code may wish to run the program; with an IDE immediately accessible, it would be easy to do this and record the results. An IDE would also enable review comments to be more structured, e.g., via automatic linking to program components (methods or variables). The program GREWPtool [54] is an IDE that allows pair programming—*collaborative coding*. It has proven to be valuable in a variety of class activities involving interaction among students and between students and instructor.

Other tools, once coupled with an LMS, could significantly enhance the experience of learning to program. Some examples include the following:

- A utility that intercepts compiler error messages and translates them into more understandable information. Two examples are Expresso [61] and DChk [36]. Both are currently implemented as preprocessors to a standard Java compiler. The authors of DChk mention upcoming efforts to integrate it into Eclipse.
- More ambitious tools that maintain a model of the user and function as personal tutors. Two examples

are Java Critiquer [112] and JITS (Java Intelligence Tutoring System) [141].

- An editor not only for code but for higher-level constructs, such as goals or plans as implemented in the GPC (Goal/Plan/Code) Editor developed by Elliot Soloway and his colleagues [137].

6.6 Integration with Visualization

Scenario: Incorporate algorithm visualization tools into an LMS.

Motivation: As stated by one of our survey respondents, there is usually no support for integration of AV into LMSs. On the one hand, the features that the respondent missed in most AV tools included integration with the maintenance of student grades and import/export with the university's registration software. These are all features that LMSs are supposed to provide. On the other hand, the respondent also missed features such as automated assessment and feedback on exercises involving AV. However, general purpose LMSs have very limited assessment capabilities with respect to their interaction with AV. For example, JHAVÉ's pop-up questions and MatrixPro's visual simulation of an algorithm's execution are something that cannot be done without a special purpose AV tool. Thus, we believe the future trend will be to integrate AV tools to LMSs in such a way that the enhanced system can seemingly provide all the features through a single system.

Let us assume that a student could get a deeper understanding of the working and behavior of finite automata with the help of an explorative visualization. The visualization offers the visual generation of a finite automaton in the form of a transition diagram on the basis of a regular expression that can be entered by the student. Furthermore, the student can enter an input word for the generated finite automaton to watch its acceptance behavior. The GaniFA tool [38, 40] is an example of such an AV system. An LMS can be used by the student for learning the fundamentals, such as the definition of regular expressions, the algorithm for generating a transition diagram from a regular expression, and so forth. The challenge is how to combine these two worlds effectively.

Current AV tools, as described in Section 3.4 and 3.5, are more or less stand-alone systems, such as GaniFA or JFLAP [118]. Their integration into an LMS is done by adding a link that starts the visualization, for example in a separate window. When should the LMS offer the student such a link? The student must have a basic understanding about the fundamentals. Only then can they profit from using the AV tool. Conversely, how can we measure the learning progress during the exploration of the visualization? This information should be communicated back to the LMS by the AV. To the best of our knowledge, those functionalities are not offered by current LMSs.

Proposed Solution: Enforce integration of algorithm/program visualizations into LMSs that support assessment and communication flow in both directions, that is, from the LMS to the AV tool and vice versa. Regarding the lower levels of Bloom's taxonomy, as described in Section 4.1.3, the LMS could help assess the student's abilities at the lower levels of the taxonomy to determine if they had appropriately mastered things like the basic definitions involved in a finite state automaton. Such mastery would be considered a prerequisite for further investigations with the visualizations to achieve understanding at higher levels of Bloom's taxonomy. Here, several assessment strategies are possible, as described in Section 6. Depending on the assessment results, the AV system could show simpler or more advanced examples, or it could be used for "what-if" exploration if the student has a good knowledge about the topic. In our example about the fundamentals of regular expressions and finite automata, this last issue would mean that the student could "play" with the visualization generation, taking it in directions that may well lead to their discovering new knowledge about the topic. The AV tools support their efforts to try out new ideas, to develop "unusual" automata satisfying particular criteria and to verify whether their hypotheses work [67].

The AV tool analyzes the student's behavior while interacting with the visualization or animation. This information is communicated to the LMS in order to add it to the learning model of the current student. In this way, both worlds are successfully combined from an educational perspective.

Discussion: A scenario like this clearly improves upon the current situation in which the LMS and the AV tool do not talk to each other. Such communication between the two tools would not only help to improve the learning success of AV/PV systems, but it would also influence the acceptance of AV tools. As we have stated before, the data interchanged between the LMS and an AV system will become increasingly complex and important, and thus standards will continue to emerge. They will also allow different AV tools to communicate with each other, and better adapt to the different needs of the learner. Thus, integration not only solves the problems pointed out by the educators, but also opens up new research questions such as how to make the interconnection of different AV tools possible, what kind of data should be interchanged, and how this information can be utilized to create adaptive learning environments. Initiatives for joining systems already exist (see for example the one to join Jeliot 3 and Moodle [91]). Jeliot 3 is also being integrated to CUMULATE [153], a centralized online user model that provides interfaces to store user activity from different learning applications and to send reports back to the different applications.

6.7 Support for Drill-and-Practice

Scenario: Integrate problem generator and automatic assessment with an LMS.

Motivation: Drill-and-practice exercises are common in the first stages of learning any given topic. These exercises do not require creativity, but only understanding of the topic. In terms of Bloom's taxonomy, the student must understand the concepts or methods in the lecture, and must be able to apply them to solve problems. In some fields, these exercises are the primary source of learning, especially in those involving psychomotor skills.

