UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Kolin Kolistelut – Koli Calling 2002

PROCEEDINGS OF THE SECOND ANNUAL FINNISH / BALTIC SEA CONFERENCE ON COMPUTER SCIENCE EDUCATION

October 18–20, 2002
Koli, Finland

Report A-2002-7

ACM K.3
UDK 681.3
ISSN 0789-7316
ISBN 952-458-197-3

Foreword

Welcome to the second – what we hope to continue as an annual – Finnish/Baltic Sea Conference on Computer Science Education. The venue is again Koli, the Finnish national landscape, like even the more informal title of the meeting tells: Kolin kolistelut – Koli Calling.

The nickname of the conference is not just a sudden brainwave but it reflects our expectations for the outcomes of the meeting. The Finnish word 'kolistelu' means clanking or rattling, but it takes place in the beautiful surroundings of Koli. We are looking forward to an inspiring atmosphere which allows us to share fresh but still fragile thoughts of how the education of our discipline could improve.

So Koli Calling hopes to irritate each of us to novel insights, but in an encouraging way: to look further away, to see clearer, distant from our everyday duties and hurries.

Compared to the first Kolistelut last year, our programme has expanded. Besides regular presentations, we have demonstrations. We also managed to get a minor textbook exhibition. Anders Berglund from Uppsala promised to take us into the world of educational research methodologies, in his invited seminar. In particular, it is a pleasure for us to have Moti Ben-Ari from the Weizmann Institute of Science, Israel, to give the keynote "From Theory to Experiment to Practice in CS Education".

I wish all of us enjoyable, relaxing, and fruitful days in the Koli area. Let us join our efforts in bringing Computer Science and its education in both practice and theory closer to each other.

Joensuu, October 18, 2002

Erkki Sutinen

Program committee

Erkki Sutinen (Chairman)	University of Joensuu, Finland
Mordechai Ben-Ari	Weizmann Institute of Science, Israel
Anders Berglund	University of Uppsala, Sweden
Michael E. Caspersen	University of Aarhus, Denmark
Valentina Dagiene	Vilnius University, Lithuania
Jaakko Kurhila	University of Helsinki, Finland
Lauri Malmi	Helsinki University of Technology, Finland
Martti Penttonen	University of Kuopio, Finland
Tapio Salakoski	University of Turku, Finland
Marja Kuittinen (Publication editor)	University of Joensuu, Finland

Organizing committee

Jarkko Suhonen (Head)	University of Joensuu, Finland
Jaana Lukkarinen (Local arrangements)	University of Joensuu, Finland
Niko Myller (Webmaster and technical support)	University of Joensuu, Finland

Contents

KEYNOTE SPEECH

From Theory to Experiment to Practice in CS Education

Mordechai Ben-Ari
Department of Science Teaching Weizmann Institute of Science Rehovot 76100 Israel
and
Department of Computer Science University of Joensuu Joensuu FIN-80101 Finland

moti.ben-ari@weizmann.ac.il, moti.ben-ari@cs.joensuu.fi

1 Introduction

In this paper, I will summarize a five-year excursion into computer science education (CSE). The excursion began with a theoretical insight, continued with experimental studies and can now be summarized in suggestions for improving the pedagogy of computer science (CS). The theoretical insight pertained to the applicability of constructivism in CSE: I concluded that students find it difficult to construct viable mental models of computers and software artifacts, because they have no pre-existing models upon which to build. I conjectured that the creation of conceptual models by educators can make it easier to acquire viable mental models, and my graduate students have performed empirical experiments to verify this conjecture. These experiments identified the important features of conceptual models that can guide educators, both teachers and developers of learning material and software.

2 Theory: constructivism, minimalism and bricolage

The following summary is based upon Ben-Ari (2001). An educational theory has four components: an *ontology* which describes what exists; an *epistemology* which characterizes knowledge; a *methodology* which explains how knowledge can be obtained; and a *pedagogy* which presents teaching techniques based upon the methodology. According to a classical view of education:

- There is an ontological reality.
- Epistemology is foundational: truth can be discovered through experiment and logic.
- Methodology: the mind is a clean slate that can be filled with knowledge.
- Pedagogy: lectures and books are the primary means of knowledge transmission.

According to a *constructivist* view of education (von Glasersfeld, 1995):

- Ontological reality is at best irrelevant because we can never truly know anything.
- Epistemology is nonfoundational: absolute truth is unattainable and fallible.
- Methodology: knowledge is acquired by creating new cognitive structures out of existing ones. Each student will construct new knowledge differently.
- Pedagogy: The task of the teacher is to guide the student in constructing her own knowledge. Feedback from teachers and students ensures that the knowledge is viable.

Proponents of constructivism often hold postmodernist positions similar to the idealist position in philosophy. I shall ignore these philosophical aspects here and concentrate on the pedagogical implications of constructivism, which are usually expressed by saying that constructivist teaching is learner-centered and based on active participation, though opponents of constructivism claim that these pedagogical principles have been known since Socrates and do not need to be justified by postmodernist philosophy.

Closely allied with constructivism is *minimalism* (Carroll, 1990), which focuses on documentation for the novice who is learning how to use software packages. The principles of minimalism are:

- Start with meaningful realistic tasks.
- Reduce reading and other passive activity, and eliminate or defer conceptual material.
- Make error recovery pedagogically productive.

The success of minimalism has been empirically demonstrated in training tasks like learning to use a word processor. But I questioned whether it can be successful for learning to perform tasks that involve complex concepts.

A final theoretical influence is the concept of *bricolage*, originally enunciated by anthropologist Claude Lévi-Strauss, and used by Turkle and Papert (1990, 1991) to describe an exploratory approach to programming and other CS activities:

> Bricoleurs construct theories by arranging and rearranging, by negotiating and renegotiating with a set of well-known materials. ... The bricoleur resembles the painter who stands back between brush strokes, looks at the canvas, and only after this contemplation, decides what to do next.

I distinguish between bricolage and trial-by-error, in that trial-by-error contains elements of analysis, conjecture and prediction, rather than just aimless trials. I believe that bricolage is not an effective methodology for professional programming, and that the normative planning style must eventually be learned, because systems involving concurrency, real-time or communications are simply not amenable to bricolage.

3 Mental models and conceptual models

A central claim in Ben-Ari (2001) was that constructivism has to be adapted to CSE because a (beginning) computer science student has no effective model of a computer; therefore, one must be provided by the teacher. I would now express this in the terminology used by Norman (1983): the computer or software or language is the *target system*; the student must develop a viable *mental model*; an essential step in this development is the creation of an explicit *conceptual model*:

> A conceptual model is invented to provide an appropriate representation of the target system, appropriate in the sense of being accurate, consistent, and complete. Conceptual models are invented by teachers, designers, scientists, and engineers. (Norman, 1983, p. 7)

The distinction between mental and conceptual models can be clarified by analyzing the concept of *WYSIWYG (What You See Is What You Get)*. WYSIWYG word processors are supposed to be user-friendly, because working with them is supposed to be analogous to writing with a pencil on a sheet of paper, but this metaphor cannot furnish an explanation for the phenomena the user encounters, so he becomes frustrated, anxious and loses self-confidence. In fact, WYSIWYG really is much deeper than the metaphor:

- What you *get* is a data structure for storing text and a set of operations on that data structure.
- What you *see* is a visual rendering of the data structure, and icons to invoke the operations.
- What you have to *do* is to construct a viable mental model of the data structure and the effect of each operation, and to give each icon its meaning as an operation.

I claim that in order to learn how to use a WYSIWYG system, the teacher or the learning materials must construct a conceptual model that will make explicit the existence of the underlying data structure and the mapping between the visual rendering and the data structure.

4 Experiment: conceptual models

I performed an experiment designed to show that users of software packages engage in bricolage (Ben-Ari, 1999). The subjects were asked to perform modifications on an MS-Word document. The subjects were highly-experienced science teachers doing graduate work in science teaching who use MS-Word regularly for writing learning materials, research papers and so on. Thus they are not to be suspected as lacking motivation to learn or an appreciation of quality pedagogy. The tasks were chosen to be relatively easy if the subject knew the relevant concepts (soft carriage returns, the distinction between a table cell and its contents, bidirectional language support, etc.), but quite difficult to perform by aimless trail and error.

Surprisingly, the subjects, in spite of their age, sophistication and experience, performed their tasks like beginning students! They did not analyze the tasks conceptually; instead, they invariably used aimless trial-and-error that I have called bricolage. Furthermore, they did not attempt to use the Help facility. I encountered frequent anthropomorphisms, like "He did it to me again," and "That's not nice of him." They were defensive and displayed a high level of anxiety. Contrary to popular opinion that learning takes place when there are real problems to be solved, when my subjects were desperate they did not use it as an opportunity to learn, but rather used dumbed-down techniques to work around the problem: "I'd work like a donkey."

I conjectured that teaching an explicit conceptual model would improve learning of MS-Word, and this conjecture was experimentally investigated by my M.Sc. student Tzippora Yeshno (Yeshno and Ben-Ari, 2001). Bilingual word processing was taught to three classes of 9th-grade students: the control group received a task-oriented tutorial, while two treatment groups received a concept-oriented tutorial. The students were instructed in common bilingual tasks such as inserting numbers or English words within Hebrew text. Superficially, the behavior of the word processor is quite complex in that the cursor direction and current language change automatically as certain keys are pressed. The control group was taught *how* to perform such tasks, while the treatment groups were taught a *conceptual model* of the target system, namely, that runs of characters in the same family (English letters, Hebrew letters, numbers and special symbols) are arranged in *blocks*, and that the superficial behavior reflects manipulations of these blocks.

Exercises and examinations were given that contained two types of questions: some which could be done using task-oriented instructions, while others required conceptual understanding. Students in the treatment groups used the conceptual model to *verbalize* their answers (both orally and in writing), leading to improved task performance. Students in the control group dealt with the assignments in different manner recalling bricolage, and could not verbalize their attempts to solve the assignments.

5 Experiment: program visualization

The most serious problem in CSE is the difficulty of learning elementary programming; this difficulty causes many beginning students to abandon the study of CS. One suggestion has been to use visualization, which we can interpret as an attempt to use conceptual models that are concrete representations. Hopefully, these models will facilitate the construction of a mental model. However, visualization in and of itself is not a panacea, and certain preconditions must exist for it to be successful. Petre (1995) showed that there are significant differences in the visualization needs of experts and novices. Experts are successful at interpreting *secondary notation* (like layout and typography) and can benefit from complex visualization. On the other hand, novices need a constrained system in which secondary notation is minimized in order to reduce the richness and the potential for misunderstanding. In algorithm and program animation, relatively little empirical research has been done and much of that has produced mixed results; see Hundhausen et al. (2002) for a survey. For example, in a series of experiments (Byrne et al., 1996; Kehoe et al., 1999; Stasko et al., 1993), Stasko showed that animations are not pedagogically useful in isolation, but require coordination with learning materials or human interaction. He also showed that the interaction with the learner should not be passive, and that exercises in prediction are important.

My M.Sc. student Ronit Ben-Bassat Levy carried out an empirical experiment with the Jeliot 2000 program animation system (Ben-Bassat Levy et al., 2002), a re-implementation of an earlier system now known as Jeliot I. (A survey of both systems appears in Ben-Ari et al. (2002).) Jeliot 2000 was designed to produce animations that are relevant to total novices, sacrificing the scope and flexibility of the Jeliot I animations. The experiment was performed on tenth-grade high school students studying an introductory course on algorithms and programming. Unlike Stasko's experiments which were short-term laboratory experiments, in our research, two full-year classes were taught, one using Jeliot 2000 and one as a control group. We believe that it is important to evaluate animations and other software tools within the framework of a complete course, rather than in a single episode, so that the students can pass the learning curve for the tool.

Here I give one example of the results; for more detail see Ben-Bassat Levy et al. (2002). The students had learned simple `if`-statements and then were given an exercise that contained nested `if`-statements, which is considered to be one of the most difficult concepts in elementary programming. In the control group, the stronger students could answer the questions only after many attempts; furthermore, they had difficulty explaining their solutions. Other students could not even answer the questions. Paradoxically, in the animation group, stronger students had difficulties answering this question, because they believed they could understand the material without referring to Jeliot 2000. Weaker students refused to work on the problem, claiming that nested `if`-statements are not legal or that they did not understand the program. The mediocre students were the ones that gave correct answers! They drew a Jeliot display and used it to hand simulate the execution of the program.

Animation does not improve performance of all students: stronger students do not need it, while weaker students are overwhelmed by the tool. However, for many students, the concrete conceptual model offered by the animation can make the difference between success and failure. The explanation is that the animation group learned to use a different and better *vocabulary of terms* than did the control group. In our theoretical terms, animations can provide a conceptual model that will assist students in forming a viable mental model.

6 Practice

The pedagogical lessons learned from this sequence of research projects can be summarized as follows:

- Explicitly teach a model of a computer: hardware, operating system, communications.
- Dig underneath your own expert knowledge to expose the prior knowledge needed to construct a viable model of the material that you are teaching.
- Construct and refine conceptual models for each topic and artifact.
- Design software tools such as visualizations to present conceptual models.
- Take into account the time needed to learn to use tools such as visualization and the time needed to explain the conceptual models.

References

Ben-Ari, M., 1999. Bricolage forever! In: 11th Workshop of the Psychology of Programming Interest Group. Leeds, UK, pp. 53–57, http://www.ppig.org/papers/11th-benari.pdf.

Ben-Ari, M., 2001. Constructivism in computer science education. Journal of Computers in Mathematics and Science Teaching 20 (1), 45–73.

Ben-Ari, M., Myller, N., Sutinen, E., Tarhio, J., 2002. Perspectives on program animation with Jeliot. Lecture Notes in Computer Science 2269, 31–45.

Ben-Bassat Levy, R., Ben-Ari, M., Uronen, P. A., 2002. The Jeliot 2000 program animation system. Computers & Education 40 (1), 1–15.

Byrne, M., Catrambone, R., Stasko, J., 1996. Do algorithm animations aid learning? Tech. Rep. GIT-GVU-96-19, Georgia Institute of Technology.

Carroll, J. M., 1990. The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill. MIT Press, Cambridge, MA.

Hundhausen, C. D., Douglas, S. A., Stasko, J. T., 2002. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing 13 (3), 259–290.

Kehoe, C., Stasko, J., Taylor, A., 1999. Rethinking the evaluation of algorithm animations as learning aids: An observational study. Tech. Rep. GIT-GVU-99-10, Georgia Institute of Technology.

Norman, D. A., 1983. Some observations on mental models. In: Gentner, D., Stevens, A. L. (Eds.), Mental Models. Lawrence Erlbaum Associates, pp. 7–14.

Petre, M., 1995. Why looking isn't always seeing: Readership skills and graphical programming. Communications of the ACM 38 (6), 33–44.

Stasko, J., Badre, A., Lewis, C., 1993. Do algorithm animations assist learning: An empirical study and analysis. In: Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems. Amsterdam, The Netherlands, pp. 61–66.

Turkle, S., Papert, S., 1990. Epistemological pluralism: Styles and cultures within the computer culture. Signs: Journal of Women in Culture and Society 16 (1), 128–148.

Turkle, S., Papert, S., 1991. Epistemological pluralism and the revaluation of the concrete. In: Harel, I., Papert, S. (Eds.), Constructionism. Ablex, Norwood, NJ, pp. 161–191.

von Glasersfeld, E., 1995. A constructivist approach to teaching. In: Steffe, L. P., Gale, J. (Eds.), Constructivism in Education. Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 3–15.

Yeshno, T., Ben-Ari, M., 2001. Salvation for bricoleurs. In: 13th Workshop of the Psychology of Programming Interest Group. Bournemouth, UK, pp. 225–235, http://www.ppig.org/papers/13th-yeshno.pdf.

PAPER SESSION 1

Producing Web Course Material with IBM Knowledge Factory Team

H. Laine and T. Kerola
Department of Computer Science, P.O.Box 26, FIN-00014 University of Helsinki

{Harri.Laine,Teemu.Kerola}@cs.Helsinki.fi

1 Introduction

The Department of CS at University of Helsinki has now for 1.5 years participated in IBM sponsored TUeLIP (Top University e-Learning International Program) project. TUeLIP is a collaborative project of six European Universities (their CS departments) and IBM. Its goal is to produce Web based e-learning material and courses to be used by the participants.

As most participants had scant knowledge of e-learning in general, the project has been a very good kick-off to introduce e-learning aspects in general to the CS departments involved. The expectations on the kind of e-learning to be used varied from self-study courses to co-operative learning, and from interactive Web courses to online instruction sessions. However, the project is not restricted to any specific course type but encourages the participants to try out new techniques and styles in their courses.

The participants were given a free ride in the form of IBM sponsoring the development of one prototype e-learning course. Introduction to Databases was selected as the prototype and its development was distributed among four universities. This gave us and all the responsible universities a good view on IBM's way in producing e-learning courses.

In general, the approach is very professional and thorough, although, it may not be suitable for our courses at large. It is like a well-defined software engineering process as compared to common arts like way of course production. However, understanding this process well gives us valuable insight on how we should proceed with our own material production.

IBM Knowledge Factory (KF) is the content production part of IBM's greater Mindspan Solutions (IBM, 2002a) scheme, that is geared at companies easing themselves into e-learning. The KF team produces, with clients' help, the final product. Knowledge Producer (KP) is an IBM internal software tool that is used by KF team to actually put together the product.

Using the course material is a separate issue to producing it, and in general the e-learning courses could be run on any platform (e.g., WebCT). However, KP is especially geared to produce material well suited for Lotus Learning Space (LLS) (IBM, 2002b). The prototype course was implemented on LLS environment.

We now introduce the Knowledge Factory process with more details. We also briefly describe our experiences with the development process, the material produced and the use of this material. Finally, we summarize our experiences and give some ideas of the future.

2 Knowledge Factory concept

A Knowledge Factory team consists of many people in various roles (IBM, 2000). Some roles can be taken by the client (company or University instructor wanting the final product), but most of the roles are filled with Knowledge Factory people. The roles are divided into two groups: External Team members are specialist and or consultants of various expertise areas, whereas the Core Team members do the actual product development. External Team has 8 roles and the Core Team has 6 roles. In practice, one person can play multiple roles in one production run, and some roles can have many people assigned to them.

The External Team can include roles played by the client. In our case, the Subject Matter Expert was the instructor who had lectured the course earlier and who was also responsible for the new Web based course. Other positions were from IBM. Production Manager has the overall project control. Other expert positions were Media Architect, Technical Specialist, Audio/Visual Specialist, Production Artist, Photographer, and Quality Tester. Audio/video specialist also found a suitable voice to present all spoken text. In our case, we as the client were also heavily involved in quality testing during the project.

The Core Team is internal to IBM. An Instructional Designer is a specialist in e-learning and (s)he also works as the liaison between the client and the core team. A Courseware Engineer is technically responsible of putting together the final product with various software tools. Content Developers do most of the actual production when they transfer the Detailed Design Documents into final e-learning material with the Knowledge Producer system. Content Analyst keeps the Content Developers in line so that the final product is what the Subject Matter Expert wants. Media Coordinator assists in integrating different medias (graphics, animations, video, audio) and Quality Coordinator checks that all internal and external standards and guidelines are being followed in production phase.