Drill-and-practice exercises also pose risks when not planned adequately. First, students may learn these problems by rote if very few are available. Second, plagiarism may discourage students. Third, lack of feedback can discourage students if they often fail and do not receive assistance to improve. Finally, several levels of difficulty are important to keep good students motivated as they improve.

LMSs support drill-and-practice by allowing the teachers to store exercises and automatically assess them. Once the lecture content has been delivered, students may work with the given concepts or methods by means of a number of simple exercises, probably in increasing degree of difficulty. However, lack of the features cited in the previous paragraph makes this support very rudimentary. Generation of problems, feedback, and adaptation to the student level are important features to maintain and even increase motivation and to discourage plagiarism.

For instance, consider a course on introductory programming. Once the syntax of expressions and statements in a programming language has been presented, the students are given exercises. In each exercise, they are requested to provide a missing statement or expression, after which the correct execution of the program is assessed.

Proposed Solution: Provide several tools: a problem generator, possibly a visualization tool, and an automatic assessment tool, that are incorporated into an LMS which records the results.

Discussion: A problem generator is needed to deliver different problems to the students. The generator would generate semi-random instances of the specification, adjusted so that trivial, very large or meaningless cases are not generated. The generator could even be adjusted to the student's expertise or learning style (Section 4.1.4). Depending on the problem, an auxiliary tool can be available to the student to better analyze the problem or to perform mundane tasks. For instance, for some problems, it can be useful to generate automatic visualizations in order to better comprehend them. Finally, an automatic assessment system could be in charge of assessing the answers and, if an answer is wrong, giving proper feedback to the student to raise their motivation (Section 4.1.5), and recording the results into the LMS.

Examples of tools that currently implement some of this scenario include, for example, Kumar's problets [73].

6.8 Support for Complex Construction Problems

Scenario: Include into an LMS support for developing comprehensive programs that require student creativity.

Motivation: As students become more experienced in a given topic, they could be exposed to more complex problems. Solving these problems would require applying given concepts or methods with some creativity. In terms of Bloom's taxonomy, the student would need to synthesize a product.

Problems can also be classified as open or closed. The former give flexibility to the student, whereas the latter impose restrictions, for example on the method to apply. It is interesting to note that this kind of open problem is the most common form of assignment in programming courses. In effect, given a problem specification in a natural language, illustrated with some examples, a working program that solves the problem must be coded.

Solving these complex problems requires the use of tools specific to the discipline to assist in the creation task. There are two reasons for this. First, the solution is typically required in a given format. In the case of programming, code is the most common format, but other formats are sometimes required (diagrams, testing documentation, etc.). Second, solving a non-trivial problem is simplified if auxiliary tools are provided. For instance, a compiler and debugger are the typical tools for programming. All of these needs are typically satisfied by using an IDE or an equivalent computing environment.

An example that can be solved as an open or a closed programming assignment follows. Given an (inefficient) multiple recursive algorithm with redundancies, develop an (efficient) iterative algorithm with no redundant computation. There are amenable methods to be applied, although some creativity is typically demanded from the student.

Proposed Solution: An LMS could support this kind of task by providing one or more specific tools to construct solutions. A generic IDE is the obvious solution for programming assignments, as described in Section 6.5. In other cases, such as the example given, the synthesis may be guided. In this case, additional tools may restrict or guide the student in successive phases, such as a wizard to navigate through successive phases of program transformation. In turn, this wizard could give access to visualization facilities to automatically display the behavior of the product at each stage. The choice of the concrete tool may also take the user's previous activities into account, based on the information gathered from tracking activities (Section 4.1.8.3), and may adapt the presentation according to the user's learning style (Section 4.1.4).

Discussion: LMS support can be enriched even further by widening its educational context. Thus, the assignment can be linked to an automatic assessment system. Furthermore, synthesis can be made collaboratively. In this case, a programming assignment would require collaborative support for editing, discussion, voting, and version management.

6.9 Enriching Augmented Learning

Scenario: Combine improved presence teaching with features of online learning environments, where "presence teaching" refers to traditional teaching as opposed to online or distance learning situations.

Motivation: Systems for face-to-face augmented learning (as discussed in Section 3.2) are geared to provide a rich environment inside the lecture, as well as recordings of the lecture itself. However, other interesting elements of the lecture are not supported outside the lecture, such as in a mobile setting (Section 4.2). For example, content visualizations shown during the presentation are not available in the learning environment under the learner's control, but simply as part of a video recording. This should provide a richer learning environment for students who attended the lecture and want to rehearse parts of it, as well as for those who could not attend the lecture.

Proposed Solution: By merging aspects of augmented learning and online learning environments, students and teachers can use a far richer environment for activities outside the lecture room. This environment would contain both the lecture materials, as a collection of multimedia including, for example, slides, video recordings and podcasts. This is already offered by many augmented learning systems. In addition, the expressiveness of an LMS can offer features that would include the following.

- *Interactive events happening during the lecture*. These might include submitted questions, which could be placed in a forum or other appropriate element inside the LMS (such as a wiki or blog). This way, students

can read the questions asked during the lecture, and see the lecturer's (or other students') answers to the questions.

- *Comments made by the students*. These might include personal annotations or information about typos or other errors, and should be annotated to the appropriate slide and be available to the students themselves. Again, this is best done in integration with an LMS that hosts the materials and knows about the participants (see Section 4.2 for the related issue of tracking student activities). The eMargo system [134] offers a mode for commenting individual slides with public discussion entries and private notes; an integration of this functionality with Moodle is currently ongoing.
- *Interactive demonstrations*. These might include algorithm or program visualizations (see Section 3.4 and 3.5), and could be made available to the students to run separately anytime after the lecture. Instead of only watching the contents in the video—with a lack of control and no way to adapt the presentation speed to personal needs—the student could run the same content as the teacher presented. A prototype system called AFFE [51] performs this job and publishes all presented entries on a web page, where students can directly run them using Java WebStart. However, this feature becomes more interesting if it is integrated into the learning environment that hosts the other learning materials for the course.

Discussion: The proposed integration of augmented learning with online learning environments provides a seamless transition from lectures to online learning. As opposed to distance learning, both students present at the lecture (taking notes or asking questions), and students who for one reason or another are absent can benefit from the scenario. By combining everything that happened during the lecture with activities outside the lecture, we bridge the gap between presence and distance education, and potentially provide substantial internal motivation for the students.

A concern here is that if all rich lecture content is available outside the lecture, student attendance might drop.

To address student concerns with privacy and both *funny* and *stupid* responses to questions, entries may be posted with the student's (LMS) identity hidden. A prototype of the *TVremote* system [10] offers this functionality by proving a FAQ webpage of the teacher-selected most relevant questions. However, this feature is not yet integrated into any LMS.

7. SUMMARY AND CONCLUSIONS

We have discussed the state of the art of learning resources within computer science education and their relationship with learning management systems. We have reported the results of a survey of educators that highlights some of the current deficiencies of such systems, and have also identified a number of educational technologies and tools that are under development within the computer science community. Based on those observations, and informed by basic pedagogical principles that relate to the delivery of computer science at the higher education level, we have presented guidelines that are appropriate for use by software developers integrating novel technologies into an LMS, thereby turning them into a CALMS.

We have also presented a collection of scenarios that represent enhancements to current learning management systems by incorporating novel computer science education technologies. With reference to our guidelines, we have discussed how these would improve the learning and teaching process.

There is no significant technical obstacle to the incorporation of any of the tools or technologies we have surveyed into current LMSs. The challenge is to engage the integration process in such a way that the enhanced CALMS will deliver enriched educational benefits effectively and efficiently.

We have not provided any formal pedagogical or software design patterns—these will be the subject of future work. Nor have we addressed detailed technical issues—a variety of interoperability technologies is available, including the use of plug-ins and standardized APIs, and these are yet to be developed.

If our vision of technical integration takes place, new pedagogical models will emerge. For example, the possibility of real-time automatic assessment within a CALMS supported classroom environment would offer exciting new educational opportunities.

Finally, we intend to provide a website that will host resources to support the integration of CS educational tools with current (Open Source) LMSs.

8. REFERENCES

[1] Abbing, J. and Koidl, K. Template Approach for Adaptive Learning Strategies. Workshop Proceedings of the Adaptive Hypermedia 2006 (AH 2006), Dublin, Ireland (2006).

[2] Advanced Distributed Learning. SCORM 2004, 3rd Edition. http://www.adlnet.gov/scorm/ (2007).

[3 Aiken, A. Moss: A System for Detecting Software Plagiarism. http://theory.stanford.edu/~aiken/moss/ (2006).

[4 Akingbade, A., Finley, T., Jackson, D., Patel, P. and Rodger, S. H. JAWAA: easy web-based animation from CS 0 to advanced CS courses. Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (Reno, Nevada, USA). ACM Press, New York, NY, USA, 2003, 162-166.

[5] Alexander, C., Ishikawa, S. and Silverstein, M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.

[6] Anderson, J. R. Cognitive psychology and its implications. W.H. Freeman, 1985.

[7] Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J. and Wittrock, M. C. Eds. A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives. Addison-Wesley, 2001.

[8] Anthony, D. Patterns for classroom education. Pattern Languages of Programs, PLoP'95. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996, 391 - 406.

[9] Bailey, T. and Forbes, J. Just-in-Time Teaching for CS0. Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (St. Louis, MO, USA). ACM Press, New York, NY, USA, 2005, 366-370.

[10] Bär, H., Häußge, G. and Rößling, G. An Integrated System for Interaction Support in Lectures. Proceedings of the 13th Conference on Innovation and Technology in Computer Science Education (Dundee, Scotland, UK). ACM Press, New York, NY, USA, 2007, 329.

[11] Bär, H., Rößling, G., Köbler, S. and Deneke, M. Evaluation of Digital Interaction Support in a Large Scale Lecture. Proceedings of the IADIS International Conference on Applied Computing. IADIS Press, Lisbon, Portugal, 2005, 63-67.

[12] Bär, H., Tews, E. and Rößling, G. Improving Feedback and Classroom Interaction Using Mobile Phones. Proceedings of Mobile Learning 2005. IADIS Press, Lisbon, Portugal, 2005, 55-62.

[13] Barnes, D. J. and Kölling, M. Objects First with Java. A Practical Introduction using BlueJ. Prentice Hall, 2006.

[14] Bergin, J. A Pattern Language for Initial Course Design. Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (Charlotte, North Carolina, USA). ACM, New York, NY, USA, 2001, 282-286.