3 Knowledge Factory process

The production process has many phases which each produce some specific document detailing the work done so far. Each phase can be implemented with different people manning the roles described above, and the documents produced in one phase are used as specifications for the subsequent phase.

Before the real production begins, one must analyse current material and goals. This determines what type of material we want to produce. Possible deliverables include Web books, material for collaborative learning, and material for Web lectures. One must also consider whether any electronic performance support system would be used. We must meet student requirements, such as can everyone log in at the same time, or do they all have access to a multimedia lab with high-speed network. All this will finally end up with consensus on what type of e-learning material we want to produce and how that material will be used in actual web course. Many instructional content delivery types can be used together, so that each instructional module is delivered with a method best suitable for it. For example, one can mix and match Web books, Web lectures and stand-alone simulators.

In our case, we decided to produce interactive Web books. One essential requirement in some parts of the course was to have a live database connection.

The first production phase delivers a High Level Design Document (HLDD) for the whole course. It states the overall goal as well as other client expectations for the product. It gives the course outline, look and feel, modules and which order the modules must be studied. It also specifies attributes for the target audience so that the final product will be suitable for it. For example, course size, student nationalities, sex, age, reading level, education level and computer literacy will affect the type of material developed and how it will be used. HLDD locks in menu structures and general navigation style to be used everywhere.

HLDD can specify an instructional theme (E.g., Pizza Restaurant with cartoon style approach) that will then be carried through the whole material. An instructional strategy and performance objectives are specified for each module and each topic. Learning goals determine when students can proceed from one module to the next. A general implementation framework must be agreed upon, so that each designer will know which functionality can be used. For example, whether voice should be used or not. In our case, KP and LLS were going to be used. Thus only the functionality within KP could be asked for.

HLDD is produced in a series of meetings with instructional designer and the client, with lots of help from the experts in the External Team. In our case, the HLDD was 27 pages long.

After HLDD is completed, each module is independently specified as a Detailed Design Document (DDD). The DDD gives a description for each view (page) that the student might encounter during the course. The views are drafted (e.g., with PowerPoint) with enough information for the content developers to produce the final product. Each view contains location information (e.g., module, topic, sub-topic) as well as the instructional information (e.g., "voice: database is ...", or "hotspot: show animation XYZ here"). Only functionality given in the HLDD framework can be used.

The DDD's are produced in meetings with Instructional Designers and Subject Matter Experts, with other External or Core Team members consulting. The Subject Matter Expert knows what are the instructional goals in each module, and the Instructional Designer does the drafting while all the time considering the limited functionality in the agreed upon framework. Other experts are consulted (e.g., by phone) when needed.

The HLDD and the DDD's are given to content developers and the rest of the core team for actual production. Core teams are a limited resource and one must book them well in advance.

The overall product is then complete, and the External Team Quality Tester will check it before it is given to the client for final evaluation. Problems are then resolved by iteration and/or consulting previous agreements stated in the HLDD and DDD's.

Finally, the product is ready to be used either at client's own environment, or at some IBM server accessible to clients and their students. In our case, the course server was installed in IBM's LLS server.

Although this process may seem large and cumbersome to university instructors accustomed to designing and implementing their own course material alone, we found it to be manageable in size as well as easily adjustable to our own specific needs.

4 Experiences with producing the course material

Introduction to Databases was selected as the pilot course for the TUeLIP project. The course was divided into 18 modules. These were distributed among 4 universities. We in Helsinki were responsible for 5 modules including Relational Algebra and three modules on SQL. We were ready to proceed fast, thus our first module became the pilot for the pilot. Dividing the responsibility of the course among many persons caused problems on the schedule. Soon it became evident that all 18 modules could not be prepared by the deadline. Actually only 8 modules has been implemented by now, and how and when to continue is still open.

We started with the Relational Algebra module. This is traditionally lectured in two hours accompanied with a practice session and home works. The starting point for making the module was the Finnish manuscript material (mainly in MS-Word format) made for the relational algebra part of the Introduction to Databases web course in Helsinki. This course material had been developed during spring and summer 2001 and the first courses using that material were to take place in autumn 2001. Thus we did not have any experience of the use of this material in a Web course. We did not know what kind of input was expected for the KF process. However, we decided to translate our Finnish manuscript in English and made only minor changes on the contents. Some examples and tasks for students were slightly modified. The manuscript contained textual parts, drafts of images, and outlines for animations to be included to illustrate the relational algebra operations. There were many mathematical formulas in the material, and it used a lecture like pedagogic style. The topics were first explained and then practiced.

The transformation of the MS-Word manuscript into a collection of HTML-pages had been straightforward in our Finnish web course. The formulas were transformed into images, and the text was split on web pages so that each relational algebra operation had a page of its own. Pop-ups and links were implemented as defined in the manuscript. Basically it was just a technical transformation. A technical writer in our university did the transformation and negotiated with the author when necessary.

The KF process used in building the TUeLIP module was different. First we spent a lot of time in specifying what should be the outcome of the course/module development effort. The specification concentrated on higher level issues, goals and motivations. Actually, most of this specification should have been done once for the entire course, but now it was done for each course module. We found out that the development tool placed restrictions on the presentation of the course material. The material was to be split into small non-scrollable screens, basically comparable to PowerPoint slides.

There were two attempts to build this module. The first one followed the manuscript strictly. The design task was, in principle, to split the material into a sequence of pages. To keep the students active, additional reflection questions were added into the material. Instructional designers were requesting for some general theme or background story for the module and proposed the Pizza taxi company that was used in some examples of the manuscript. We had to change some examples of the manuscript to fit with the theme.

The design was outlined as PowerPoint slides. The slides contained all material intended for the page and comments and instructions for subsequent phases of development. We started the design as a session lead by the Instructional Designer. She was not familiar with the topic. During the two day session we designed the HDDL, the overall structure of the course and about 20 DDD course pages. The Instructional Designer copied material from the manuscript, pasted it on PowerPoint slides, negotiated with us and wrote down instructions. After working for a few hours this way we came to a conclusion that we could do the splitting of material much faster by ourselves, especially because this module had many structurally similar parts and we had already covered two of them. So we continued on our own. After we had split the material into slides and included some additional tasks, the Instructional Designer took over the material, which at that time consisted of about 150 slides and 30 pages of animation scripts. The material was intended not only to replace lectures but also the time spent on reading text book, doing homework and attending practice sessions. After still one full day session and many e-mails and phone calls the final design was produced. It consisted of about 150 slides but the proposed animations were reduced to sequences of steps. Graphics were not present and it was impossible to see how to navigate through the course material.

A brief demo that contained the explanation of the example database and one relational algebra operation was ready in October 2001. The demo was fine. The amount of resources used in developing the demo module affected resource allocation for later modules. Furthermore, the number of modules to be developed was reduced.

The first delivery of the pilot module on the Lotus Learning Space environment took place in January 2002. Most of the module was fine, but there were a lot of small bugs, especially in formulas, and navigation did not work as we had assumed. Query building part of the module was a disaster. The theme (pizza taxi) and the example database were overemphasized.

In January 2002 another KF team took over the module. Database experts who knew also about the topic were included in the course development teams. The new team had new ideas. They proposed changing the pedagogic style to induction learning (Holland et al., 1986) so that the students would have to deduce the operations and the formulas based on examples presented to them. Originally, the operations and their formulas were introduced first and practiced after that. The style was changed and we had to figure out new examples for introducing the concepts In the first version only some pages contained audio. Now it was included in all pages. Graphics and navigation were redesigned. New decorative graphics were inserted. A second delivery took place in March 2002. We had asked for browser and operating system independency, but it became obvious that Internet Explorer in Windows environment was the only browser that could be used. There were still some navigational and some other minor problems. No animations were included and the conclusion part of the module had some problems. But mostly it was OK, and this was the module to be tested in experimental courses.

We made also another TUeLIP module, about SQL data definition. This module was simpler than the relational algebra module. It did not use induction learning, but used a more traditional approach instead. The first delivery in April 2002 was satisfactory.

Other universities prepared six modules during spring 2002. They did not have as much material prepared in advance as we had, and there were some problems in fitting their schedules and resources with the KF team. In spite of distributed development the modules fit together quite well.

5 Experiences with the material

It became obvious that only a few modules of all the designed ones would be ready during spring 2002. However, there was urgent need to experiment with the modules during spring or summer 2002. It was decided that the universities involved would arrange courses, where parts of the course would be based on completed TUeLIP modules and the rest would be covered by lectures or by some other means. In Helsinki we experimented with a course for foreign students in May. The course had 6 hours of lectures, and four TUeLIP modules were used, two of which were our own. In addition, we translated the SQL-part of our Finnish web course into English. The translation was done in order to prepare for the future TUeLIP modules assigned to us. Because the look and feel of the SQL-modules was not the one used in the TUeLIP modules, we called them pre-TUeLIP modules. This part of the course used live database through the SQL-Trainer (Laine, 2002) practicing tool. We consider to include this tool also to the TUeLIP versions of these modules, although it cannot be linked fluently with LLS.

There were 15 students registered for the course, but only 11 students ever logged in the Lotus Learning Space server, and only 9 students were logged in the SQL-Trainer server in the latter part of the course. Finally, 8 students took the exam and all of them passed the exam. At the same time there was an independent exam for the same course and we used the same questions for both exams. Independent exam was taken by 17 students and 10 of them passed it. However, we cannot draw any far-reaching conclusions about these statistics. The exam had only one question about the topics taught in the four TUeLIP modules. Also, independent exams are typically taken by students that have failed to pass the course earlier. There are also students that try to pass the course without practicing at all. Some of the students that passed the independent exam had studied the course using the Finnish web course material.

Students reported some technical problems in the use of the Lotus Learning Space. Strict untold assumptions about the Windows setting caused problems for some students. Discussion groups did not work for all students. Nobody succeeded in providing feedback through LLS forms even if they tried. However, students were mostly satisfied with the material. Some considered audio as unnecessary, but others liked it. The tasks in TUeLIP modules were considered easy as compared to the SQL-Trainer tasks that they had to pass to get credit.

6 Summary and future plans

Knowledge Factory process is a well-defined process. It has many roles and phases. Tasks and roles in the IBM's process reflect good understanding of how to build an e-learning course. We feel this model suits well for companies doing rather small e-learning courses without having people dedicated for education. However, university courses are typically large. The subject matter experts are also experts on teaching the matter. We do not exactly know how much developing of these pilot modules cost. But our guess is that developing university courses this way is far out of our economic possibilities. To find out proper ways to combine the roles, tasks and phases, and to find proper people for those roles is essential to develop a more cost effective way of course building. Fully utilizing the capabilities of teachers and providing them support when needed might be the solution. To provide support for instructors our department has accepted an e-learning strategy that defines two support units: a pedagogic support unit and a technical support unit.

To complete the Introduction to Databases course one possibility is that universities try to use the KP tool by themselves to build the still missing modules of the prototype. We will also experiment with other content production systems in near future.

References

Holland, J. H., Holyoak, R. E., Thagard, P. R., 1986. Induction: Process of inference, learning and discovery. MIT Press, Cambridge, MA.

IBM, 2000. IBM Knowledge Factory. CD-ROM.

IBM, 2002a. IBM Mindspan Solutions.
URL `http://www.ibm.com/mindspan`

IBM, 2002b. Lotus Learning Space.
URL `http://www.lotus.com/learningspace`

Laine, H., 2002. SQL-Trainer. In: Proc. of the 1st Annual Finnish/Baltic Sea Conference on Computer Science Education. University of Joensuu, Department of Computer Science, pp. 13–17, Report A-2002-1.

Electronic course material on Data Structures and Algorithms

A. Korhonen, L. Malmi, P. Mård, H. Salonen, P. Silvasti
Helsinki University of Technology
Department of Computer Science and Engineering
PO Box 5400, 02015 HUT
{archie,lma,pmard,hsalonen,psilvast}@cs.hut.fi

1 Introduction

Electronic textbooks and other course material built on internet technology are becoming widely accepted. From the student's point of view such learning material can be divided into two categories. First, passive *instructive material* includes text, pictures and ready-made algorithm animations. Second, *constructive material* allows the learner to interactively modify ready-made examples, and also design and explore examples of his own. We consider this latter form of material to be more valuable because it increases the learner engagement. There is also empirical evidence suggesting that learners do better through active rather than passive activities (Lawrence et al., 1994), Hundhausen et al. (2002). In this paper we focus on tools and techniques to produce (inter)active course material for computer science education, especially on Data Structures and Algorithms.

Several approaches have been introduced to promote constructive learning material in which the learner is actively involved in the learning process. One example is the support for learner-built visualizations such as reported by Stasko (1997). There the learner is completely involved in the visualization process. However, the method lacks the possibility to give immediate automatic feedback on one's performance. Similar activity and involvement can be achieved by algorithm simulation functionality, which we shall discuss later in this paper.

Another example is JCAT (Brown and Raisamo (1997)) and its predecessors. The systems are designed for creating active and collaborative electronic textbooks intended to classroom setting during a lecture. In such a setting, an instructor controls the animation, while students view the animation by pointing their Web browsers at the appropriate page. In our approach, however, the aim is to activate the learner by providing material that could be studied outside classroom at one's own pace. In such a case the problem is how to gather information on the human interaction with the system and how to motivate the learner to explore the provided material. We can tackle both of these problems by introducing exercises that the learner should solve during the session. The method described in this paper combines the support for custom input data sets (Brown, 1988) and dynamic questions such as introduced, for example, in JHAVÉ (Naps et al., 2000) and "Algorithms in Action" (Stern et al., 1999).

Similar to Interactive Exploratory Algorithm Learning introduced by Faltin (2002) we allow the learner's to explore the possible states of a data structure, but we can also provide exercises and immediate feedback on the learners performance during the exploration. Faltin does not promote exercises. According to our approach, algorithm simulation works as an engine to activate the learner to continuously interact with the system. The exercises serve as a basis for such an interaction in a sense that we can provide motivating problems to solve and to work with. We can also produce the correct model solutions for any given input data set. Thus, by comparing the model solution to the simulation process we can provide immediate feedback on the correctness of the learner made simulation.

Some of the problems we are tackling here are similar to those identified by Ross and Grinder (2002) in their paper discussing Hypertextbooks. They argue that constructing algorithm animations for the use of students is different than using algorithm animations for demonstration purposes on lectures. We agree but in this paper we take this even further by setting up a pedagogical framework for designing interactive electronic books.

Section 2 presents the terminology of electronic textbooks. In Section 3 we discuss the pedagogical issues behind designing textbooks and present a model schema for a book in the area of data structures and algorithms. Section 4 presents implementation issues of our prototype and finally Section 5 summarizes our contributions.

2 Elements in an Electronic Textbook

Electronic textbooks include the following elements, each of which has a different role for the reader. *Text* is used for defining concepts and explaining their meaning and properties. In addition, text is naturally used for explaining and discussing applications where concepts are used. *Figures* are used for clarifying the structure of concepts, as well as the relations of different concepts. Figures are also useful for explaining processes. A *sequence of figures, a comic strip* is a way of showing the phases of an activity. Its advantage is that the reader can easily track the changes in the process in his/her own phase. *Animation* is a dynamic form of the previous one, in which figures are shown one after another in some predefined time schedule. Transitions between two adjacent states can be shown as a smooth movement of objects, or as a straightforward transition. Often the user is able to *control* the speed of animation, as well as to run it backwards. Also the level of details shown can possibly be specified.

Up to this level the role of the student is essentially passive. He is reading, looking at, or observing given information. This holds, even though there may be options of controlling the speed, style and granularity of the visualization. The following elements include and support the constructive process. *Simulation* is an activity in which the student uses his conceptual knowledge to explore the working of a system by changing the application-specific data or data structures and observing the consequences. *Passive exercises* are exercises in which model solutions may be available using any of the elements listed above, but no on-line evaluation is carried out. *Active exercises* are activities in which student chooses, modifies or creates information, submits it for evaluation and gets feedback automatically.

In addition, electronic textbooks may contain *navigation elements*, which promote user's ability to navigate smoothly in the book, and *tracking elements* which record and log information about user's activities. The user may see this information himself, for example, as a cumulative sum of points, or a list of completed sections and exercises. Some tracking information can be collected by the system designer or the teacher to get information about how the book is used, or how the student progresses in his work.

3 Designing an Electronic Textbook for Data Structures and Algorithms

3.1 Pedagogical Issues

A textbook should never be designed without a strong pedagogical view about the topic. We therefore consider briefly some phases of learning, which are relevant in this context. First, students need *motivation* to understand the necessity of new information. Second, they need some kind of *overview* of the topic. Third, they have to *internalize* information, *i.e.*, to learn details and attach them to the whole. Fourth, they have to *apply* the learned new information to solve problems in order to master it. Finally, some kind of *evaluation* or *control* of their learning is needed.

All these phases should be considered when designing a textbook, and different elements like plain text, figures, animations, simulations and exercises should be carefully chosen to support the various phases of learning.

3.2 Usability Factors

Ross and Grinder (2002) point out several technical and usability factors that have to be taken into consideration when producing applets for student use and embedding them into an electronic book in a consistent way. 1) Applets must be simple enough for novice students to use. 2) There must be controls for stepping the animation back and forth and playing the whole animation at a time. 3) It should be possible for students to experiment algorithms with different data sets and to create new animations from scratch. 4) For teachers, there must be a tool for creating new algorithm animations easily, as well. 5) The applets should be uniform and exchange data with each other, so that an example animation a student has created in one applet could be augmented in the next applet. 6) The internal data structure of the animation should be separated from the graphical representation, which makes it possible to check student created animation against a model solution.

3.3 Model of an Electronic Book for Data Structures and Algorithms

Here we give a sample vision of a chapter in a book discussing balanced binary search trees.

The chapter begins with a textual explanation of the properties of binary search trees. This part is accompanied with both figures of badly balanced trees and animations producing bad cases. The student can use simulation to verify the behavior with his own data. These activities should give him motivation to study how to prevent such behavior.

Next, a brief overview of different strategies for carrying out balancing, like global rebuilding and local adjustments are explained. Plain animations with the speed controlling option about different strategies are shown to give a more concrete flavor of them.

In order to internalize relevant structures, a detailed description of AVL trees, red-black trees and B-trees follows. Example animations with textual explanations are given. The student is provided simulation assignments where he should apply this information to build a data structure with given random data. He has the option of getting the evaluation of his sequence as a feedback, after which new random data for the assignment is given. Finally, he has an option to send his sequence to the server for controlling evaluation.

After the algorithm is understood on a conceptual level, the actual implemented code is shown and explained textually to give a deeper understanding of the algorithm. Accompanied with this there are examples where the student can follow and control code visualization together with the conceptual visualization. These visualizations are instructive, but in them the student does not have to apply the new information. Therefore, new assignments follow, where he should by means of algorithm simulation provide an input sequence, which activates certain parts of the code, for example, a left-right rotation. Again, evaluation feedback on his sequence is given graphically by showing which parts of the code were executed (in each recursion level, if codes are given using a recursive algorithm).