[15] Bergin, J., Eckstein, J., Manns, M. L. and Sharp, H. Feedback Patterns. http://www.jeckstein.com/pedagogical-Patterns/feedback.pdf

[16] Bergin, J., Eckstein, J., Manns, M. L. and Sharp, H. Patterns for Active Learning. http://www.jeckstein.com/pedagogical-Patterns/activelearning.pdf

[17] Bergin, J., Manns, M. L., Marquardt, K., Eckstein, J. and Sharp, H. Patterns for Experiental Learning. http://www.jeck-stein.com/pedagogicalPatterns/experientiallearning.pdf

[18] Bess. Bess Peer Assessment Software. http://sourceforge.net/ projects/bess

[19] Biggs, J. and Collis, K. Evaluating the Quality of Learning: The SOLO Taxonomy. Academic Press, New York, 1982.

[20] Bloom, B. S. The Taxonomy of Educational Objectives: The Classification of the Educational Goals. Longman Group Ltd, 1956.

[21] Bruner, J. The Culture of Education. Harvard University Press, Cambridge, MA, 1996.

[22] Budd, T. An Active Learning Approach to Teaching the Data Structures Course. Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (Houston, Texas, USA). ACM Press, New York, NY, USA, 2006, 143-147.

[23] Bull, S. and Reid, E. Individualised Revision Material for Use on a Handheld Computer. In Attewell, J. and Savill-Smith, C. Eds. Learning with Mobile Devices: Research and Development 2004. Learning and Skills Development Agency, London, UK, 2004, 35-42.

[24] Carle, A., Canny, J. and Clancy, M. PACT: An Annotated Course Tool. Proceedings of ED-MEDIA 2006. AACE Press, Charlottesville, VA, USA, 2006, 2054-2060.

[25] Carle, A., Clancy, M. and Canny, J. Working with pedagogical patterns in PACT: initial applications and observations. SIGCSE Bulletin, 39, 1 (2007), 238-242.

[26] Carter, J., AlaMutka, K., Fuller, U., Dick, M., English, J., Fone, W. and Sheard, J. How shall we assess this? SIGCSE Bulletin, 35, 4 (2003), 107-123

[27] Clancy, M., Titterton, N., Ryan, C., Slotta, J. and Linn, M. New roles for students, instructors, and computers in a lab-based introductory programming course. SIGCSE Bulletin, 35, 1 (2003), 132-136.

[28] Cogliati, J. J., Goosey, F. W., Grinder, M. T., Pascoe, B. A., ROSS, R. J. and Williams, C. J. Realizing the promise of visualization in the theory of computing. Journal of Educational Resources in Computing, 5, 2 (2005), 5.

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

[30] Creative Commons Project. http://creativecommons.org (2008).

[31] Dann, W., Cooper, S. and Pausch, R. Learning to Program with Alice. Prentice Hall, 2006.

[32] Davies, P. Peer-Assessment: Judging the quality of student work by the comments not the marks? Innovations in Education and Teaching International (IETI), 43, 1 (2006), 69-82.

[33] Davis, E. A. and Linn, M. C. Scaffolding Students' Knowledge Integration: Prompts for Reflection in KIE. International Journal of Science Education 22, 8, (2000), 819–837.

[34] Dearden, A. and Finlay, J. Pattern Languages in HCI: A Critical Review. Human-Computer Interaction, 21, 1 (2006), 49-102.

[35] Demetrescu, C., Finocchi, I. and Stasko, J. T. Specifying Algorithm Visualizations: Interesting Events or State Mapping? Revised Lectures on Software Visualization, International Seminar. Springer-Verlag, London, UK, 2002, 16-30.

[36] Depradine, C. and Gay, G. Active participation of integrated development environments in the teaching of object-oriented programming. Computers & Education, 43, 3 (November 2004), 291-298.

[37] Diehl, S. Ed. Software Visualization. Springer, Heidelberg, 2002.

[38] Diehl, S., Görg, C. and Kerren, A. Animating Algorithms Live and Post Mortem. In Diehl, S. Ed. Software Visualization; LNCS State-of-the-Art Survey. Springer, 2002, 46-57.

[39] Diehl, S. and Kerren, A. Reification of Program Points for Visual Execution. Proceedings of the First IEEE International Workshop on Visualizing Software for Understanding and Analysis (VisSoft '02). IEEE Computing Society Press; IEEE, Paris, France, 2002, 100-109.

[40] Diehl, S., Kerren, A. and Weller, T. Visual Exploration of Generation Algorithms for Finite Automata. Implementation and Application of Automata; Lecture

Notes on Computer Science, LNCS 2088. Springer, 2001, 327-328.

[41] Dunbar, K. How scientists really reason: Scientific reasoning in real-world laboratories. In Sternberg R. J., Davidson J. Eds. Mechanisms of insight. MIT Press, Cambridge MA, 1995, 365-395.

[42] Dunn, R. and Dunn, K. Teaching Students through their Individual Learning Styles: A Practical Approach. Prentice-Hall, Reston, VA, 1978.

[43] Eclipse Foundation. Eclipse. http://www.eclipse.org

[44] Edwards, S. H. Improving student performance by evaluating how well students test their own programs. Journal of Educational Resources in Computing, 3, 3 (2003), 1.

[45] Felder, R. M. and Silverman, L. K. Learning and Teaching Styles in Engineering Education. Engr. Education, 78, 7 (1988), 674-681.

[46] Friedland, G., Knipping, L., Rojas, R. and Tapia, E. Teaching with an intelligent electronic chalkboard. ETP '04: Proceedings of the 2004 ACM SIGMM workshop on Effective telepresence. (New York, NY, USA). ACM Press, New York, NY, USA, 2004, 16-23.

[47] Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernan-Losada, I., Jackova, J., Lahtinen, E., Lewis, T. L., Thompson, D. M., Riedesel, C. and Thompson, E. Developing a computer science-specific learning taxonomy. SIGCSE Bulletin, 39, 4 (December 2007), 152-170.

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

[49] Gardner, H. Multiple Intelligences: The Theory in Practice. Basic Books, 1993.

[50] Griswold, W. G. and Simon, B. Ubiquitous presenter: fast, scalable active learning for the whole classroom. Proceedings of the 11th Conference on Innovation and Technology in Computer Science Education. (Bologna, Italy). ACM Press, New York, NY, USA, 2006, 358-358.

[51] Häußge, G. Flexible Verteilung und einheitliche Bedienung von interaktiven Visualisierungen. Proceedings der Pre-Converence Workshops der 5. e-Learning Fachtagung Informatik DeLFI 2007. (Siegen, Deutschland). Logos Verlag, Berlin, Germany, 2007, 85-92.

[52] Henriksen, P. and Kölling, M. Greenfoot: Combining Object Visualisation with Interaction. OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (Vancouver, BC, Canada). ACM Press, New York, NY, USA, 2004, 73-82.

[53] Hernán-Losada, I., Velázquez-Iturbide, J. Á and Lázaro-Carrascosa, C. A. Programming learning tools based on Bloom's taxonomy: proposal and accomplishments. Proceedings of the 8th International Symposium of Computers in Education (SIIE 2006). (Leon, Spain, October 24-26). 2006, 325-334.

[54] Hickey, T. J., Langton, J. and Alterman, R. Enhancing CS programming lab courses using collaborative editors. J. Comp. Sci. in Colleges, 20, 3 (February 2005), 157-167.

[55] Higgins, C., Hegazy, T., Symeonidis, P. and Tsintsifas, A. The CourseMarker CBA System: Improvements over Ceilidh. Education and Information Technologies, 8, 3 (2003), 287-304.

[56] Hillside Group. http://hillside.net

[57] Hine, N., Rentoul, R. and Specht, M. Collaboration and Roles in Remote Field Trips. In Attewell, J. and Savill-Smith, C. Eds.Learning with Mobile Devices: Research and Development 2004. Learning and Skills Development Agency, London, UK, 2004, 69-72.

[58] Holmes, N. The Craft of Programming. IEEE Computer, 41, 5 (2008), 90-92.

[59] Hoyles, C., Healy, L. and Schutterland, R. Patterns of discussion between pupil pairs in computer and non-computer environments. Journal of Computer-Assisted Learning, 7(1991), 210-226.

[60] Hoyles, C. and Shutterland, R. Logo mathematics in the classroom. Routledge, London, 1989.

[61] Hristova, M., Misra, A., Rutter, M. and Mercouri, R. Identifying and correcting Java programming errors for introductory computer science students. Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education. (Reno, NV, USA). ACM Press, New York, NY, USA, 2003, 153-156.

[62] IMC Advanced Learning Solutions. The fast track to e-learning content: LECTURNITY, the award-winning Rapid Authoring Tool. http://www.lecturnity.de/en/products/lecturnity.

[63] Jadud, M. C. Methods and tools for exploring novice compilation behavior. ICER 2006: Proceedings of the 2006 International Workshop on Computing Education Eesearch. (Canterbury, Kent, UK). ACM Press, New York, NY, USA, 2006, 73-84.

[64] Joy, M. and Luck, M. Plagiarism in Programming Assignments. IEEE Transactions on Education, 42, 1 (1999), 129-133.

[65] Joy, M., Griffiths, N. and Boyatt, R. The boss online submission and assessment system. Journal of Educational Resources in Computing, 5, 3 (2005), 2.

[66] Karavirta, V., Korhonen, A., Malmi, L. and Stalnacke, K. MatrixPro - A Tool for On-The-Fly Demonstration of Data Structures and Algorithms. In Korhonen, A. Ed. Proceedings of the Third Program Visualization Workshop. (Coventry, UK). The University of Warwick, UK, 2004, 26-33.

[67] Kerren, A. Generation as Method for Explorative Learning in Computer Science Education. Proceedings of the 9th Conference on Innovation and Technology in Computer Science Education (ITiCSE '04). (Leeds, UK). ACM Press, New York, NY, USA, 2004, 77-81.

[68] Kerren, A., Müldner, T. and Shakshuki, E. Novel Algorithm Explanation Techniques for Improving Algorithm Teaching. Proceedings of the 3rd ACM Symposium on Software Visualization (SoftVis '06).

(Brighton, UK). ACM Press, New York, NY, USA, 2006, 175-176.

[69] Kitcher, P. The advancement of science. Oxford University Press, New York, 1993.

[70] Kolb, D. A. Experiential Learning: Experience as the Source of Learning and Development. Prentice-Hall Inc, New Jersey, USA, 1984.

[71] Korhonen, A. Visual Algorithm Simulation. Ph.D. Thesis, Dept. of Computer Science, Helsinki University of Technology, 2003.

[72] Ktoridou, D. and Eteokleous, N. Adaptive M-learning: Technological and Pedagogical Aspects to be Considered in Cyprus Tertiary Education. Proceedings of the 3rd International Conference on Multimedia and Information and Communication Technologies in Education (m-ICTE2005). 2005.

[73] Kumar, A. Generation of problems, answers, grade, and feedback – Case study of a fully automated tutor. Journal of Educational Resources in Computing, 5, 3 (September 2005), article 3.