The pedagogical elements mentioned above could also be used to build an adaptive textbook. The information shown to the student, as well as the functionality allowed to him would be dependent on his previous performance. In a simulation the balancing rotation operations would be allowed only after the student has mastered correctly the basic binary search tree operations. Similarly AVL tree insertion/deletion operations were allowed only after the student has mastered the rotation exercises. Details of algorithm implementation would be shown after the student understands the algorithm on a conceptual level.

4 About Prototype Implementation

We have implemented a prototype, which partially fulfills the schema presented in the previous section. The book includes traditional passive elements (text and figures) and active constructive elements, such as interactive animation apples, algorithm simulation applets and exercise applets. In this section we briefly discuss some technical issues about their implementation.

All exercises and animations are applets that are based on the Matrix framework (Korhonen and Malmi, 2002; Korhonen et al., 2001). Matrix is a framework for creating visualizations and animations of data structures, and for simulating algorithms by interactively drag&dropping graphical objects on the screen. Using the same framework allows us to easily create similar looking applets with identical user interfaces and functionalities.

All prototype applets have a simple navigation bar and hyperlinks are used in navigating through the book or linking to other material. Animations and exercises are integrated into the text; a new window is opened only if there is a special need for it. Thus, the applets integrate seamlessly into the book.

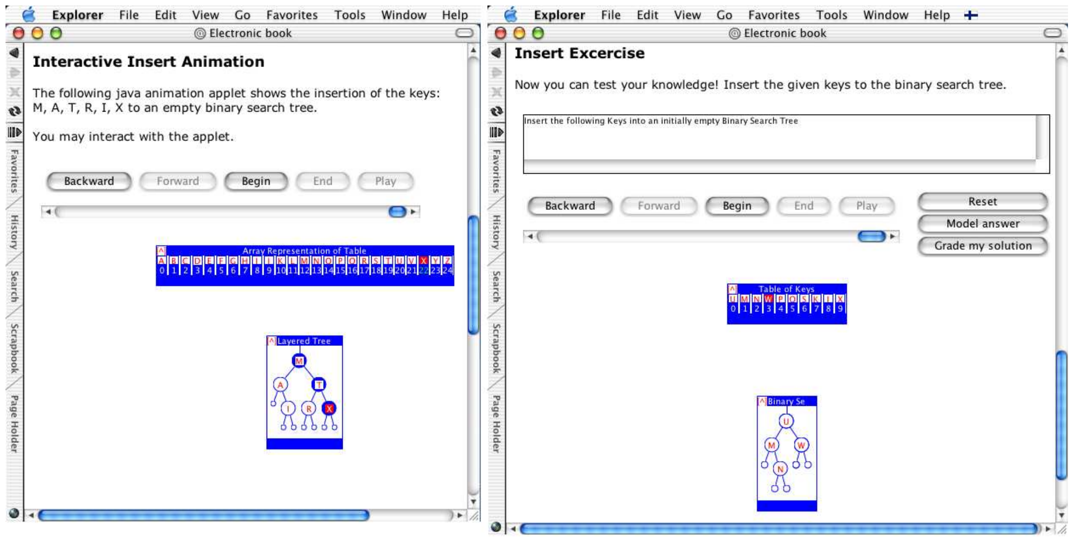
Algorithm animation applets (see Figure 1) are used for viewing data structures and algorithms in action. However, one can also do algorithm simulation with them thus allowing the user to explore the working of an algorithm more closely. This simulation facility may be explicitly emphasized in the text, or not, depending on the pedagogical function of the exercise. Each animation applet has a control bar allowing stepwise execution of the animation, both forwards and backwards. Alternatively animations can be converted into Scalable Vector Graphics format, which enables smooth transitions between adjacent steps.

Exercise applets (Figure 1) are designed for solving exercises in terms of algorithm simulation, *i.e.*, by graphically manipulating data structures. As in animation applets the user can backtrack the simulation sequence, for example, to correct an error. The exercises are assessed automatically by a press of a button, and instant feedback follows. The model answer of the exercise opens into a new window so that it is possible to view it side by side with user's own solution. With exercise applets one can test his own understanding of the particular algorithm. The course instructor can also view answers submitted by students as step-by-step animations.

The applets and any other building blocks of the electronic book do not depend on each other and they can be easily used elsewhere. For example, the applets could be included in instructor's slide set to be shown separately. We do not have code animation applets at the moment. However, this feature of Matrix has been demonstrated in Korhonen et al. (2002) and we are working on incorporating it into the applet versions.

5 Summary

We summarize our contributions as follows. First, we have represented a novel way to enhance the level of interaction and engagement between the learner and the electronic textbook he is studying. When learners use our interactive electronic textbook of data structures and algorithms, after reading the text they can deepen their understanding by looking at algorithm animations embedded in the textbook. The speed of continuous animations can be adjusted. Students can also step through the execution of the animation as discrete steps. Moreover, students can design input data for the algorithm and see how the animation executes with different data sets, and thus explore the behavior of the algorithm more closely. Finally, students can solve exercises in which they perform the steps of an algorithm manually by


Interactive Insert Animation

The following java animation applet shows the insertion of the keys: M, A, T, R, I, X to an empty binary search tree.

You may interact with the applet.

Insert Excercise

Now you can test your knowledge! Insert the given keys to the binary search tree.

Insert the following Keys into an initially empty Binary Search Tree

Figure 1: An animation applet and an exercise applet.

directly manipulating the data structure through the applet GUI. The applet assesses the steps student has performed and gives instant feedback of the correctness of the simulation. In this way the student can easily test his understanding of the algorithm.

Second, the current framework is capable of delivering algorithm animations in standard or *de facto standard* formats. Thus, in the future it is possible to exchange study material between web-based learning environments more easily. A piece of animation can be adopted from one environment to another and viewed with 3rd party viewers without any extra effort.

Third, by introducing exercises we gain a natural way to gather information about learners' activity. The applets embedded into the electronic textbook provide means for collecting data of students' performance. The efficiency of the material can be assessed not only by how far the learner proceeds in the material but also by means of correct solutions for exercises. In addition, while learners perform steps in an exercise manually, we could track what kind of errors they make. Are there any errors that seem to be repeated more often than others?

References

Brown, M., 1988. Algorithm Animation. MIT Press, Cambridge, Massachussets.

Brown, M., Raisamo, R., 1997. JCAT: Collaborative Active Textbooks Using Java. Computer Networks and ISDN Systems 29 (14), 1577–1586.

Faltin, N., 2002. Structure and Constraints in Interactive Exploratory Algorithm Learning. In: Software Visualization: International Seminar. Springer, Dagstuhl, Germany, pp. 213–226.

Hundhausen, C., Douglas, S., Stasko, J., 2002. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing 13 (3), 259–290.

Korhonen, A., Malmi, L., May 2002. Matrix – Concept Animation and Algorithm Simulation System. In: Proceedings of the Working Conference on Advanced Visual Interfaces. ACM, Trento, Italy, pp. 109–114.

Korhonen, A., Nikander, J., Saikkonen, R., Tenhunen, P., November 2001. Matrix – Algorithm Simulation and Animation Tool. http://www.cs.hut.fi/Research/Matrix/.

Korhonen, A., Sutinen, E., Tarhio, J., 2002. Understanding Algorithms by Means of Visualized Path Testing. In: Software Visualization: International Seminar. Springer, Dagstuhl, Germany, pp. 256–268.

Lawrence, A., Badre, A., Stasko, J. T., 1994. Empirically Evaluating the Use of Animations to Teach Algorithms. In: Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO. pp. 48–54.

Naps, T. L., Eagan, J. R., Norton, L. L., March 2000. JHAVÉ: An Environment to Actively Engage Students in Web-based Algorithm Visualizations. In: Proceedings of the SIGCSE Session. ACM, Austin, Texas, pp. 109–113.

Ross, R. J., Grinder, M. T., 2002. Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resource for the Web. In: Software Visualization: International Seminar. Springer, Dagstuhl, Germany, pp. 269–283.

Stasko, J., 1997. Using Student-Built Algorithm Animations as Learning Aids. In: The Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education. ACM, San Jose, CA, USA, pp. 25–29.

Stern, L., Søndergaard, H., Naish, L., 1999. A strategy for managing content complexity in algorithm animation. In: Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education, ITiCSE'99. ACM Press, Kracow, Poland, pp. 127–130.

PAPER SESSION 2

Learning by Gaming in Adaptive Learning System

W. Hämäläinen and N. Myller
Department of Computer Science, University of Joensuu, P.O. Box 111, FIN-80101 Joensuu, Finland

J. López-Cuadrado
Department of Computer Languages and Systems, University of the Basque Country, Aptdo. 649, San Sebastian 20080 Guipuzcoa, Spain

S. H. Pitkänen
Educational Technology Centre, University of Joensuu, P.O. Box 111, FIN-80101 Joensuu, Finland

whamalai@cs.joensuu.fi

Abstract

We present a new idea how to develop a learning environment that could motivate students to keep on learning. We introduce the idea of learning environment that supports learning by gaming but still gives students the possibility of collaboration. We explain the generation of new problems for students via genetic algorithms so that the games can be adapted to each player's skills individually.

1 Introduction

Problem-based learning (Soini et al., 1998) is usually considered very motivating, but we don't have any guarantees that the learning process goes on after solving the problem and getting the feedback. The critical question is: how to generate new challenging problems from the feedback or the learning process? Such problems should be individually adapted to the specific learner, strenghtening the weakest areas but still giving new realistic challenges to conquer.

We want to support collaborative learning. Especially if the learning happens without teacher, the guidance of the system or other learners is crucially important. It is also known that people learn by teaching others. The human lust for competitions should neither be ignored.

For these purposes we planned a new gaming environment, which is shared by all the students (players) but which still generates individually adapted problems for each student. In our system we use genetic algorithms to generate new individual problems for each student.

Our system will be meant for teaching theoretical foundations of Computer Science, i.e. regular expressions, grammars, automata and Turing machines. We shall represent the idea of generating new problems for regular expressions and corresponding automata with genetic algorithms.

In this paper we shall first introduce related work and our game environment as well as a few example games. After that, we shall consider the idea of how to generate new problems with genetic algorithms. Finally, we shall evaluate the possibilities of our system.

2 Related work

The idea of using games in an educational environment as a learning method is not new. In fact, many teachers and pedagogues worry about how to make this integration possible and when talking about e-learning the same thing applies. Some authors (Randel et al., 1992; VanSickle, 1986) have concluded that simulation games are at least as effective as other methods for teaching knowledge about facts, concepts and application of knowledge. We consider that using video-games can strengthen and support the learning process, especially within difficult (i.e. theoretical) domains.

A lot of work has been made in this respect. Some proposals for gaming and simulation-based learning architectures are available (Angelides and Paul, 1993), and adaptive gaming environments have been developed (Carro et al., 2002). It is even possible to make use of other existing environments, as EDUCO (Kurhila et al., 2002b), which is web-based and collaborative, or even some platforms as QUAKE (www.idsoftware.com), a fascinating well-known environment which provides all the properties required to implement collaborative and competitive gaming. In fact, we can find some experiences like this: for example, Marine fire teams have been training at computer labs in the USA, learning battlefield tactics and decision making, by using an adapted version of the commercial game DOOM (Prensky, 2001).

At this moment there are many software companies which are advocates of digital game-based learning, such as Games2train (games2train.com), which offers business training solutions in a wide set of games, or LearningWare Inc. (learningware.com) which provides funny TV-style competitive and collaborative games for classroom review and reinforcement.

3 Requirements for the gaming environment

In this chapter we present some criteria for the adaptive gaming environment providing collaborative learning possibilities from a set of games which are individually adapted to each user. Angelides and Paul (1993) have explained the elements of gaming-simulation in detail but we are not going to consider our requirements as deeply. However, the basic requirements are the same as Angelides and Paul's.

Firstly we discuss how we can prevent students getting lost, and after that we give some ideas about how to promote collaborative learning in our learning environment.

3.1 How to help students to find the right path?

If the whole collection of games would be available from the beginning of the gaming, many students could get lost. In other words, they would not know which game they should choose or how to proceed in this new environment.

One solution is to define several stereotypes or levels of difficulty such as beginner, intermediate and expert player. A student who wants to play on beginners' level will be able to choose only two or three games (the easiest ones), while expert players will have the whole collection available. Full explanation of the new games is given in beginners'level in order to introduce the idea and rules of the games.

When a player collects a concrete amount of points, or she gives some evidence of mastering some ability or skill (i.e. she solves difficult problems correctly), she will pass to the next level. This means that at this moment, a few new games will be available for her. On the other hand, if she selects a level which is too difficult for her, then she will be carried to a lower one. This fact might be detected, for example, when the player loses all her bonus points because she has bought lots of hints with them. These concept id quite close to the idea of Vygotsky (1978) where the student is kept in his/her zone of proximal development (ZPD). The learning space model that is implemented in adaptive learning environment called AHMED is close to Vygotsky's idea (Kurhila et al., 2002a).

3.2 How to promote collaborative learning from a set of individually adapted games?

At the very first stages of every game, the student will play alone. This will occur until she acquires some ability level. After this training stage the student will be able to play against or in collaboration with the other students.

In some games several students could play at the same time in a competitive way. The aim of the game in this case would be to win the others. In other games collaborative learning could be reached by playing in turns. In this way the answers and feedback of the system would be visible for everybody. The main idea in this is that after several trials somebody will find the correct answer, but she has used some hints that were feedback for the another players. These hints would have been unreachable for her if she had played the same stage alone. In some games cooperation would be possible for example asking for help from others or solving the problems together in real-time (Siekkinen and Huhtinen, 1999).

To make collaborative/competitive gaming more adaptive, players with similar level could play together and only those players would be exposed at the same time. The concrete problem to play with could be dynamically generated using teh recently solved problems of each player as the seeds for genetic algorithms which are discussed in Chapter 5.

After having played a certain game it would be important that players that have succeeded in it could give some tutoring or hints to those students who have not (Soini et al., 1998). This kind of collaborative tutoring between students can also be reached in a more general way with including highscore list in each of the games. We consider that it does not only motivate students to keep on playing (i.e. to reach the highest score) but it can also provoke some kind of collaboration between students. Concretely, if a student is unable to advance in a game, although she has visited the help examples many times, she will probably ask help from those players which are situated higher on the highscore list. In this way, more proficient students can instruct those with lower abilities and probably learn more by themselves as well (Dillenbourg et al., 1995; Rochelle and Teasley, 1995).

3.3 Possible starting points

As one possible gaming environment we could use EDUCO a learning environment supporting social navigation (Kurhila et al., 2002b). It is web-based and it can be used with any web-browser so the games can be added in the system in any form that the web-browser understands can be integrated to this system. EDUCO provides also a map where players can see each others movements and chat view where students can communicate with each other. It provides the basic features that one could need for described gaming environment.

The map view could be used to visualize the players differently depending on e.g. their progress and in this way students could know from whom they could ask help and with whom they could play collaborative games. In chat view students could discuss about the games and also ask help from other students.

Another possible environment is totally different. Most Computer Science students play very eagerly first-person shooter games like Quake in their leisure time. That kind of games have several elements, which motivate to play. The player is adventuring in an unknown world and has to find the way out. In the way one has to overcome monsters, find keys or secret levers to open the doors, utilize new weapons, first-aid kits or other available objects. In addition several players can play in the same game world and discuss with each other like in IRC.

That kind of game environment would be very stimulating for our purposes. Quake offers all the properties required for adapting collaboration game environment, but it is also fascinating for students, who enjoy playing the original Quake.

For example a monster can be a finite automaton, which has to be fed by strings it accepts, until it bursts. Or the game world can be constructed as a labyrinth, which has signs – strings – in the doors or walls and the key (a regular expression) guides to find useful corridoors or objects. Opening the wrong door (marked with a string, which does not belong to the given grammar) may cause some damage, e.g. let a monster free. Edible and poisonous food may also be marked with strings or equivalent expressions.

The old versions of Quake engines are freely available in Internet so that people all around the world may develop new ideas into them (http://www.quakeforge.net/). We could also put our original game environment to the net so that students could develop it further and invent new adventures (problems) concerning not only theoretical foundations of Computer Science but also other areas of Computer Science.

4 Descriptions of example games

Feeding a monster: You meet a monster, which is in fact a finite automaton. Feed the monster with strings, which it accepts, or it attacks you! The same string can be used only once. (See Appendex 1, Fig. 1)
Finding pairs: In the table there are several cards, with a regular expression in each. Find the pairs, which have identic expressions! In variations of this game the pairs may consist of identic finite automata (after minimizing the pairs represent the same automaton) or a deterministic and an nondeterministic automaton (determinizing is needed).
A labyrinth game: You are given a key, which is a regular expression. You have to find your way out off the labyrinth as fast as possible. In the walls of labyrinth there are strings. By following the strings, which are defined by your key expression, you will find the shortest way out. This game may also be a competition between two players or with the computer. (See Appendix 1, Fig. 1)
Picking mushrooms: You enter a square in forest, where mushrooms grow. Some of them are eatable and give you points (more strength, money etc), but others are poisonous (you lose points). The mushrooms are marked with strings and you should invent a regular expression, which defines only eatable mushrooms, but not the poisonous.

5 Genetic algoriths in generating new problems

The main feature of our self-motivating learning environment is to generate new problems from the feedback earlier games. The new problems should be adapted to the players current skills. The most difficult exercises should be practised more, but too difficult exercises should also be avoided. We have used the rule that after every solved problem a new harder problem will be generated, based on how many trials were needed to solve the earlier problems. For this purpose we have used genetic algorithms.

In the genetic algorithms the population of some individuals develops through simulated evolution. Each individual is represented by a finite string of symbols (genome), which usually encodes a solution in a given problem. In the iterative process of evolution a selected part of population, the parents, are crossed to form new individuals, the offspring. The crossover is performed by exchanging parts of parents' genomes (substrings). In addition the genomes may be mutated with some small probability, for example some symbols are changed randomly. From the new population the best ones are again selected for the further evolution.

The main difference in our game environment is that we are not generating new solutions but new problems. It is easiest to generate new problems in the level of regular expressions. (When we add problems about context-free grammars/pushdown automata and Turing machines, the corresponding grammars will be processed.) Crossing and mutating the transition functions of an automaton doesn't guarantee that we get another same kind of automaton (i.e. in the same level of difficulty). Instead, the difficulty of regular expressions can be saved and the regular expressions can always be translated to corresponding automata.

The goodness (fitness) of a regular expression depends on a player and is evaluated according the player's game history. The most difficult type of expressions are selected for crossover and mutation to get new similar type of problems. Usually students feel the most difficult such expressions, which consist of several alternative parts (separated with union operator) and which have closures of subexpressions (* operators, like $(ab \cup ba)*$). Crossing such expressions produce probably new same kind of operations.

In crossing the parent expressions are divided in two parts in some random point, still considering the paranthesis (no missing paranthesis are allowed). The parts are combined and new expressions are mutated. In mutation a symbol can be changed (e.g. a is changed to b or ϵ), a union operation can be added or deleted or the exponent can be changed (0, 1 or * i.e. the base is missing, present as such or given *).