[74] Laurel, B. Computers as Theatre. Addison-Wesley, 1993.

[75] Learning Environments for Progressive Inquiry Research Group. Fle3 > Future Learning Environment. http://fle3.uiah.fi (2008).

[76] Lehtinen, E. and Rui, E. Computer supported complex learning: An environment for learning experimental method and statistical inference. Machine Mediated Learning, 5, 3&4 (1995), 149-175.

[77] Lehtinen, E., Hakkarainen, K., Lipponen, L., Rahikainen, M. and Muukkonen, H. Computer Supported Collaborative Learning: A Review. The J.H.G.I. Giesbers Reports on Education, Nr. 10,. Department of Educational Sciences, University of Nijmegen, 1999.

[78] Lesgold, A., Weiner, A. and Suthers, D. Tools for thinking about complex issues. Proceedings of the 6th European Conference for Research on Learning and Instruction, 1996.

[79] Linn, M. C., Davis, E. A. and Bell, P. Internet Environments for Science Education. Lawrence Erlbaum Associates, Inc, Mahwah, NJ, USA, 2004.

[80] Liu, T., Kiang, J., Wang, H. and Wei,TakWai Chan and LiHsing. Embedding EduClick in Classroom to Enhance Interaction. Proceedings of the International Conference on Computers in Education (ICCE. (Hong Kong, China). IEEE Press, 2003, 117-125.

[81] Lockitt, B. Mobile Learning. http://nt6140.vs.netbenefit.co.uk/pdf/handheldcomputing3t.pdf (2005).

[82] Lukosch, S. and Schümmer, T. Groupware development support with technology patterns. International Journal of Man-Machine Studies, 64, 7 (2006), 599-610.

[83] MacManus, T. Mobile What? The Educational Potential of Mobile Technologies. Proceedings of the World Conference on E-Learning in Corporations, Government, Health, and Higher Education. 2002, 1895–1898.

[84] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O. and Silvasti, P. Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. Informatics in Education, 3, 2 (2004), 267-288.

[85] Manolescu, D., Voelter, M. and Noble, J. Pattern Languages of Program Design 5 (Software Patterns Series). Addison-Wesley Longman Publishing Co., Inc., 2006.

[86] McConnell, J. Active and Cooperative Learning: Tips and Tricks (Part I). SIGCSE Bulletin, 37, 2 (2005), 27-30.

[87] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In SIGCSE Bulletin. 33, 4, (2001), 125-180.

[88] Meisalo, V. and Lavonen, J. Bits and processes on markets and webs. An analysis of virtuality, reality and metaphors in a modern learning environment. Journal Teacher Researcher, 2 (2000), 10-27.

[89] Meisalo, V., Sutinen, E. and Tarhio, J. Modernit oppimisympäristöt. Tietosanoma, Finland, 2003.

[90] Mitrovic, A. Learning SQL with a computerized tutor. SIGCSE Bulletin, 30, 1 (1998), 307-311.

[91] Moreno, A. Program Animation Activities in Moodle. Proceedings of the 13th Conference on Innovation and Technology in Computer Science Education, (Madrid, Spain). ACM Press, New York, NY, USA, 2008, 361-361.

[92] Moreno, A., Myller, N., Sutinen, E. and Ben-Ari, M. Visualizing programs with Jeliot 3. Proceedings of Advanced Visual Interfaces, AVI 2004. 2004, 373-376.

[93] Morth, T., Oechsle, R., Schloss, H. and Schwinn, M. Automatische Bewertung studentischer Software. Proceedings der Pre-Conference Workshops der 5. e-Learning Fachtagung Informatik (DeLFI 2007). (Siegen, Germany). Logos Verlag Berlin, 2007.

[94] Muir, D. Adapting Online Education to Different Learning Styles. In Anonymous Proceedings of the National Educational Computing Conference, "Building on the Future". Chigago, IL, 2001, 1-15.

[95] Munoz, M. and Kloos, C. A Web Service Based Architecture for Push-Enabled M-Learning. Proceedings of IADIS Mobile Learning Conference. (Malta). IADIS Press, Lisbon, Portugal, 2005, 135-140.

[96] Myller, N., Laakso, M. and Korhonen, A. Analyzing engagement taxonomy in collaborative algorithm visualization. Proceedings of the 12th Conference on Innovation and Technology in Computer Science Education. (Dundee, Scotland, UK). ACM Press, New York, NY, USA, 2007, 251-255.

[97] Naps, T. L. JHAVÉ -- Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement. IEEE Computer Graphics and Applications, 25, 5 (2005), 49-55.

[98] Naps, T. L. and Rößling, G. JHAVÉ - More Visualizers (and Visualizations) Needed. In Rößling, G. Ed. Proceedings of the Fourth Program Visualization Workshop. Electronic Notes in Theoretical Computer Science, 178, 4 (2007), 33-41.

[99] Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L.,

McNally, M., Rodger, S. and Velázquez-Iturbide, J. Á. Exploring the Role of Visualization and Engagement in Computer Science Education. SIGCSE Bulletin, 35, 2 (2003), 131-152.

[100] Northumbria Learning. JISC Plagiarism Advisory Service. http://jiscpas.ac.uk/

[101] Novak, J. D. Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Corporations. Lawrence Erlbaum Associates, 1998.