Usually the crossover between parents is performed with some probability ("crossover rate"), but in our case the last two expressions of the same difficulty level may be always crossed. Also the mutation usually takes place only with very small probability, but in our game large variation is desirable and mutation can be applied to several parts of the expression with some quite high probability.

For example, let's suppose the parent expressions are $r_1 = a{*}b{\cup}b{*}a$ and $r_2 = ba{*}b{\cup}aa{*}b$. They are divided into parts $a*$, $b{\cup}b{*}a$, $ba*$ and $b{\cup}aa{*}b$. After crossing we get new expressions $r_3 = a{*}b{\cup}aa{*}b$ and $r_4 = ba{*}b{\cup}b{*}a$. The new expressions are mutated in two random points and we get $r_3' = a{*}bab$ ($\cup$ is deleted and the exponent * is changed to 0) and $r_4' = b{\cup}a{*}b{\cup}b{*}a*$ ($\cup$ is added and exponent 1 is changed to *).

6 Conclusions

Games have been used for a long time in teaching, but often just for fun. Gaming and structures of games can be used in many ways, also applied to teaching and learning of theoretical knowledge in any kind of domains and levels of studying. The domain defines the way in which it is most optimal to construct new knowledge structures. It does not need to be funny to learn, but for reaching the best quality of learning it should feel like an experience. One has challenges, a little bit of competition to maintain one's self-motivation, one benefits from it, one gets feedback from one's actions and one notices that one can help other students. All these things are also elements of games.

Adaptive systems give every student their own learning experiences. This way, everybody feels like individual, not one of the mass, and cooperates with each other. If there is not enough teaching capacity, the adaptive systems can give feedback like humans: individual and just suitable for students achievements. For that purpose the system must collect accurate student's learning history and analyze it in an appropriate way.

This kind of learning situation gives a lot of research possibilities for the learning domain itself, constructing knowledge structures in the domain, user modelling, artificial intelligence and educational sciences.

Acknowledgements. We thank Quake expert Kimmo Fredriksson and information technology specialist Jani Jaakkola for expertizing us on Quake matters.

References

Angelides, M. C., Paul, R. J., 1993. Towards a Framework For Integrating Intelligent Tutoring Systems and Gaming-Simulation. In: Evans, G. W., Mollaghasemi, M., Russell, E. C., Biles, W. E. (Eds.), Proceedings of the 25th conference on Winter simulation. Los Angeles, United States, pp. 1281–1289.

Carro, R. M., Breda, A. M., Castillo, G., Bajuelos, A. L., 2002. A Methodology for Developing Adaptive Educational Game Environments. In: Bra, P. D., Brusilovsky, P., Conejo, R. (Eds.), Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems. Vol. 2347 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, pp. 90–99.

Dillenbourg, P., Baker, M., Blaye, A., O'Mailley, C., 1995. The evolution of research on collaborative learning. In: Spada, P. R. . H. (Ed.), Learning in humans and machines. Towards an interdisciplinary learning science. Elsevier, Oxford, pp. 189–211.

Kurhila, J., Lattu, M., Pietilä, A., 2002a. Using Vector-Space Model in Adaptive Hypermedia for Learning. In: Cerri, S., Gouardères, G. (Eds.), Intelligent Tutoring Systems 2002

(ITS'2002). Vol. 2363 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, pp. 129–138.

Kurhila, J., Miettinen, M., Nokelainen, P., Tirri, H., 2002b. EDUCO - A Collaborative Learning Environment Based on Social Navigation. In: Bra, P. D., Brusilovsky, P., Conejo, R. (Eds.), Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems. Vol. 2347 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, pp. 242–252.

Prensky, M., 2001. Marines Learn at the Speed of Doom. Learning Circuits, ASTD's Online Magazine All About E-learning Http://www.learningcircuits.org/2001/feb2001/doom.html.

Randel, J., Morris, B., Wetzel, C., Whitehill, B., 1992. The Effectiveness of Games for Educational Purposes: A Review of Recent Research. Simulation & Gaming 23, 261–276.

Rochelle, J., Teasley, S., 1995. The construction of shared knowledge in collaborative problem solving. In: O'Mailley, C. (Ed.), Computer supported collaborative learning. Springer-Verlag, Berlin, pp. 69–97.

Siekkinen, M., Huhtinen, R., 1999. The Use of Computer in Finnish Preschool Settings Toy or Tool for Children? In: Levonen, J., Enkenberg, J. (Eds.), Learning and Instruction in Multiple Contexts and Settings. Proceedings of the Second Joensuu Symposium on Learning and Instruction. Vol. 73 of University of Joensuu Bulletins of the Faculty of Education. University of Joensuu.

Soini, H., Koivula, S., Ropponen, L., Tensing, M., 1998. Peer Collaboration in Higher Education - Peer Consultation as a Method for Promoting Students Self-Regulation and Small Group Learning. In: Järvelä, S., Kunelius, E. (Eds.), Learning and Technology dimensions to learning processes in different learning environments. Vol. 1 of Electronic Publications of University of Oulu. University of Oulu, pp. 77–89, http://herkules.oulu.fi/isbn9514248104/isbn9514248104.pdf.

VanSickle, R. L., 1986. A Quantitative Review of Research on Instructional Simulation Gaming: a Twenty Year Perspective. Theory and Research in Social Education 14, 245–264.

Vygotsky, L., 1978. Mind in Society: The development of higher psychological processes. Harvard University Press, Cambridge, MA.

Appendix 1

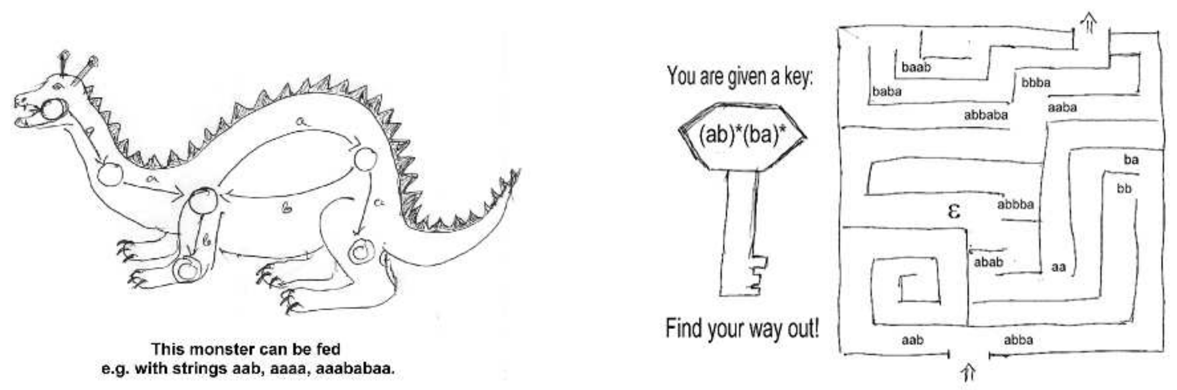


Figure 1: Some sketches for example games.

Increasing student participation through online virtual environment

Harri Hämäläinen, Jouni Ikonen and Jari Porras
Lappeenranta University of Technology, P.O.Box 20 53851 Lappeenranta, Finland

{harri.hamalainen, jouni.ikonen, jari.porras}@lut.fi

1 Keywords

Virtual education, online virtual environment, student participation, wireless technologies.

2 Introduction

The actual problem in a normal learning environment is the lack of real-time feedback. Many of the students are too shy to participate actively on learning by commenting the presentation, proposing their ideas or asking questions. Especially on courses with tens or hundreds of students this is a large problem. Teachers may not notice students' weak knowledge on a particular issue until verifying their exams - which is definitely too late. A possibility to react earlier for this would be useful for both students and teachers.

The goal for the project was to implement instruments that help to increase the interaction between the teacher and the students by using modern technologies. Technologies themselves are not the main concern but they provide one possible approach for the given goal. The implemented virtual environment should be utilized during the lessons to give the students a chance to have a real-time effect on the lectured topics.

The advantages from teacher's point of view are incontestable. Teachers often have problems to estimate their student's awareness of current issue. Often they explain issues they see hard to understand very precisely and do not necessarily pay attention to the easier issues. Occasionally even these issues may be hard to understand. Our approach does not remove the problem, but hopefully makes it easier for the teacher to realize it.

As teaching is discussed in this publication, it should not be considered to be limited to a high school or university environment. The feedback system can be applied to corporate environments or even kindergarten as suitable technology is likely to become available in the near future.

3 Related work

One of the first steps on virtual interactive lecturing was a "Classroom Communication System". It allowed the teacher to present questions during the lecture and see the diagrams. It was highly hardware dependent accepting only a certain type of devices with a fixed wired network. It mainly replaced the traditional quizzes with an automated approach. It solved only some parts of the whole problem. (Dufresne et al., 1996)

One step forward is the "Student Response System (SRS)", a server-side web application, which requires only a web browser on client side. Quizzes can be answered with handheld computers via wireless networks. The results can be seen as charts and can be projected to students. This approach only replaced wired links with wireless ones. (University of North Carolina at Wilmington and Hypercube, 2002)

"Ubiquitous Computing in Education (UCE)" is a project targeted also for non-present participants. The implementation is built to be run in Java Virtual Machine to maintain platform independency, which nevertheless may cause some problems. Their system includes e.g. voting, online feedback and quiz. (Mauve et al., 2001)

Darmstadt University of Technology has a project called "Open Client Lecture Interaction (OCLI)". Their approach is very similar to ours; students can send questions during the lecture, they can submit ratings and the teacher can do tests online. The approach used by OCLI is based on eXtended Markup Language (XML) and Extensible Stylesheet Language Transformation (XSLT), requiring only a device with a web browser or a Wireless Application Protocol (WAP) enabled cell phone from the client. Although the service is applicable to almost any device with a browser, the WAP approach may increase the complexity of the system. (Trompler et al., 2002)

4 Wireless technologies

To use the interactive learning environment students need devices through which they can use the implemented services. Increasing number of laptops and Personal Digital Assistants (PDA) gives a change to utilize the online learning in the near future. In our approach both Bluetooth and wireless local area network (WLAN) could be used as the communication methods for the online virtual environment. However only WLAN ideas have been implemented so far.

- Bluetooth is a wireless data transmission technology that offers a connection with two or more devices at a short distance. In the future the chip will most likely be included in many electrical devices, such as mobile phones, laptops and personal digital assistants. This makes it possible to use them as peer-to-peer devices on a short distance within a classroom. Group work during the lecture could be a good example of short-range communication.

- WLAN technology gives a change to connect into local area network without plugging into a wired LAN wall outlet. To operate WLAN network needs an access point though which data is routed to the Internet. The advantage of WLAN is a larger coverage area that makes it possible to connect the whole classroom of students into the same discussion.

5 Case: Increasing interaction through modern technology

We have built up a system that gives a possibility for the students to affect the learning situation. We provide modules that can be used to support different events during the presentation. Because of the modularity, new features will be rather easy to include into existing systems. The implemented modules are presented in the following list:

- Presentation slides available for everybody

  Nowadays most of presentations are given in electric form (ppt, pdf). Using the features of this software, teacher can easily share the presentation for students by uploading it into the server, so students can then have this slideshow for themselves before the lecture. In some cases the students may even change the content of the presentation.

- Student feedback and questions

  Students also have a possibility to give feedback and make questions. The original idea was that the screen projected into canvas divided into separate frames. In the largest frame is placed the most important thing, the presentation. All the questions will be printed next to the running slideshow, so that all the participants, as well as teacher, have a possibility to see them and to comment.

- Lecture evaluation

  Students can also evaluate the presentation by giving votes. Teacher sees the results from the selected interval as a real-time diagram on the canvas. Poor evaluation during a short period indicates a need to explain actual issues more precisely. In traditional learning environment some students may raise questions and demand more explanation but unfortunately this does not happen too often.

- Student evaluation

  Teachers do not necessarily know whether the students really have understood the issues in a way teacher wanted them to. Teacher now has a simple way to create an online quiz to watch over understanding. Compared to old-fashioned random tests given in paper the summary of results can now be seen immediately through a web-form. For teachers it gives a possibility to react, if the results demonstrate that the students have not understood the issues well enough. Quizzes may also perform as a part of final grade for the students.

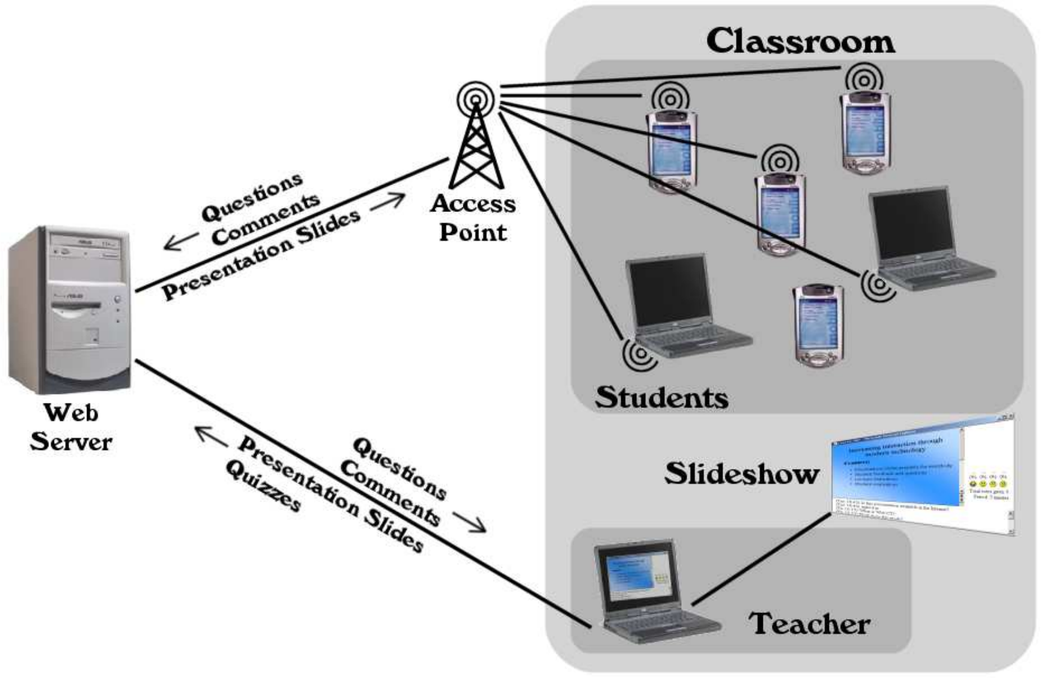


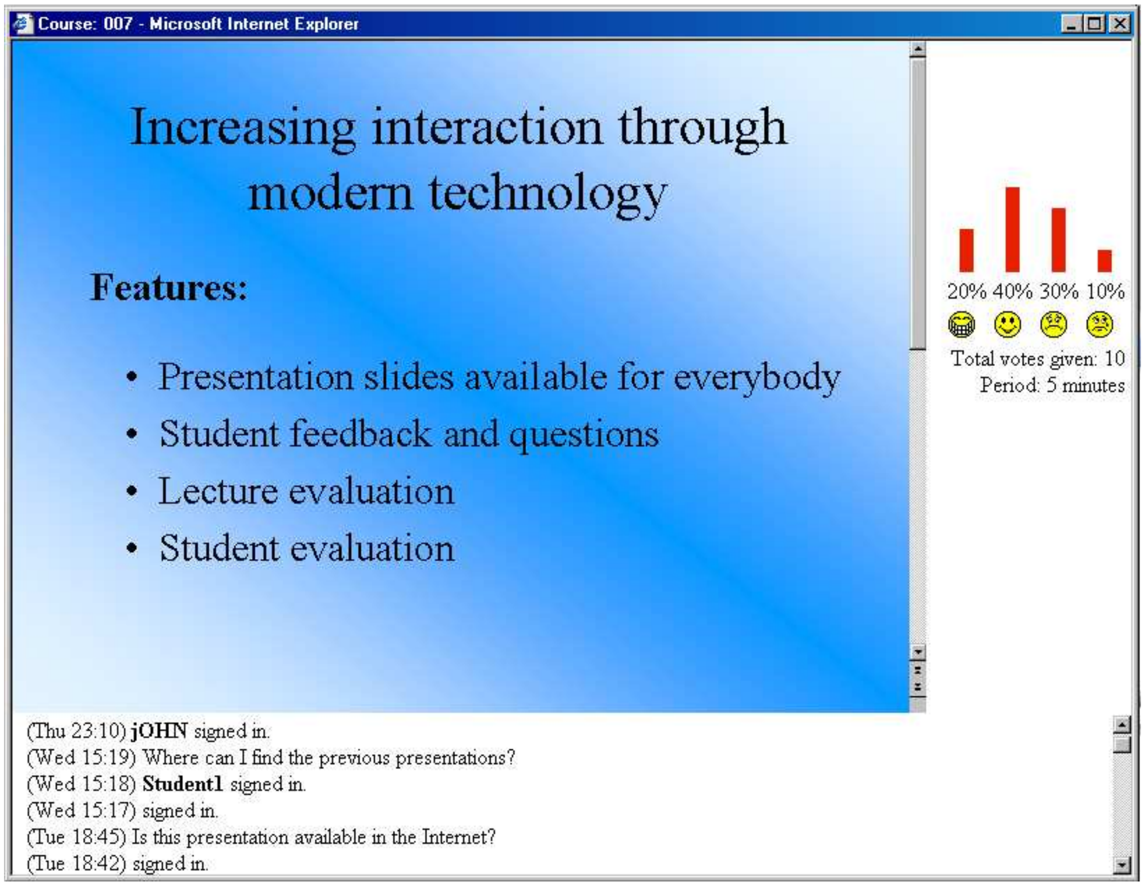
Figure 1: Principles of the functionality.

Figure 1 describes the relationships and functionality of the service. All the data between students and the teacher goes through a web server, where it is stored for later analyzation. Because normally there is no wired connection to network present for students, they connect mobile. Teacher instead normally has a wired connection that he can use to run the service.

The teacher's view shown in the canvas is represented in figure 2. In the top-left frame is shown the current slideshow. The right frame includes the results of students' evaluation of the presentation. In the bottom frame are shown the questions and feedback sent by students. By setting all the relevant information into one framed screen we decrease the need for additional overhead projectors.

The user interface of the service is kept as simple as possible. By doing this we aim that the service is platform independent. There is no need to install any programs on the client side, just a simple web browser is all we need.

So far the system has not been tested in education environment. Testing is considered to be done during autumn 2002 within a course of laboratory of datacommunications. On the lecture some PDAs with network access will be provided to students. Students can use them to participate the online teaching. Based on the experiences and feedback modifications are probably made. As a result it can be seen if this kind of service is useful from both teachers' and students' point of view.



Increasing interaction through modern technology

Features:

- Presentation slides available for everybody
- Student feedback and questions
- Lecture evaluation
- Student evaluation

Total votes given: 10
Period: 5 minutes

(Thu 23:10) **jOHN** signed in.
(Wed 15:19) Where can I find the previous presentations?
(Wed 15:18) **Student1** signed in.
(Wed 15:17) signed in.
(Tue 18:45) Is this presentation available in the Internet?
(Tue 18:42) signed in.

Figure 2: Screenshot from the presentation projected into the canvas.

6 Conclusions

This paper presents an idea where virtual environment is brought as a part of online teaching experience. This environment uses modern wireless communication technologies like Bluetooth and WLAN to increase interaction between the students and the teacher. WLAN technology has been added on an existing virtual environment and functions like lecture evaluation, student evaluation through quizzes, questions and presentation material are implemented. Through this environment the teacher will have better understanding (grading) of students and may improve (individualize) the lectures. Students may affect the content and presentation of lectures and finally get better understanding on the issue.

References

Dufresne, R. J., Gerace, W. J., Leonard, W. J., Mestre, J. P., Wenk, L., 1996. Classtalk: A Classroom Communication System for Active Learning. Journal of Computing in Higher Education 7 (2), 3–47.

Mauve, M., Scheele, N., Geyer, W., 2001. Enhancing Synchronous Distance Education with Pervasive Devices.

Trompler, C., Muhlhauser, M., Wegner, W., 2002. Open Client Lecture Interaction: An Approach to Wireless Learners-in-the-Loop. In: Proceedings of 4th International Conference on New Educational Environment.

University of North Carolina at Wilmington, P. E., Hypercube, 2002. Numina II Student Response System.
URL `http://aa.uncwil.edu/numina/srs/`

An Interactive Tool for Study Planning

T. Niklander, V. Raatikka, and A. Rytkönen
Department of Computer Science, PO.Box 26, FIN-00014 University of Helsinki

Anni.Rytkonen@cs.Helsinki.FI

1 Introduction

Planning the studies is one of the most important tasks a university student should do. At a large department, like the Department of Computer Science at the University of Helsinki, this is very demanding and complex, because there are so many alternatives to choose from. All course combinations are not acceptable for the Master's degree even if the count of credit units is enough. Without having feedback about the study plan the student can easily miss some obligatory courses that are necessary for the degree.

A personal study plan (in Finnish, Henkilökohtainen opintosuunnitelma, HOPS) has been defined to help the students when planning their studies. It has been in use for several years in multiple places. For example, the University of Joensuu has excellent web-pages (see OVI-hanke (2002)) that give students advice on making the study plan. Following this information the student is able to make a concise study plan, which gives good guidance for the studies. However, we want to go a step further. We want to provide the student with an interactive tool to guide in the study planning. A similar idea has been presented also by the student union at the Helsinki University of Technology (see TKY (2002)). The Helsinki University of Technology already has a set of tools for students to use (see Nybergh (2001)), but the prototype study-planning tool, called eHOPS, is not included in the tool set.

Our goal with the interactive study-planning tool is in the automatic verification of the plan. The tool does not replace a teacher in the planning process, but helps the student to avoid the obvious mistakes in the plan. The support for creation and maintenance of the study plan becomes even more important for students participating in distance learning (see Finnish Virtual University (2002)). The virtual university is aimed to provide a student with a large collection of alternatives creating more variation in the possible study plans for one student. The virtual courses may be offered by multiple universities and the course contents may also overlap. To overcome the problem of course overlaps, the student must have support when making her own study plan. Our tool is created to provide instantaneous support for students during the planning process.

Eight universities in Finland have formed the OODI-consortium (2002) for developing a large support system for learning and teaching. Currently it provides a study register, called OODI, for storing student information: enrolment and course credits. It already provides students with an excerpt of the study register. There is also a simple planning tool, which allows the students to list courses for terms and years. Our planning tool allows this too, but it also checks the plan against the course prerequisites and syllabus.

Most of the functionalities described in the following sections are to be developed in the future. At present, there is a prototype in test use.

2 Overview of the system

For planning one's studies, the student must gather information which is available in multiple sources often only as a printed copy: the curriculum for the starting academic year or term; the present and possibly the previous syllabus (if the student has studied more than a couple of years); the course preliminary requisites; and the excerpt from the study register. After finishing the study plan, the student registers for the courses with a separate system.

Everything mentioned above, and more, is combined in the study planning tool. The student will be able to plan her studies for the current and incoming terms. There is available a list of courses from which to pick the right courses. There is a possibility to include the list of passed courses in the system, and after doing that, the possibility to view the study situation as a graph (or a list, if wanted). Of course, the plans sometimes change. If so, updating the study plan is easy, because it is saved in the system.

If the student so requires, the tool can suggest courses that should be taken as soon as possible, so as to be able to take advanced courses in the same field later on. If the suggestions continue throughout the syllabus, the tool could make the whole study plan for the student. The suggestion can be based on the number of credit units per term that the student wishes to take. The study planning tool contains general models for studies. The student can use these models as a base for her study plan.

The study plan is also considered to be a preliminary registration for the courses. Additionally, the student can allow the system to register her to the courses and the exercise groups planned for the incoming term. Furthermore, at the end of the term, when the passed courses are added to the study register, the student may compare the plan made at the beginning of the term with the list of passed courses. The plan can also be used as a base for a termwise timetable, where it is possible to include time for homework as well as hobbies and working hours. The timetable can be stored and printed, as well.

The tool cannot, of course, make the decision for the student about the students' own interests. It is only able to support the student in making a study plan that is realistic to implement: the course order in the study plan is sensible and the amount of work during the studies is manageable. The tool is not designed nor meant to replace the face-to-face communication between student and teacher. Even when using the planning tool, the students still need to discuss their plans with their teachers. Discussing one's studies with a mentor, tutor, professor, or study counselor is important for the student to clarify her goals. The clarification of these goals in the form of a study plan is the current trend.

3 Inner functionality

In computer science the course contents are often based on each other. Participation in an advanced course usually requires that the student has passed the basic, preliminary courses. In addition to these obligatory prerequisites, there are recommendations and remarks about good knowledge of the contents of the preliminary courses. These dependences form a large directed graph (Fig. 1), which is not easy to navigate. The study plan must follow these dependences, and the planning tool can check them automatically. This is an enormous help to the student.

The first and second year basic courses are lectured every term. More advanced courses are lectured in either the Autumn or the Spring term. Some special courses may be lectured only every second year. These rules are called the principles of teaching.

The syllabus has a tendency of changing every year or two at the Department of Computer Science. This means that some students have passed courses during several syllabi. The prerequisites for the present courses are made according to the current syllabus. Some course groups in the different syllabi are equal to each other, though they may have different names. Therefore, when the prerequisites are checked, these equivalences have to be taken into account.

The list of courses given during a term are presented to the student sorted according to the students current situation. The courses which the student has already passed (or which equal the passed courses) are not shown. The immediate recommendable courses are listed first. After them, the rest of the courses that have their prerequisites filled are presented. If or when there are courses in which the student is not allowed to participate yet, because of lacking prerequisites, they are presented last. For a student heading for a specific subprogram the obligatory courses for that program are pointed out in the sorted list.

When the student continues her planning to future terms, the tool will, of course, take into account the previous term plans as passed and present the list of courses according to that. For future terms, for which the curriculum may not be available yet, the principles of teaching is used instead.

In order to manage all the details mentioned above, the tool needs at least

- the current and previous syllabi for a list of the obligatory courses on different levels and programs,
- the curriculum or teaching program for a list of courses to be given during the current term or academic year,
- the principles of teaching, for the terms for which there is no curriculum available yet,
- dependences between the courses as a list or a graph to show the possible study paths,
- the current situation of the student as, for example, a list of passed courses or an excerpt of the study register, and
- an earlier study plan, if any, to show the student the continuation in the plan and to avoid filling in the same information again.

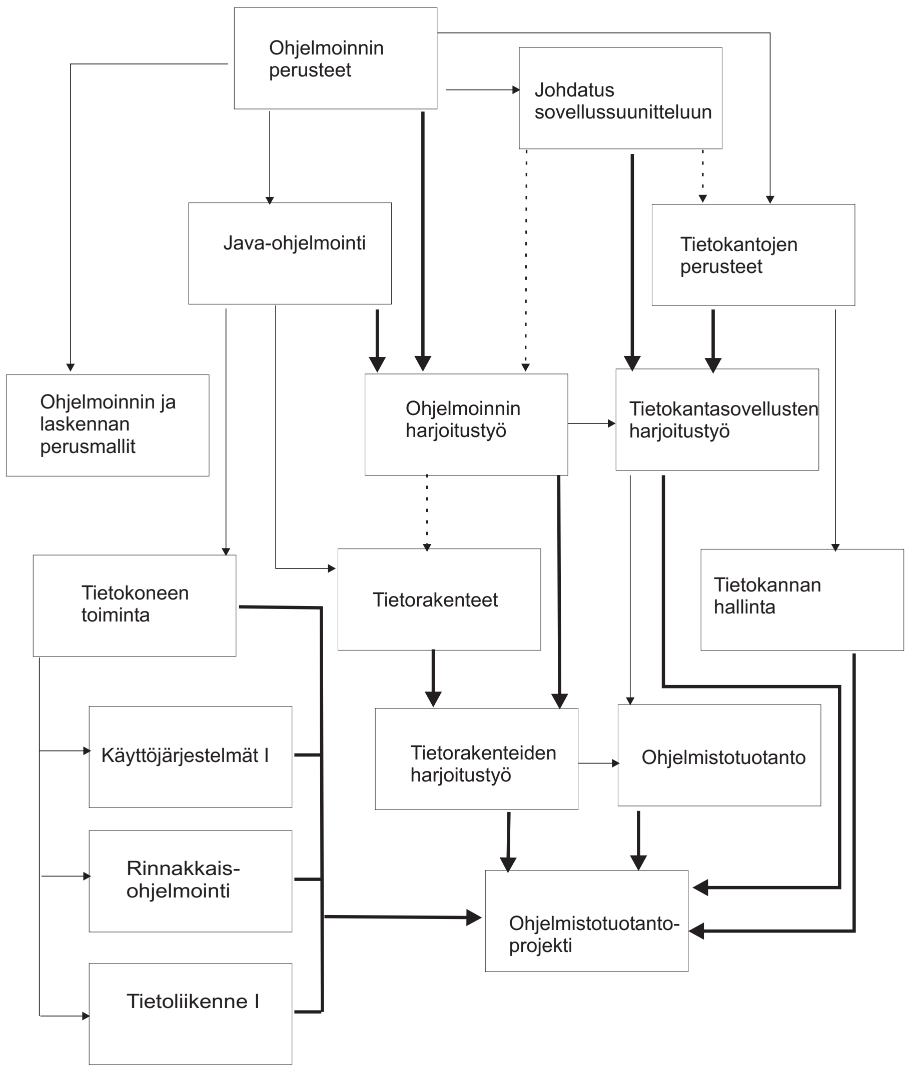


Figure 1: An example of a graph describing the course prerequisites. The bold lines represent obligatory prerequisites. The thin lines show, that the student should have good knowledge of the contents of the preliminary courses. The dashed lines are only recommendations.

The planning tool has separate sources for each piece of information. The earlier study plan is stored to a local database, where it can be fetched when needed. The current situation of the student's studies can be fetched from a central study register like OODI, or it can be asked from the student. In the case of OODI, the excerpt of the study register can also be used. The passed, but missing, courses must be included manually by the student herself.

The curriculum with the lecture times can be fetched from the department's teaching database. The current and earlier syllabi, the principles of teaching, the course dependences and the course equivalences must be provided by the administrative staff responsible for them. A simple and easily maintainable format (based on XML) is necessary for them. To reduce the work of the administrative staff, this information should be usable also when producing parts of the printed student guide about the curriculum and the syllabus.

As a result of the planning process, there are several outputs. The student is provided with the complete study plan for the coming term. She is able to save it in the system for later use and print it on paper, if needed. The study plan includes the lecture times for the planned courses. If there is any overlapping between course timetables, all the alternatives are shown. These times with optional hobby or working times can be used to form a weekly timetable, which is then saved separately from the study plan.

The student can see the situation of her studies as a graph of courses like in the Figure 1 with the passed courses marked with date and grade. After making the study plan, the planned courses are in the graph as well, with planned terms and periods.

At some point of the studies, the student has finished a study module by passing enough courses. To get a grade for the module, she must collect the courses that she wants to include into the module by selecting them from the list of passed courses. The course list can be saved in the system and printed for the professor to accept and sign. Changes are allowed, until the professor has accepted the module.

4 Interest groups

The tool is basically designed for students to plan their studies with. But besides the students, there are other interest groups, too (see Fig. 2). The other interest groups are all members of the staff.

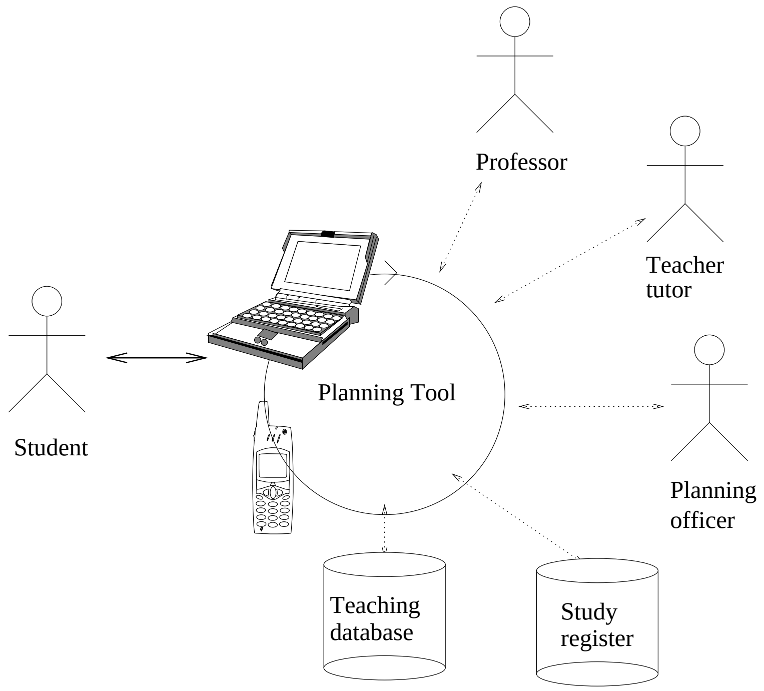


Figure 2: The study planning tool and the connections to the interest groups.

The course planning officer can use the summary of the study plans in estimating course sizes and thereby is able to prepare with a large enough lecture hall and a proper amount of exercise groups. He could also see the most popular course combinations and build the curriculum accordingly, with the minimum amount of overlapping between the combined courses.

When the student wants to have a grade for a finished study module, she could inform her professor of the designed course combination using the planning tool. The professor could then get familiar with the plan in advance or they could work on it together. When the course combination is ready, the professor can sign physically or electronically the course list after which changes are no longer allowed. The module is then registered to the study register.

A teacher tutor appointed to the student could, as part of the tutoring process, use the tool for checking that the student has made a study plan for the term or terms ahead. He could also comment on the plan in detail and in the right context instead of writing an e-mail. The tutor could get an overall impression of the student group he is tutoring, as there is an amount of different graphs visualising e.g. correlation of courses planned and courses passed termwise.

5 Current situation and future plans

The planning tool has been designed to be implemented in phases. At present, there is a first prototype of the system, which will be in test use in the Autumn 2002. The prototype was implemented as a student group project in the Spring 2002, and another student group is continuing the work in the Autumn 2002. The prototype allows making list-based plans, shows the current curriculum in whole and allows the input of an excerpt of the study register. During the Autumn term, the part that parses the course list according to the curriculum and the list of passed courses, should be included to the tool. Probably, there is going to be yet another student group in the Spring 2003 that will implement more functionality; and the tool is going to be in test use all the time among students that participate in teacher tutoring.

To increase the student's use of the planning tool, it should support the slogan 'Anywhere, anytime, anyhow'. From the student's perspective, the study plan should be available using any access mechanism on hand. The current web-based system is quite close to this kind of access, because it is not fixed to a specific computer or program. However, in the future when new equipment appears, the planning tool and the student's own study plan must be available also through those.

A planning tool which guides students through the information flow is an important step further from the current paper guides. Its proper application should help the students to make their own decisions within the limits allowed in syllabus and curriculum.

References

Finnish Virtual University, 2002. Suomen virtuaaliyliopisto. http://www.virtuaaliyliopisto.fi/, (17.9.2002).

Nybergh, M., 2001. Sähköiset työkalut opiskelun ja opetuksen tukena. Peda-forum - tiedotuslehti 8 (1), 18–19.

OODI-consortium, 2002. Opiskelun ja opetuksen tuen tietojärjestelmä. http://www.oodi.fi/, (17.9.2002).

OVI-hanke, 2002. Ohjausta virtuaalisesti. http://ovi.joensuu.fi/, (17.9.2002).

TKY, 2002. EHOPS in Brief. http://www.tky.hut.fi/ tky-kopo/tvt/, (17.9.2002).

PAPER SESSION 3

Problem Based Learning in Introductory Programming Does It Scale Up?

P. Kinnunen and L. Malmi
Helsinki University of Technology
Department of Computer Science and Engineering
P.O.Box 02015 HUT, Finland
{pakinnun,lma}@cs.hut.fi

1 Introduction

Problem Based Learning (PBL) is a learning method supporting the constructivistic learning theory (Gunstone, 2002). Students are given cases which deal with practical problems and phenomena related to course topics, and they come together in groups to discuss the problem and what they should learn about it. This method has been widely used in medical sciences and law and business schools (Huey, 2001). Some versions of it have also been applied in natural sciences (Williams, 2001).

In Helsinki University of Technology we adopted the PBL method for one small-scale, *i.e.* about 30 students yearly, introductory programming course in 1999. This course (Helsinki University of Technology, course T-106.219, 2002), hereafter called the Standard PBL course, S-PBL, has been given yearly since then with surprisingly good results (Malmi et al., 2002). Whereas in the ordinary courses of introductory programming (here called Standard courses) some 30-50 percent of students drop the course, the dropping rate has been practically zero in the S-PBL course. Moreover, students clearly aim at good learning results, instead of just passing the course. Finally it worth noting that the students on the S-PBL course, about half of them females, have no prior programming experience, whereas in the Standard course many students have some prior knowledge of programming.

The S-PBL method, unfortunately, does not scale up to large courses due to lack of tutoring and classroom resources. We have therefore developed a new lighter version which aims at preserving the important properties of the method without needing that much resources. During the academic year 2001-2002 two prototype versions were given. The results, discussed in this paper, are still somewhat contradictory.

We present the S-PBL method and its lighter version, called Light PBL, in Section 2. In Section 3 we present some statistics about the prototype courses. Other observations are discussed in Section 4. Section 5 concludes.

2 Problem Based Learning

The key idea in Problem Based Learning is that students discuss in groups practical problems and/or phenomena which are related with the topic to be learned. They analyse the case and discuss what they already know about it. Based on this they set up their own learning goals in order to understand the case better. Later on they come together to discuss and share what they have independently learned. The emphasis in the method is not solving the problem, which can be called Problem Oriented Learning. The emphasis is that the problem cases initiate the process of defining learning goals.

The basic difference between the PBL version and the standard version of the programming course is that the PBL version has very few or no lectures. Instead, the students use the 7-step method, explained below, in their weekly session to discuss the *PBL cases.* On both version they also submit programming exercises biweekly, a programming project at the end and take a final examination.