[102] O'Malley, C., Vavoula, G., Glew, J., Taylor, J., Sharples, M. and Lefrere, P. Guidelines for Learning/Teaching/Tutoring in a Mobile Environment. Open University, 2003.

[103] Pareja-Flores, C., Urquiza-Fuentes, J. and Velázquez-Iturbide, J. Á. WinHIPE: An IDE for functional programming based on rewriting and visualization. ACM SIGPLAN Notices, 42, 3 (2007), 14-23.

[104] Parlante, N. JavaBat java practice problems. http://javabat.com.

[105] Parsons, D. and Ryu, H. A Framework for Assessing the Quality of Mobile Learning. Learning and Teaching Issues in Software Quality, Proceedings of the 11th International Conference for Process Improvement, Research and Education (INSPIRE). (Southampton Solent University, UK). 2006, 17-27.

[106] Parsons, D., Ryu, H. and Cranshaw, M. A Study of Design Requirements for Mobile Learning Environments. Proceedings of the IEEE International Conference on Advanced Learning Technologies 2006 (ICALT). 2006, 96-100.

[107] Pavlovic, J., Pitner, T. and Kubasek, M. Digital Library for PDA Facilities. Proceedings of IADIS Mobile Learning Conference. (Malta). IADIS Press, Lisbon, Portugal, 2005, 169-275.

[108] Peterson, P. and Swing, S. Student Cognitions as Mediators of the Effectiveness of Small Group Learning. Journal of Educational Psychology., 36(1985), 351-372.

[109] Phillips, D. C. Ed. Constructivism in Education (Ninety-Ninth NSSE Yearbook). University of Chicago Press, Chicago, IL, 2000.

[110] PPP: The Pedagogical Patterns Project. Pedagogical Patterns.http://www.pedagogicalpatterns.org.

[111] Prechelt, L., Malpohl, G. and Philippsen, M. Finding Plagiarisms among a Set of Programs with JPlag. Journal of Universal Computer Science, 8, 11 (November 2002), 1016-1038.

[112] Qiu, L. and Riesbeck, C. K. An incremental model for debeloping computer-based learning environments for problem-based learning. ICALT 2004: Proceedings of the IEEE International Conference on Advanced Learning Technologies. (Washington, DC). IEEE Computer Society, 2004, 171-175.

[113] Radenski, A. Digital Support for Abductive Learning in Introductory Computing Courses. Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, KY, USA). ACM, New York, NY, USA, 2007, 14-18.

[114] Radenski, A. Python First: A Lab-Based Digital Introduction to Computer Science. Proceedings of the 11th Conference on Innovation and Technology in Computer Science Education (Bologna, Italy). ACM, New York, NY, USA, 2006, 197-201.

[115] Rajala, T., Laakso, M., Kaila, E. and Salakoski, T. VILLE – A Language-Independent Program Visualization tool. Proceedsing of Seventh Baltic Sea Conference on Computing Education Research (Koli Calling). Conferences in Research and Practice in Information Technology 88, (2008).

[116] Razmov, V. and Anderson, R. Pedagogical Techniques Supported by the Use of Student Devices in Teaching Software Engineering. Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (Houston, Texas, USA,). ACM, New York, NY, USA, 2006, 344-348.

[117] Rising, L. Understanding the Power of Abstraction in Patterns. IEEE Software, July-August (2007), 2-7.

[118] Rodger, S. and Finley, T. JFLAP - An Interactive Formal Languages and Automata Package. Jones and Bartlett, 2006.

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

[120] Ross, R. Hypertextbooks and a Hypertextbook Authoring System. Proceedings of the 13th Conference on Innovation and Technology in Computer Science Education. (Madrid, Spain). ACM Press, New York, NY, USA, 2008, 133-137.

[121] Ross, R. Theory of Computing. http://www.cs.montana.edu/ webworks/projects/theoryportal/.

[122] Rößling, G. and Ackermann, T. A Framework for Generating AV Content on-the-fly. In Rößling, G. Ed. Proceedings of the Fourth Program Visualization Workshop, Electronic Notes in Theoretical Computer Science 178, 4 (2007), 23-31 .

[123] Rößling, G. and Hartte, S. WebTasks: Online Programming Exercises Made Easy. Proceedings of the 13th Conference on Innovation and Technology in Computer Science Education Conference. (Madrid, Spain). ACM Press, New York, NY, USA, 2008, 363.

[124] Rößling, G., Mehlhase, S. and Pfau, J. A Java API for Creating (not only) AnimalScript. Proceedings of the Program Visualization Workshop 2008 (PVW 2008), (2008), 105-112.

[125] Rößling, G. and Vellaramkalayil, T. First Steps Towards a Visualization-Based Computer Science Hypertextbook as a Moodle Plugin. Proceedings of the Program Visualization Workshop 2008 (PVW 2008), (2008), 29-36.

[126] Rößling, G. Translator: A Package for Internationalization for Java-based Applications and GUIs. Proceedings of the 12th Conference on Innovation and Technology in Computer Science Education (ITiCSE 2006). (Bologna, Italy). ACM Press, New York, NY, USA, 2006, 312.

[127] Rößling, G. and Freisleben, B. ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation. Journal of Visual Languages and Computing, 13, 3 (2002), 341-354.

[128] Rößling, G. and Freisleben, B. AnimalScript: An Extensible Scripting Language for Algorithm Animation. Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001). (Charlotte, North Carolina, USA). ACM Press, New York, NY, USA, 2001, 70-74.