2.1 7-step method

The version of PBL that we have applied is called the 7-step method which we adopted from Schmidt (1983). Each case has 7 steps. 1) The students read the given stimulus, such as a written description of a problem or a report of observations (with analysis omitted). 2) They identify the key issue of the case and give a name for it. 3) A brainstorm is used to find concepts which students associate with the case. Typically students write their ideas to self-stick notes and attach them on a whiteboard or a paper on a wall. 4) An initial model of the information related to the case is built. Related collections of slips are recognized and their relations can be emphasized by drawing lines and arrows. Typically upper level concepts are identified in this phase. 5) The group defines their *learning goals*, *i.e.*, issues which they should learn to understand the model and the case more deeply.

The phases 1–5 are called the opening of the case (typically 0.5 – 1 hours). After that they enter Step 6 of independent study (one week), during which each student searches for information to learn about the goals. It is highly important that the learning goals are joint; everybody studies the same things. After this they come together in Step 7, to close the case (1–2 hours). Now they discuss what they have learned. The discussion should not be limited to repeating facts but explaining their meaning and importance. Typically in one weekly meeting the group closes the previous case and opens a new one. Thus the meeting takes 2-3 hours.

Students handle the cases in groups of about 6-8 students and a tutor. The tutor is a domain expert available for asking questions and, possibly guiding the process. It is important that the tutor should not act as a teacher. The guidance works best in the form of asking questions from the group, not giving answers, unless explicitly requested.

Another important issue is that students should not use only one source of information. If several books, internet pages, articles etc. are used, students find different points of view which considerably enriches the discussion in the closing session.

2.2 Light PBL

For two reasons, the previous PBL method does not scale up to mass courses with hundreds of students. First, the university has not enough classrooms of group work rooms for PBL groups. Second, we need too many tutors for the groups. Therefore a new method was launched such that could scale up, but which should retain the key properties of the original method.

The students are divided into groups of 6-8 students, as before. For each group a tutoring assistant is assigned, but the assistant is present only in the first two PBL cases so that the group learns the method. Thereafter the group meets weekly to discuss the cases on its own. Moreover, the members are responsible for finding a location for their meetings. In their meetings students work using the 7-step method. After each meeting the group writes down the learning goals set up in the opening and the problems identified in the closing session. Problems typically relate to topics which were jointly considered somewhat unclear for the group. This report is sent to the assistant by email. The assistant may comment on the learning goals by email to clarify them.

The assistant meets 2-4 groups weekly in a session of about an hour. In this session the problems that the groups had identified in their previous meetings, are discussed. In addition, the assistant can provide feedback about the exercises which were submitted on the previous week.

3 Results

3.1 Drop out rates

The key problem of the Standard course is the high drop out rate. Typically some 30-50 percent of enrolling students fail to pass the course. On the S-PBL course dropping out has not a been a problem and therefore the drop out rate on the Light PBL course is of central interest.

The Light PBL method was applied on two parallel basic programming courses, hereafter called L1 and Y1, in years 2001-2002. The course syllabuses are roughly the same, but L1 has slightly stricter requirements. The course populations differ so that in general students of L1 have a better background in marks [1] than Y1 students. Moreover, many L1 students take more programming courses than Y1 students. Almost all students of the courses study computer science as their minor; students of computer science curriculum have a programming course of their own.

The results are shown in Table 1. Light PBL students of L1 have statistically significantly better results (less drop out) than students of Standard L1 course. All these 34 students (21 males, 13 females) were volunteers. However, they had no prior programming experience, whereas a considerable minority of Standard L1 course students know some programming initially.

In the Y1 course in spring 2002, 55 (31 males, 24 females) students volunteered to the Light PBL version. 34 (20 men, 14 women) of them were randomly chosen and the rest 21 students were left to the control group which took the ordinary course. About 20 percent of the PBL students had some knowledge of programming initially. This time no difference between the Light PBL group and the Standard course students was found. The control group performance was inferior to both other groups. However, the differences were not statistically significant.

Table 1: Results of the L1/Y1 course and L1/Y1 light PBL versions. The Y1 control group had equal requirements as the standard Y1 group.

Student status	L1 ordinary		L1 L. PBL		Y1 ordinary		Y1 L. PBL		Y1 control	
	N	%	N	%	N	%	N	%	N	%
Submitted something	462	100.0	34	100.0	550	100.0	25	100.0	17	100.0
Passed exercises	303	65.6	30	88.2	408	74.2	16	64.0	12	70.6
Passed project work	256	55.4	28	82.4	285	51.8	11	44.0	9	52.9
Passed examination	283	61.3	27	79.4	285	51.8	12	48.0	6	35.3
Passed whole course	272	58.9	26	76.5	231	42.0	10	40.0	5	29.4

Since students have several options to take the exam during the academic year, neither L1 nor Y1 drop out results are final at the time of writing this. However, we emphasize that L1 and Y1 results are not comparable in any case due to the variation of the population and the requirements.

3.2 Analysis

Based on questionnairies the main reason of dropping the course is the lack of time. Programming courses were more laborious than students had anticipated when planning their timetable.

The second reason for many dropouts was that Light PBL turned out to be a different studying method than students had thought, based on the information they had. Either they thought it as a plain group work method or a lighter method to study programming. For some students Light PBL required too much self-direction. These misconceptions were mostly observed in the Y1 / Light PBL course, which unfortunately had some information delivery problems in the course beginning. All students did not receive enough information about the working method, and how much work is needed.

[1] Based on university entrance examination and student matriculation examination.

4 Discussion

In this section we discuss observations and student feedback of the method. Feedback material has been collected by web questionnairies, personal interviews and on-site observations during group sessions.

About the PBL-method The PBL method, especially the Light PBL version emphasizes students' own responsibility of their studies. This has its pros and cons. Students generally appreciate that they have many possibilities to start processing the cases. Moreover, setting up the learning goals themselves motivates their studies and promotes the real need for searching for new information.

Students were eligible to create a more active attitude towards their studies and some students changed the conception of themselves as a learner. They felt that they were able to learn also hard subjects although it might take some time and effort.

Tutor's role The lack of a tutor in PBL sessions had its pros and cons, as well. Students were not afraid to ask also simple questions without being embarrassed. They felt that the threshold to ask other students was reduced. In addition, the students used the same "language" in the discussion whereas the tutor could use jargon which was not fully understood. On the other hand, the absence of the tutor in the PBL session allowed the discussion to stray to irrelevant topics too easily. In some groups, students missed the tutor because they missed somebody to look after if they study hard enough.

Most students found the weekly tutor meetings very helpful. Due to the small group students dared to ask simple questions, something they would not dare to do on a mass lecture. In a tutor meeting students found helpful that they could ensure that their learning goals were rational. In addition, the tutor was able to explain some important aspects, which students had ignored. Personal advices and feedback were the offerings of the tutor meetings.

Group work Group work was generally considered positive. Actually, to be able to work in a group was in fact the major reason why students wanted to take part in the Light PBL course. Several positive arguments could be identified from the feedback. First, in a group one can talk and ask questions and get little tips about the exercises and the subject with other students. The group thus acts as a social net for getting help. It also relieves the anxiety caused by the course. Second, in order to be able to explain and define one's own opinion one really has to internalize the topic. Third, in a group many different sides of the subject are raised, and one gets a more versatile picture from the topic. Fourth, it was a release for many students to find out that the other members of the group knew as little about programming as themselves. In that way, they did not feel like more ignorant than others. Finally, through PBL it is possible to learn group-working skills, which students figured they would need in their future working life.

4.1 Students requirements

As a whole, the PBL method (both S-PBL and Light PBL) requires more involvement and self-discipline than traditional courses.

First, students have to be mature and their motivation is important. Even though the group helps keeping up the motivation, the student self has to have stamina and good will to study through the whole course. Somebody's low motivation in a group has a negative effect on the group spirit and the general motivation.

Second, students have to have enough time to participate group meetings, to study for their goals and to do weekly exercises/essays. It takes some self-discipline to stick to this study plan. A general problem distracting this is that group members' timetables may be incompatible.

Third, students have to have somewhat advanced studying skills. They have to be able to search for relevant information from many different sources. Students have to be able to find out what is essential in the case and then to set learning goals. It is also important that at the group meetings students work effectively and do not stray from the case.

Composition of the group should be homogeneous. Students' previous knowledge and skills should be about the same. This is important for two reasons. First, when everybody knows as much or little about the subject, students are not afraid to ask simple questions. Second, if somebody knows more, others would be likely to ask him/her every time instead of finding out the answer together. There would be a greater danger for those who know less to become passive.

5 Conclusion

We have presented a new version of the PBL method such that could be scalable to mass courses. Two pilot versions of the method on an introductory programming courses have shown that the method has many beneficial features on students' learning. However, the drop out rates of the pilot courses have been contradictory.

A new version of the Light PBL method will be carried out in fall 2002. We will use more problem-oriented stimuli. In addition, we emphasize that the groups should prepare deliverables when closing the case. These include design plans for the given problem and concept maps prepared on the learned topics. We hope that these additions will aid groups working more together, which should discourage drop out.

References

Gunstone, R., 2002. Constructivism and learning research in science education. In: Phillips, D. (Ed.), Constructivism in Education. Opinions and Second Opinions on Controversial Issues. No. Part 1 in Ninety-ninth Yearbook of the National Society for the Study of Education. The University of Chicago Press, Chicago, USA.

Helsinki University of Technology, course T-106.219, 2002. Home page for course T-106.219, Information Networks Studio 1. http://www.cs.hut.fi/Opinnot/T-106.219/s2001/ (in finnish).

Huey, D., 2001. Problem-based learning: The potential utility of problem-based learning in the education of clinical psychologists and others. Education for Health 14 (1), 11–19.

Malmi, L., Nuutila, E., Törmä, S., 2002. Ongelmalähtöinen oppiminen ohjelmoinnin perusopetuksessa (problem based learning in teaching basic programming). In: OPE^2 - Opetuksen kehittäminen TKK:lla, kokemuksia opetuksen kehittämishankkeista (reports from teaching experiments carried out in HUT). TKK:n opetuksen ja oppimisen tuen julkaisuja. Helsinki University of Technology.

Schmidt, H., 1983. Problem-based learning: Rational and description. Medical Education 17, 11–16.

Williams, B., 2001. Introductory physics, a problem based model. In: Dutch, B. J., Groh, S. E., Allen, D. E. (Eds.), The Power of Problem Based Learning. A Practical "How To" For Teaching Undergraduate Courses in Any Discipline. Styles, Sterling, Virginia, USA.

Teaching Computer Science by Playing

W. Hämäläinen
Department of Computer Science, P.O. Box 111, FIN-80101 Joensuu, Finland

whamalai@cs.joensuu.fi

1 Introduction

In this paper I introduce new experimental methods in teaching theory of computer science in the university. Althoug I have experiences only in university teaching the same methods can be applied as well in other levels of studying. Role of programming exercises, simulation and other computer-aided methods has always been emphasized in the short history of computer science teaching, but physical methods has been nearly ignored. Still it has been proved that use of all senses in demonstrations and learning by doing give the most comprehensive learning experience(Carnevale, 2000; Gibbs, 1988). In this paper I shall call all these methods in general as experimental learning/teaching.

This kind of teaching connotes so called Perinetic School[1], which emphasizes use of all kind of new creative and revolutional methods in teaching/learning. As the Perinetic School is still quite unknown I shall first describe its main ideas.

The name of the Perinetic School comes from Greek (per=through, nein=to swim) and means "learning by swimming". In the Perinetic School learning process happens by swimming like in the ancient Peripatetic School by walking. The Perinetic Scool was founded in the beginning of 21st century in Mekrijärvi, Eastern part of Finland. It is told that originally the students and teachers really went for swimming in shallow water to have educational discussions. Since that the Perinetic teachers may still take their students to the water, but the Perinetic School has a larger metaphorical meaning. The knowledge is referred as "Mare Nostre", "our ocean", mother of all life, creativity and imagination – something still unknown, into which the learners have to jump bravely and learn to deal with it. Like in water the beginner may be confused and afraid of sinking into huge amount of information unless fighting against it by all means, but after learning to relax one notices that the water itself bears and there is no hurry.

The main tasks of the teacher in the Perinetic School are to courage the students to jump into something new and find the existing curious and playing child in oneself. Music, playing, games and arts can be used as tools also in teaching very theoretical and abstract things e.g. datastructures and algorithms in computer science. Creative teachers have always used this kind of methods in their teaching, but there are no public reports and evaluation of them.

I shall now introduce a few experimental methods I have used in teaching computer science in the university. After that I shall try to evaluate their role in teaching and report students' feedback. Finally I shall give some ideas, how to apply the ideas into new areas in computer science.

2 Some games about computer science

2.1 Drawing classdiagrams from music

Learning the conceptual modelling of entities as objects of classes, which have their own attributes, methods and relations is quite hard for some students. That kind of object-oriented thinking should first be practised with common things surrounding us in real world. The students are not motivated to read and mecanically analyze several application descriptions, nor do they learn to understand the underlying modelling paradigm. Instead the teacher

[1]The school was founded in An International Summer School on educational technology, University of Joensuu, Aug 2002.

may give the task in the form of music: the students listen to music and try to draw a class diagram of the story in lyrics. The first step can be to draw an object diagram, which is then generalized to the class diagram.

For example when I was lecturing ”Introduction to Application Design” (Hämäläinen, 2000a) I let my students listen to a song ”Viimeinen kylähullu” (”The last village nut”) by Juice Leskinen. The story tells about a benevolent but simple man, who helps everybody in the village but is considered as a useless nut by the society. In the song the state tries to put such people into mental hospital. Students experienced this very surprising but facinating beginning of the lecture and I couraged them to do the same at home, when they are practising for the exam.

The lyrics and resulting class diagram are shown in the appendix.

2.2 Simulating object collaboration diagrams

Object collaborating diagrams show, how and in which order the objects cooperate to carry out some task. Every object has a set of methods, which it can execute based on its data contents or return values of service calls to other objects.

This kind of collaboration can be played as a game, in which individul students or groups of students play objects. Each object is given one or more methods – usually very simple algorithms – and some data contents. To execute the service at least some of the methods require service calls to other objects. Each student or group is given a name card so that the objects can be easily recognized. Service calls are simulated by running to the right object, asking the service and waiting for response. If there are enough students, the whole group can be divided in parts, which competite agains each other.

For example I have used this game after the previous one in ”Introduction to Application Design”. First we considered, what kind of methods we should add to the class diagram to implement the task ”Find all village nuts and put them into mental hospital”. Then we completed the collaboration diagram with order of execution. After that we shared the roles and began the game, in which two competiting states were hunting their village nuts. After some confusion the game succeeded quite well and afterwards even the sleeping students were awaken and motivated to know, what was the idea of the game.

PhD Harri Laine in the University of Helsinki has also tried similar play, in which the students simulated the bottle recycling machine (Laine, 1996).

2.3 Simulating abstract datastructures

Also the abstract datastructures and their algorithms can be simulated by playing. The students may construct linked lists, binary trees, graphs or heaps. Each student is playing a node in structure and one student may be a pointer. The students are given some data, for example numbers or letters, which they should order by the algorithm or search the biggest one. Another interesting game is playing a balanced binary tree, in which the inner nodes have arithmetic operations and the leaf nodes have integer numbers, and the main task is to evaluate the expression.

I have tried this game only in the Christmas party of Department of Computer science, in which the participants varied from undergraduate 2nd year students to professors. The game (on in fact a competition between several groups) succeeded well and it was believed to be suitable also in real teaching.

2.4 Simulating finite automata

In this game the group of students is given a transition diagram of a finite automaton. They may look at it for a while and then they have to construct the same automaton themselves. Each student corresponds a state of the automaton and knows only, how the automaton behaves in one's state i.e. is s/he the final state or to whom one should transfer the control. The string is given to the initial state as a stack of paper cards, one symbol in each. Every state reads one symbol, takes off the card and gives the rest to the following state.

The final state reports, if the string was accepted or not. This game I have only tried in the Christmas party, but this time the idea of finite automata was forgotten or new for more participants. Especially those participants gave good feedback about the game: they found it really enlightening demonstration.

2.5 Some related experiments

I have also tried some other experiments, which don't require concrete playing, but still attract imagination an use of all senses. For example the Finnish salt liquorice sweeties are usually black and diamond-shaped like the composition symbol in class diagrams. Sweeties can be used as counters when completing the class diagram with composition relations. Afterwards the salt liquorices can be eaten by meditating the meaning of the composition: "A salt liquorice addict is as existentially dependent on salt liquorice as it were a composition relationship."

Writing poems brings also variety to the scientific texsts. One Finnish teacher in physics has given his students a task to write poems about physical phenomena and laws. I developed the idea further and gave an exercise to program a poet generator, which should produce poems by given grammar (Hämäläinen, 2000b). The idea was to use the technique of implementing a finite automaton, but in this case the automaton didn't read strings but generated strings by selecting a random transition from the current state. Some students owned themselves very ambititiously for this task and the resulting poems about the Pumping lemma, automata and grammars were also usable in the later teaching (Hämäläinen, 2001).

3 Evaluation

I don't have any statistics to evaluate the use of playing and other experimental methods in computer science teaching. The course official feedback about innovativity of teaching covers other things, too. But I have got verbal feedback about playing immediately and afterwards in feedback forms.

The students seem to be eager to try everything new, if the teacher encourages them from the beginning. That's why the first year students are easiest group for such experiments: they haven't got used to the traditional university teaching yet. Still all the students can be lured to try new methods, if the atmosphere in the course is open and the teacher is oneself innovating. For example in my course "Introduction to Application Design" the students varied from first to third year students and the oldest ones were middle-aged. Still nearly all (except one, who didn't give any reason) took a part the object collaboration game.

The feedback of students (and also exercise teachers, who hear only the stories the students tell) has been good. The traditional teaching has been considered quite boring and there might really be a danger of falling asleep. Concrete playing gives wanted change and stimulate the students to study also the theory more carefully.

The games also work as problems, which wake the iterest to understand the secrets of diagrams, structures and algorithms. Even if the game itself proves to be a confusion, the students are afterwards motivated to learn, what was the idea, what they should have done, and the play may be tried again. The students may also be asked to invent new games in the context of the course, which serves as learning by teaching.

4 Final remarks

The atmosphere in university teaching at least in computer science has been rigid for new teaching methods. Yet all abilities of human being should be utilized in learning. Playing is part of human being and it should be couraged in all learning and investigating. Playing has been shown to be inspirating both for students and teachers. The games and plays concretize abstract things, courage in understanding the underlying theory and offer wanted change in monotonic teacher-centered lectures.

The previous examples are free to be utilized and developed further. The same kind of plays can be applied to several areas in computer science. It is hard to give any generic recipe for generating new plays, but the following ideas seem promising:

- Message passing/routing in the network
- Graph algorithms, like finding the shortest path or Hamiltonian cycle in the network or solving the Travelling Salesman Problem. This is especially suitable for simulating parallel algorithms, in which the group of students travels the graph and labels the nodes.
- Resurse sharing protocols, like dining philosophers, readers and writers, time-stamp method.
- The idea of recursion in algorithms. The algorithm itself should be interesting like Hanoi towers.
- Turing machines.
- Finding the pair, who has the equivalent expression. The students can be given e.g. relation algebra, row calculus and column calculus expressions (or equivalent SQL–queries) and they have to find the corresponding expressions.
- Sorting algorithms.
- Pigeon-hole principle and its applications.
- Listening music and constructing a relational model of the song or answering the given SQL–query on it (the song should be selected carefully so that the lyrics really describe meaningfull entities and relations).
- Simulating SQL–queries. Each student is given a data record and they have to answer the query consisting `group by` experssion (e.g. counting the number of members in the group and calculating the average/min/max value of selected data field.)
- Coding and decoding. This game is easy to implement in the class room: one of the students invents a message and codes it. The coded message is transfered to another student, who has to decode it. For example professor Pasi Fränti has simulated Huffman coding with this game. Also suitable for encrypting methods.

5 Acknowledgements

I thank professor Timo Alanko, professor Pasi Fränti and FM Reijo Siven for their play ideas.

References

Carnevale, D., 2000. Scholar says: 'Learning by Doing' Is the Key to Quality Instruction. Cronicle of Higher Education (May, 30).
URL `http://chronicle.com/free/2000/05/2000053001u.htm`

Gibbs, G., 1988. Learning by Doing. A Guide to Teaching and Learning Methods. Further Education Unit, London.
URL `http://www.chelt.ac.uk/gdn/gibbs/`

Hämäläinen, W., 2000a. Johdatus sovellussuunnitteluun/Introduction to Application Design (course homepage).
URL `http://www.cs.helsinki.fi/u/whamalai/jss/jss.html`

Hämäläinen, W., 2000b. Ohjelmoinnin ja laskennan perusmallit/Models for Programming and Computing: exercise 3.
URL `http://www.cs.helsinki.fi/u/whamalai/olpe/eharj3.ps`

Hämäläinen, W., 2001. Ohjelmoinnin ja laskennan perusmallit/Models for Programming and Computing: Collection of Poems generated by Automata (in Finnish).
URL `http://www.cs.helsinki.fi/u/whamalai/olpe/runoja.html`

Laine, H., 1996. Systeemityön menetelmt/Information Systems Methodologies: exercise 5 (in Finnish).
URL `http://www.cs.helsinki.fi/u/laine/syme/syme_h5.txt`

6 Appendix

VIIMEINEN KYLÄHULLU
Hänet auto tuo uuden kodin luo,
taivas vettä valuttaa.
Kapuloiduin suin, ilmein masennetuin,
häntä pitää taluttaa.
Väkivalloin viedään noihin
ihmiskunnan arkistoihin
kylähullu viimeinen.
Hän oli tärkeä mies, hän arvonsa ties,
mies täynnä tarmoa.
Aina ketä vaan riensi auttamaan
eikä kerjännyt armoa.
Aina kaikkensa hän antoi,
lumet loi ja postit kantoi
kylähullu viimeinen.
Nyt on valtio kylään tullu
ja on siellä kuin kotonaan
ja viimeinen kylähullu
pois hoidetaan.
Kylän kahvilaan illoin istumaan
hän saapui kaskuineen
ja yhtenään, hyvää hyvyyttään,
toiset kiusas laskuineen.
Ehkä oli hän liian viisas,
se ihmisille piisas.
Hän on
kylähullu viimeinen.
Mutta kylään uus saapui arvokkuus
viran sosiaalisen.
Otti kohteekseen hullun vanhenneen,
siitä sekosi kaali sen.
Kun maailma tasapäistyy
niin persoonat sivuun väistyy.
Hän on
kylähullu viimeinen.
On valtio kylään tullu...
On valtio kylään tullu...
Joskus paikka tää tulvi elämää,
nyt se kaatui paperisotiin.
Se autioituu, kuolee puistossa puu,
väki viedään hoitokotiin.
Missä syy on moiseen vainoon?
Näin päästään tasapainoon,
Pois viedään
kylähullu viimeinen.
Kylähullu viimeinen...
Kylähullu viimeinen...

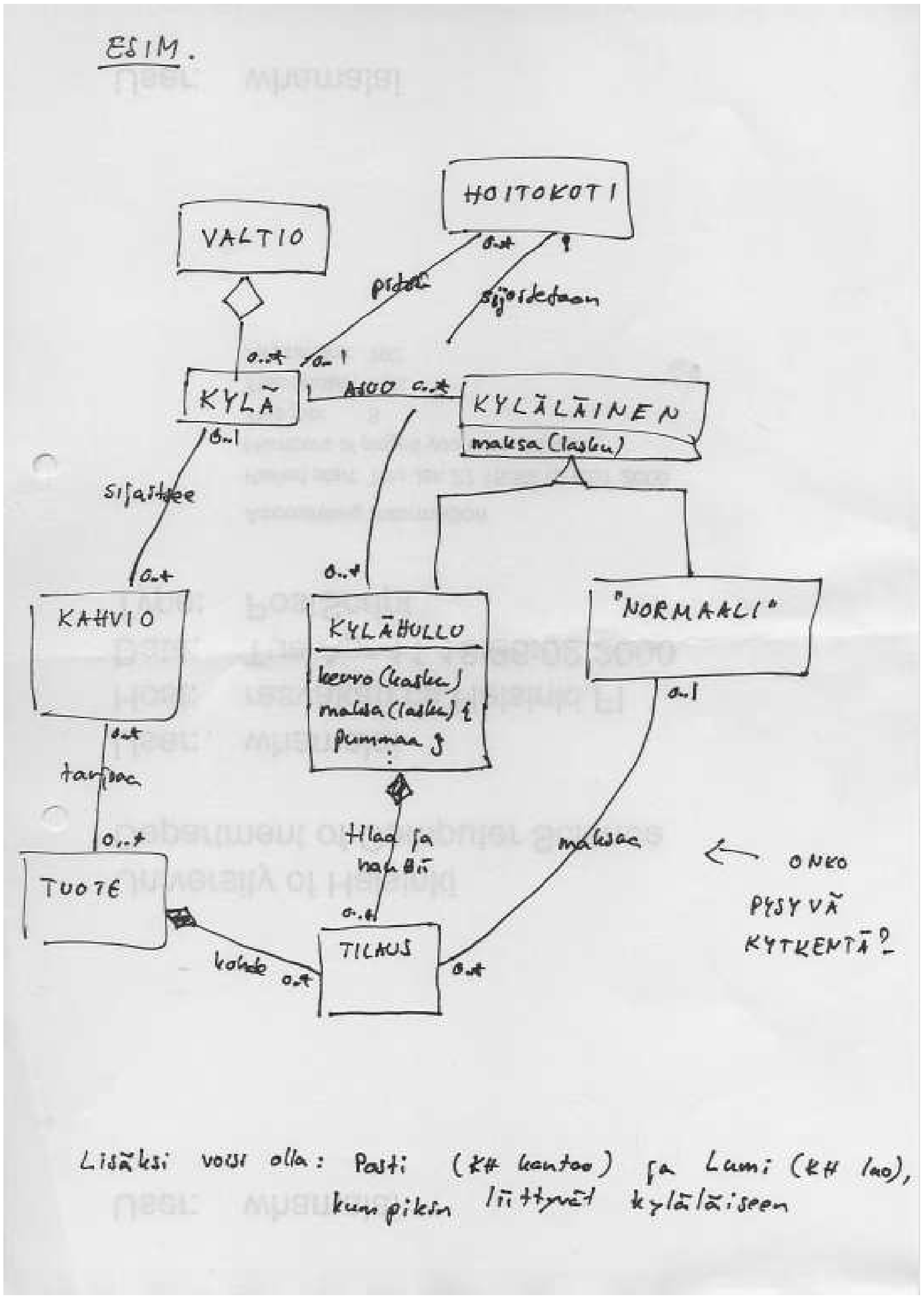


Figure 1: The lyrics and the corresponding class diagram.

PAPER SESSION 4

Effortless Creation of Algorithm Visualization

V. Karavirta, A. Korhonen, J. Nikander, and P. Tenhunen
Department of Computer Science and Engineering
Helsinki University of Technology
PO Box 5400, 02015 HUT, Finland

{vkaravir,archie,jtn,ptenhune}@cs.hut.fi

1 Introduction

The idea of using visualization technology to enhance the understanding of abstract concepts like data structures and algorithms, has become widely accepted. One of the main obstacles for fully taking advantage of algorithm visualization seems to be the time and effort required to design, integrate and maintain the visualizations. The process of creating software visualizations (SV) is thought to be too laborious to be worthwhile (Hundhausen et al., 2002). Thus, "a future challenge is to create tools and methodologies which will result in the use of SVs by the majority of computer science educators" (Domingue, 2002).

Several attempts have been made to introduce a system that levels out the burden of creating new visualizations (Haajanen et al., 1997; Korhonen and Malmi, 2002; LaFollette et al., 2000; Naharro-Berrocal et al., 2002). However, none of these systems has gained wide recognition. Therefore we have surveyed several systems in order to identify why the creation of software visualization is such a laborious process.

The users of visualization systems can be divided into two classes. First, there are *producers*, who use the systems to create visualizations. Second, there are *consumers* who benefit from the visualizations. By using, for example, Jeliot (Haajanen et al., 1997), it is possible for the consumer to be present and view the visualization while the producer constructs it, but it is not possible to save the visualization for later use. However, in systems like Animal (Rößling, 2000) and JAWAA (Pierson and Rodger, 1998), the producer first creates the visualization and it is used later on. There are also systems like Matrix (Korhonen and Malmi, 2002), where both approaches are possible.

If the visualization is recorded for later use it should be in some widely–used format. Moreover, the usefulness of a visualization is limited, if consumers need a special viewing program. It is impossible to use such visualizations as parts of Web-Based Learning Environments like JHAVÉ (Naps et al., 2000) or integrate them into hypertextbooks as argued by Ross and Grinder (2002). Thus, we only cover systems that are capable of delivering visualization in some widely used format such as Java applet or Scalable Vector Graphics.

Furthermore, there must be some minimum level of interaction between the consumer and the visualization (Rößling and Naps, 2002). Therefore systems that do not offer any control over the visualization have been left out of this study.

The paper is organized as follows. Section 2 defines the term effortless and section 3 describes the taxonomy used to characterize the example systems (Animal, JAWAA, Jeliot and Matrix). The evaluation is given in section 4. Finally, section 5 makes conclusions based on the observations and gives some future perspective.

2 Effortless algorithm visualization

In this section we define what we mean by *effortless creation of algorithm visualization*, *i.e.*, how well does the system support creation of algorithm visualization? This is a highly subjective measure including many factors. We see two categories characterizing the effortlessness of a given SV system: *WYSIWYG* (What You See Is What You Get) and *On-the-fly use*. Both of them look at the question from the producer's point of view.

Most people prefer WYSIWYG systems to systems based on textual input. Thus, our first category divides systems into two groups, those who support WYSIWYG and those who do not. Implementing the program to be visualized is not considered in this context. However, if the code must be manually annotated (JAWAA) without a WYSIWYG editor, the system is considered to be non-WYSIWYG.

One characteristic that has a great influence on the effort required is the ability to use the system on-the-fly. Let us consider a use case where a lecturer wants to demonstrate the behavior of a binary search tree (BST). With predefined animations it is very hard to adapt to all possible student questions. With animations created on the spot, the lecturer can easily answer "What if" type of questions by animating the various BST operations with different input data.

The degree of on-the-fly use can be measured by estimating the effort needed to prepare a new case to be visualized. Thus, we distinguish the systems by ranking the degree of preparation as follows: 1) no prior preparation, 2) requires programming of the algorithm, 3) requires programming and annotation and 4) not supported.

3 Taxonomy

In this section we introduce the criteria used to categorize algorithm visualization systems. Our criteria are derived from the taxonomy of Price et al. (1993). We have selected a subset of the taxonomy (generality and presentation style) that we feel relevant to the matter at hand based on the selected software visualization systems. Since the taxonomy is a bit old, we have extended or modified the meaning of some criteria.

Based on our presumptions of criteria and their relations to the effort, we have made three hypothesis that we will introduce in the following.

Category *generality* measures the area of use of the system. There are two subcategories of generality in the taxonomy we are interested in: *language* and *applications*. The language criterion tells what programming language the visualization objects must be specified in, if any. The applications criterion, or especially its subcategory *speciality*, specifies what a particular system is good at visualizing or, in other words, what the system is designed for.

Our hypothesis for the relation between generality and effort is the following. "*The more general a system is the more effort is needed to create a visualization with it.*"

The *presentation style* category describes the appearance of the visualization. We have selected two interesting subcategories: *graphical vocabulary* and *animation*. We have extended the original meaning of the graphical vocabulary with the idea of available concepts. It measures the graphical object's awareness of what it is representing.

Our hypothesis regarding graphical vocabulary is: "*The more primitive graphical vocabulary a system has the more effort is needed to create a visualization with it.*"

Animation describes the type of animation provided by a visualization system. Animation has three levels: none, step–wise and smooth. None means that the system is able to create static images. A step–wise animation is a set of still images. In smooth animation transitions between steps of the animation are smooth.

Our last hypothesis regarding animations is: "*It takes more effort to create smooth animations than to create step–wise animations.*"

4 Evaluation

In this section we present the outcomes after applying the criteria described in section 2 and the selected taxonomy represented in section 3.

4.1 Effortless systems

In the following we represent the relative ranking of the example systems. All these systems have aspects that support effortless use in creation of algorithm visualizations.

Table 1: Generality.

System	Rank	Language	Speciality
Animal	IIIII	n/a	any
JAWAA	IIII	any	any
Jeliot	III	EJava	code and algorithm animation
Matrix	II	Java	algorithm animation

Table 2: Presentation Style.

System	Graphical vocabulary	Animation
Animal	geometric shapes, list element	smooth, step–wise
JAWAA	geometric shapes, tree, graph, array, stack, queue	step–wise, (smooth)
Jeliot	primitive data types, array, stack, queue	smooth
Matrix	arrays, lists, trees, graphs	step-wise, smooth

1. Matrix supports both WYSIWYG editing and on-the-fly use with no prior preparation or at most by programming the algorithm only.
2. Jeliot supports both WYSIWYG editing and on-the-fly use with programming.
3. JAWAA, no WYSIWYG editor and on-the-fly use with programming and annotation.
4. Animal supports WYSIWYG editing, but not on-the-fly use.

We have ranked on-the-fly use to be a more valuable feature from the perspective of effortless creation. This is because it is more time-consuming to produce each animation instance of an algorithm separately with a WYSIWYG editor (Animal) compared to the hand-coding of the algorithm and producing all the instances by running the algorithm with different input data. However, the major division here is whether 1) a system supports both, WYSIWYG editor and on-the-fly use or 2) only one or none of these. The first two systems (Matrix and Jeliot) belong to the first category and the latter two (JAWAA, Animal) to the other.

4.2 Generality

Generality is measured with two criteria, language and speciality. We have considered a system's speciality to be more significant indicator of generality than the language. It should be noted, however, that the taxonomy fails to properly categorize systems, such as Animal and Matrix, that allow direct manipulation of the visualization. In both of these the user does not have to have *any specific* language in mind while creating the visualization. However, this has no effect on the ranking that follows.

The evaluation is summarized in Table 1. The most discriminating characteristic is Animal's and JAWAA's capability of visualizing (almost) anything. The other systems are only good at visualizing predefined data types. However, since Animal is not restricted to *any* language while in JAWAA *some* language must be used, we consider Animal to be more general than JAWAA. Jeliot can visualize and animate almost arbitrary Java code and is therefore more general than Matrix, in which code animation is not fully supported. Therefore, in terms of generality the systems can be ranked as follows: Animal, JAWAA, Jeliot, and Matrix.

4.3 Presentation style

Presentation style, represented in Table 2, is related to the set of artifacts that visualizer can manipulate. It is divided into graphical vocabulary and animation.

The graphical vocabulary determines the level of the conceptual abstractions the visualizer can use. Of the included systems Animal has the most simple concepts. The concepts

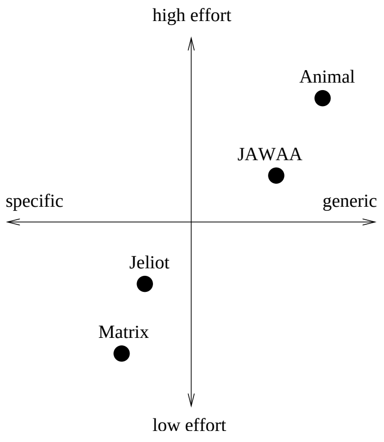


Figure 1: *Generality* vs. *effort.*

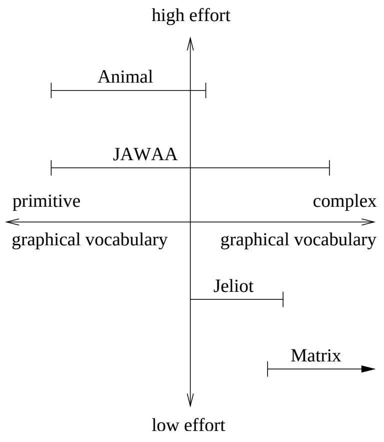


Figure 2: *Graphical vocabulary* vs. *effort.*

available in Jeliot and JAWAA are more complex, while those available in Matrix are even more complex.

We feel that none of the available styles of animation: none, step–wise or smooth – is better than the others, but a system that supports several styles is better than a system that supports only one. Using Animal or Matrix it is possible to create snapshots of an animation, as well as create step–wise or smooth animations. Using JAWAA it is possible to create either step–wise or smooth animations, though it takes more effort to create smooth animation. Jeliot supports only smooth animation.

5 Conclusions

In this paper we have studied four algorithm visualization systems. We have also introduced a taxonomy and presented three hypotheses how this taxonomy relates to the effort needed to create visualizations.

5.1 Generality and effort

In general, systems can be divided to four quadrants as shown in Fig. 1. Animal and JAWAA are *generic, high-effort* systems, whereas Jeliot and Matrix are *specific, low-effort* systems. None of the systems identify as *specific, high-effort* or as *generic, low-effort.*

In our first hypothesis we argued that *the more general a system is the more effort is needed to create a visualization with it.* The evidence gathered supports this. However, it would be interesting to see systems that qualify to the category of generic, low-effort.

5.2 Presentation style and effort

Our second hypothesis was that *the more primitive graphical vocabulary a system has the more effort is needed to create a visualization with it.* The graphical vocabulary axis in Fig. 2 is divided into two parts – primitive and complex. In primitive graphical vocabulary we have only simple drawing components, whereas complex graphical vocabulary includes abstractions of data structures. The more complex the graphical vocabulary is, the more abstract concepts are supported. The segment of a line implies the range of covered graphical concepts. Moreover, in Matrix one has the option to expand its graphical vocabulary. For example, it is possible to combine the reference implementations for lists and arrays in order to produce an adjacency list without programming a new concept.

The data in the fourfold table (Fig. 2) agrees with our original hypothesis. However, none of the systems extends to the quadrant of primitive graphical vocabulary and low-effort. The question is whether primitive graphical vocabulary always leads to a high effort system?