[129] Rößling, G. and Naps, T. L. A Testbed for Pedagogical Requirements in Algorithm Visualizations. Proceedings of the 7th Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002). (Århus, Denmark).ACM Press, New York, NY, USA, 2002, 96-100.

[130] Rößling, G., Naps, T., Hall, M. S., Karavirta, V., Kerren, A., Leska, C., Moreno, A., Oechsle, R., Rodger, S. H., Urquiza-Fuentes, J. and Velázquez- Iturbide, J. Á. Merging Interactive Visualizations with Hypertextbooks and Course Management. SIGCSE Bulletin, 38, 4 (2006), 166-181.

[131] Rößling, G., Trompler, C., Mühlhäuser, M., Köbler, S. and Wolf, S. Enhancing Classroom Lectures with Digital Sliding Blackboards. Proceedings of the 9h Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004). (Leeds, UK). ACM Press, New York, NY, USA, 2004, 218-222.

[132] Scardamalia, M. and Bereiter, C. Technologies for knowledge-building discourse. Communications of the ACM, 36, 5 (1993), 37-41.

[133] Scheele, N., Seitz, C., Effelsberg, W. and Wessels, A. Mobile Devices in Interactive Lectures. Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunication (ED-MEDIA). (Lugano, Switzerland). AACE Press, Charlottesville, VA, USA, 2004, 154-161.

[134] Sesink, W., Göller, S., Rößling, G. and Hofmann, D. eMargo: Eine Digitale Randspalte zum Selbststudium (nicht nur) der Informatik. Proceedings der Pre-Conference Workshops der 5. e-Learning Fachtagung Informatik (DeLFI 2007). (Siegen, Germany). Logos Verlag Berlin, Germany, 101-108.

[135] Shakshuki, E., Müldner, T. and Kerren, A. Algorithm Education Using Structured Hypermedia. Advances in Distance Education Technologies Series 2, 5 (2008)), 58-84.

[136] Skinner, B. F. The Technology of Teaching. Appleton-Century-Crofts, New York, 1968.

[137] Soloway, E., Guzdial, M. and Hay, K. E. Learner-centered design: the challenge for HCI in the 21st century. Interactions, 1, 2 (April 1994), 36-48.

[138] Srinivas, H. 44 Benefits of Collaborative Learning. http://www.gdrc.org/kmgmt/c-learn/44.html

[139] Stasko, J. T., Domingue, J., Brown, M. H. and Price, B. A. Software Visualization. MIT Press, USA, 1998.

[140] Suzuki H., Hiroshi K. Identity formation/transformation as the process of collaborative learning through AlgoArena. Proceedings of The Second International Conference on Computer Support for Collaborative Learning. (Toronto, Ontario, Canada). 1997, 280-288.

[141] Sykes, E. Developmental process model for the Java intelligent tutoring system. Journal of Interactive Learning Research, 18, 3 (2007), 399-410.

[142] Trætteberg, H. and Aalberg, T. JExercise: a specification-based and test-driven exercise support plugin for Eclipse. Proceedings of the 2006 OOPSLA workshop on Eclipse technology eXchange. (Portland, Oregon, USA). ACM Press, New York, NY, USA, 2006, 70-74.

[143] Trnkova, J., Rößling, G., Sugonyak, O. and Mühlhäuser, M. WiBA-Net: A Web-Based Learning Platform for Civil Engineers and Architects. Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA). (Lugano, Switzerland). AACE Press, Charlottesville, VA, USA, 2004, 144-149.

[144] Velázquez-Iturbide, J. A., Perez-Carrasco, A. and Urquiza-Fuentes, J. SRec: An animation system of recursion for algorithm courses. Proceedings of the 13th Conference on Innovation and Technology in Computer Science Education, ITiCSE 2008. (Madrid, Spain). ACM Press, New York, NY, USA, 2008, 225-229.

[145] Velázquez-Iturbide, J. A., Redondo-Martin, D., Pareja-Flores, C. and Urquiza-Fuentes, J. An instructor's guide to design web-based algorithm animations. LNCS, 4823(2008), 440-451.

[146] Vogel, R. and Wippermann, S. Didaktische Design Pattern zur Dokumentation von Lehr-Lern-Formen an Hochschulen. http://www.didaktische-design-patterns.de/index.html (2005).

[147] Vygotsky, L. S. Mind in Society: The Development of Higher Psychological Processes. Harvard University Press., Cambridge MA, 1978.

[148] Wagner, G. VisionQuest Users Guide. Collaborative Technologies Co., Austin, TX, 1991.

[149] WebPA. Webpage. http://webpaproject.lboro.ac.uk.

[150] White, D. R. and Joy, M. S. Sentence-based natural language plagiarism detection. Journal of Educational Resources in Computing, 4, 4 (2004), 2.

[151] Wilkerson, M., Griswold, W. G. and Simon, B. Ubiquitous presenter: increasing student access and control in a digital lecturing environment. SIGCSE Bulletin, 37, 1 (2005), 116-120.

[152] Woolley, J. D. Young children's understanding of fictional versus epistemic mental representations: Imagination and belief. Child Development, , 66 (1995), 1011-1021.

[153] Yudelson, M., Brusilovsky, P. and Zadorozhny, V. A user modeling server for contemporary adaptive hypermedia: An evaluation of the push approach to evidence propagation. In Conati, C., McCoy, K. F. and Paliouras, G. Eds.User Modeling, Volume 4511 of Lecture Notes in Computer Science. Springer, Heidelberg, 2007, 27-36.