In our last hypothesis we stated that *it takes more effort to create smooth animations than to create step-wise animations.* This is not true. Our counter example is Matrix – it takes precisely the same effort to create smooth or step-wise animations with it.

5.3 Final remark

As the first two hypothesis imply there are some open research problems that are relevant to the topic at hand. Especially we are interested to see whether it is possible to develop the generic AV systems further toward effortless creation of algorithm visualization. On the other hand, it is also an open question if the current low effort systems could be developed further to be more generic.

References

Domingue, J., 2002. Software Visualization and Education. In: Software Visualization: International Seminar. Springer, Dagstuhl, Germany, pp. 205–212.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., Vanninen, P., 1997. Animation of user algorithms on the Web. In: Proceedings of Symposium on Visual Languages. IEEE, Isle of Capri, ITALY, pp. 360–367.

Hundhausen, C., Douglas, S., Stasko, J., 2002. A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing 13 (3), 259–290.

Korhonen, A., Malmi, L., May 2002. Matrix – Concept Animation and Algorithm Simulation System. In: Proceedings of the Working Conference on Advanced Visual Interfaces. ACM, Trento, Italy, pp. 109–114.

LaFollette, P., Korsh, J., Sangwan, R., 2000. A Visual Interface for Effortless Animation of C/C++ Programs. Journal of Visual Languages and Computing 11 (1), 27–48.

Naharro-Berrocal, F., Pareja-Flores, C., Urquiza-Fuentes, J., 2002. Redesigning the Animation Capabilities of a Functional Programming Environment under an Educational Framework. In: Second Program Visualization Workshop. ACM, HornstrupCentret, Denmark.

Naps, T. L., Eagan, J. R., Norton, L. L., March 2000. JHAVÉ: An Environment to Actively Engage Students in Web-based Algorithm Visualizations. In: Proceedings of the SIGCSE Session. ACM, Austin, Texas, pp. 109–113.

Pierson, W., Rodger, S., 1998. Web-based Animation of Data Structures Using JAWAA. In: Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education. ACM, Atlanta, GA, USA, pp. 267–271.

Price, B., Baecker, R., Small, I., 1993. A Principled Taxonomy of Software Visualization. Journal of Visual Languages and Computing 4 (3), 211–266.

Ross, R. J., Grinder, M. T., 2002. Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resource for the Web. In: Software Visualization: International Seminar. Springer, Dagstuhl, Germany, pp. 269–283.

Rößling, G., 2000. The ANIMAL Algorithm Animation Tool. In: Proceedings of the 5th Annual SIGCSE/SGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00. ACM, Helsinki, Finland, pp. 37–40.

Rößling, G., Naps, T. L., 2002. A Testbed for Pedagogical Requirements in Algorithm Visualizations. In: Proceedings of the 7th Annual SIGCSE/SGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02. ACM, Aarhus, Denmark, pp. 96–100.

A Picture is Worth a Thousand Words? - UML as a Multimedia Learning Tool.

C. Schulte
University Paderborn Fuerstenallee 11 33102 Paderborn Germany

carsten@uni-paderborn.de

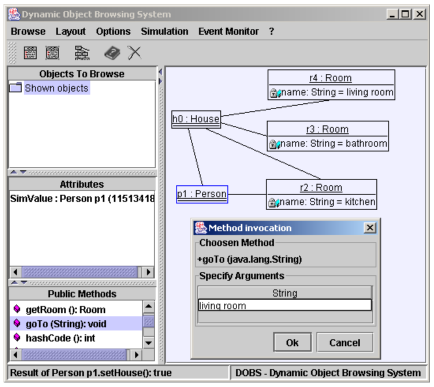
1 Introduction: The Difficulty in Learning Object-Oriented Technologies

Since the introduction of object-oriented technologies (OOT) seems crucial, educators being involved in teaching OOT have searched for suitable teaching concepts, especially for the beginners' level. The discussion on suitable pedagogies is often concentrated on the schedule of topics to be taught (see e.g. ACM (2001), Becker (2001)). The difficulty finding a suitable schedule is due to the fact that object-oriented concepts are particularly interdependent. Educators face a situation in which several different didactical approaches co-exist - but only few criteria for selection. However such criteria can be derived from learning theory. Ben-Ari (2001)developed criteria in order to assess those teaching concepts for introducing OOT which claim to rely on a constructivist learning theory. According to constructivist learning, a learner constructs new knowledge based on his previous knowledge. An important aspect to consider is, how the previously knowledge is being taken into account: "Constructivism suggests that programming exercises should be delayed until class discussion has enabled the construction of a good [mental] model of the computer [..] Premature attempts to write programs lead to bricolage [try and error programming, C.S.] and delay the development of viable [mental, C.S.] models" (Ben-Ari (2001), p.14). UML in conjunction with a case-tool provides means to show beginners object oriented examples - models and implementations - and thus enable the development of a suitable model of a computer before attempting to write programs. But these possibilities will be explored more detailed in section 2 of the paper.
In section 3 the elaboration of the mental model will be discussed. Referring to OOT Hadjerrouit distinguishes three types of knowledge: Object-oriented concepts, object-oriented language and problem specific knowledge. From "a constructivist point of view, these knowledge types need to be closely related to each other in order to be useful for problem-solving" (Hadjerrouit (1999), pp.172). Problem specific knowledge refers to the software development process: It means to know how, when and why to use a language tool or concept. Here UML diagrams can help to make visible relations between elements. The use of a case-tool allowing the implementation of the programme in UML helps to strengthen the relation between modelling and programming and thereby also strengthens the relation between language details and object oriented concepts. Both of the following sections are mainly focused on finding and describing teaching methods, in section 4 the question is discussed, how to exploit the visualisations of a subset of UML in order to maximize learning.

2 The Introductory Phase

The learning sequence to be assesed here (Schulte and Niere (2002)) is based on the concept of cognitive apprenticeship (Collins et al. (1989)). In the first lessons where the students are introduced to the tools and first get to know object-oriented concepts, programming is delayed: The students at first investigate a problem domain, e.g. a small board game. Afterwards the object-oriented point of view is explained to them as a means to describe a system as a set of individual objects working together. They are asked to describe the problem domain from this perspective. For example they can explain a dice: It is an object with six numbers; it can be thrown by a player and always shows one of the six numbers as result. In the next step the students are given an object-oriented model of the problem domain, described by CRC-Cards (Beck and Cunningham (1989)). They explore the model by performing the object game

(Bergin (2000)) in which some of the students act as objects. The teacher ensures that the actors only make use of the possibilities listed on the particular CRC-card - it is quite likely that a student acts according to his perception of the problem domain, and not according to the CRC-card. The aim of this is to put across that an object is limited to the actions described on its corresponding CRC-card and to explain how an object-oriented programme actually works: by the interaction of independent objects



SimValue : Person p1 (11513418

name: String = living room

name: String = bathroom

name: String = kitchen

getRoom (): Room

goTo (String): void

hashCode (): int

+goTo (java.lang.String)

living room

Result of Person p1.setHouse(): true

DOBS - Dynamic Object Browsing System

Figure 1: The running programme in the debugger Dobs.

After that the students explore the java-implementation of the model with the aid of a graphical debugger (Dobs, see fig. 1) which can show the actual state of the java virtual machine (VM). The debugger allows method invocation and displays the object structure of the VM as an UML-like object diagram: objects, associations, (public) methods and (public) attributes with their current value. Consequently, the students realise that an object-oriented programme behaves quite similar to the object-game they have played before.

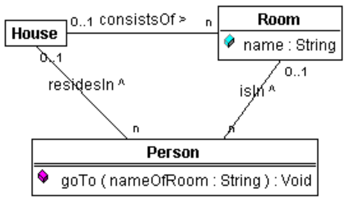


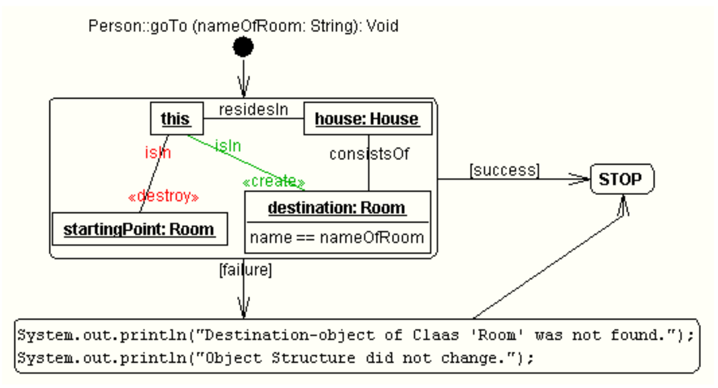
Figure 2: The Class-Diagram.

Then the students are given the first presentation of an object-oriented programme: The UML-class-diagram of the 'debugged' programme (2). After the examination of the class model the students learn how the methods are implemented. Therefore an UML-like diagram type called 'story diagram' is used (see fig. 3). It shows control flow and object manipulation as a collaboration-diagram. To sum it up, the introductory phase introduces the technical terminology, tools, and some basic concepts of object-oriented modelling.
Is this first phase of the teaching-concept suitable to develop an appropriate mental model of OOT? Well, the students' previous knowledge is applied by using examples familiar to them, such as board games which are presented as different models, showing a clear trace and the relations between the problem domain and the object-oriented model running in Dobs. The first task at the computer is to use an object-oriented model with Dobs (see fig. 1) then to have a look at the class-diagram (fig. 2) and thereafter at a diagram of an implemented method (fig. 3). Students do not see source code. So the learning sequence focuses on providing a kind of visual understanding of an object-oriented programme. This outline fosters a basic understanding and presents an intuitive way from a problem domain to

an object-oriented model. The intended mental model promotes an object-oriented program to be understood as a set of connected, yet independent objects, which can only act according to their (class) description. The computer as visually represented by Dobs (see fig. 1) acts as a store for objects, thereby providing an object structure. Methods, described as 'story diagrams' (fig. 3) are a means to manipulate such a structure of connected objects (for more details see Schulte and Niere (2002)).

3 The Elaboration of the mental model

Later on, after the introduction and practice of several concepts like loops, method implementation, gui's and event handling, the students work in pairs or small groups. The students almost work without the guidance of the teacher and develop a programme with a graphical user interface and event handling. At this point, the teacher plays the role of an expert, answers questions and organises classroom meetings. The aim of this phase is to foster the integration of the different knowledge types.



```
System.out.println("Destination-object of Claas 'Room' was not found.");
System.out.println("Object Structure did not change.");
```

Figure 3: The Implementation of a method.

In a case study that was conducted at two local schools, we noticed an interesting difference between the two courses: Overall, course A made slower progress, but in the post-test students showed a better performance. We assume this result is due to the fact that in course A the teacher spent more time in letting the students explain object-oriented models and their own small programmes: Frequently students had to present their programme in Dobs and its class-diagram by a video projector and to explain it verbally to the whole course. The others and/or the teacher then asked questions or gave suggestions for improvements.
This teaching method harmonizes very well with cognitive apprenticeship, as it fosters reflection and articulation helping to integrate knowledge. In course B this teaching method wasn't used as often and when, it was focused mainly towards the use of the tool: When a presented model was working in Dobs it wasn't examined or questioned afterwards.

4 Learning with Multimedia

As our observations indicate, the efficiency of the teaching concept isn't based on the tools functionality alone, but on its use in the teaching and learning process (see Kozma (1994)) also. In search for explanations and improvement of the teaching concept we have to ask, how tool support and teaching methods fit together in order to elaborate suitable mental models. Therefore, we examine the elaboration of mental models in more detail - which of course is a little bit speculative, because we conducted no psychological study with in-depth-investigation of the students' mental models. But we can draw some conclusions from a cognitive theory about multimedia learning, which focuses on the elaboration of mental models. The theory can applied here, because it fits into the general frame of constructivist learning theory used

in the course concept. Thereby we draw attention to the interdependency between visual and verbal presentation, and their cognitive processing and integration into a mental model. In short: The tool supports visualization, the teaching method adds verbalization.
The multimedia learning theory from Mayer is based on "dual coding theory, cognitive load theory, and constructivist learning theory" (Moreno and Mayer (2000)). Learning with multimedia is analysed on three different levels: The applied presentations, the sensory inputs (mainly visual and auditory), and the internal processing in the learner's working memory. Mayer conducted several empirical studies that support the main issues of the theory. The theory claims the following: "a) working memory includes independent auditory and visual working memories[..], b) each working memory store has a limited capacity [..], c) humans have separate systems for representing verbal and non-verbal information[..], d) meaningful learning occurs when a learner selects relevant information in each store, organizes the information in each store into a coherent representation, and makes connections between corresponding representations in each store" (Moreno and Mayer (2000)). Schnotz (Schnotz (2001)) adds that every presentation triggers both visual and verbal processing. He claims a deep interdependency between both of the mental representations so that every sensory input is being dually coded. For every topic to learn a learner develops a verbal and a pictorial mental model - however a (visual) presentation can favour visual processing, whereas a verbal presentation can favour the development of a verbal mental model. So understanding supposed to be deeper, if the models can be integrated, including the prior knowledge. Thus to foster active learning and deeper understanding, multimedia presentations should 1) avoid 'split attention', 2) present verbal information auditory rather than as on-screen-text, 3) synchronize verbal and visual materials or present them simultaneously, and 4) avoid unnecessary information.

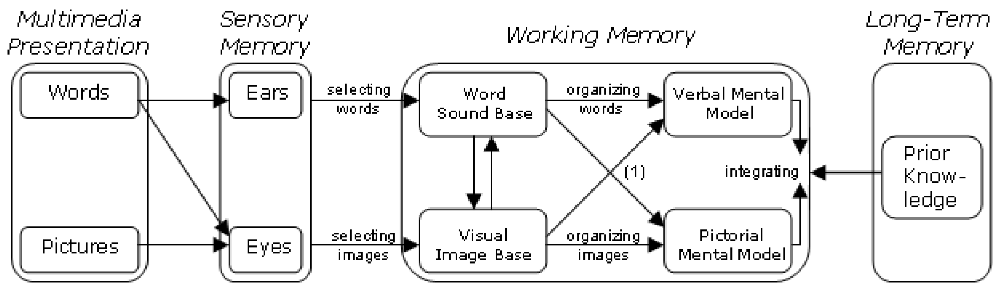


Figure 4: Cognitive theory of multimedia learning, graphics from Moreno and Mayer (2000), additions (1) by Schnotz (2001).

We assume that in the two courses the visual presentation of the UML-like diagrams (class diagrams, story diagrams and object diagrams) has indeed fostered a visual mental model which according to Schnotz (2001) is mentally represented analogue to the structures and/or functions of the original. Such a mental representation helps to draw conclusions by analogy. In other words: students have learnt that an object-oriented programme consists of a connected structure of objects as shown by Dobs (fig. 1), a method operates on the structure (fig. 3, the rectangle). They gained an overview about the object oriented idea and about some of its important concepts. Deeper understanding relies on the integrating of the knowledge types mentioned by Hadjerrouit (1999): Programming language, object oriented concepts, and problem solving skills. The use of UML as a programming language (enabled by the software development tool Fujaba) links programming aspects with design issues, thereby providing opportunities to integrate language details with object oriented concepts. The integration of problem solving skills is supported by the way the students have to develop UML-diagrams and to reflect on their solutions.
If the UML-diagrams in general favour visual processing, then verbal explanations are needed to develop a verbal mental model that according to Schnotz holds more detailed information

and abstracted knowledge. And those verbal explanations aren't merely necessary to develop understanding for the use of the tools but to teach problem solving skills and help to integrate them with knowledge about the object oriented language and object oriented concepts. This would explain why in course A students learnt more: They were instructed to articulate and reflect on the visual diagrams. So they gained a deeper understanding of the visual programming language and the development of such diagrams. This way, the main principles from cognitive apprenticeship (see Collins et al. (1989)) are supported also.

This leads to the conclusion that teamwork and classroom presentations with an emphasis on appropriate verbal explanations is an effective way of using UML (and CRC-cards) as a multimedia learning tool:
(1) The use of a visual programming language (like UML in Fujaba) in a constructivist learning environment is more effective, if it is used as a means to describe object structures in order to visualize the general idea of OOT.
(2) On the other hand, the teacher should take care that the visually programmed models are verbally explained by and to the students in order to foster the integration of the verbal and the pictorial model. So one might say, a picture needs a thousand words.

References

ACM, 2001. Computing Curricula 2001. http://www.computer.org/education/cc2001/-final/index.htm, volume II-Computer Science. Approved Final Draft of the Computer Science Volume, December 15, 2001.

Beck, K., Cunningham, W., 1989. A laboratory for teaching object oriented thinking. In: Conference proceedings on Object-oriented programming systems, languages and applications. ACM Press, pp. 1–6.

Becker, B. W., 2001. Pedagogies for CS1: A Survey of Java Textbooks. Manuscript: http://www.math.uwaterloo.ca/ bwbecker/papers/javaPedagogies.pdf.

Ben-Ari, M., 2001. Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching 20 (1), 45–73.

Bergin, J., 2000. The Object Game. http://csis.pace.edu/ bergin/patterns/objectgame.html.

Collins, A., Brown, J., Newman, S., 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In: Resnick, L. (Ed.), Knowing, learning, and instruction: Essays in honor of Robert Glaser. Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 453–494.

Hadjerrouit, S., 1999. A constructivist approach to object-oriented design and programming. In: Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education. ACM Press, pp. 171–174.

Kozma, R. B., 1994. Will media influence learning? Reframing the debate. Education technology research and development 42 (2), 7–19.

Moreno, R., Mayer, R. E., 2000. A Learner-Centered Approach to Multimedia Explanations. Deriving Instructional Design Principles from Cognitive Theory. http://imej.wfu.edu/articles/2000/2/05/index.asp. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning 2 (2).

Schnotz, W., 2001. Wissenserwerb mit Multimedia. Unterrichtswissenschaft 19 (4), 293–318.

Schulte, C., Niere, J., 2002. Thinking in Object Structures: Teaching Modelling in Secondary Schools. http://prog.vub.ac.be/ecoop2002/ws03/acc_papers/Joerg_Niere.pdf. Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP, Malaga, Spain. 2002.

Using Empirical Modelling to Simulate Robotics in Kids' Club

P. J. Eronen
Department of Computer Science, University of Joensuu, P.O Box 111, FIN-80101 Joensuu, Finland
peronen@cs.joensuu.fi

J. Järvelä
Research and Development Centre for Information Technology in Education, University of Joensuu, P.O Box 111, FIN-80101 Joensuu, Finland
jjarvela@cc.joensuu.fi

C. Roe
Department of Computer Science, University of Warwick, United Kingdom, CV4 7AL
croe@dcs.warwick.ac.uk

M. Virnes
Department of Computer Science, University of Joensuu, P.O Box 111, FIN-80101 Joensuu, Finland
mvirnes@cs.joensuu.fi

1 Introduction

The Kids' Club, run by the Department of Computer Science at the University of Joensuu in Eastern Finland, gives children the chance to construct robots using the LEGO Mindstorms kits (The LEGO group, 2002) and program them using computers. The children in the club are all between 10 and 14 years old and have been participating in the group for up to one year. They have developed a competent understanding of programming and basic robotics, which was demonstrated by the success they achieved in winning their league in the RoboCup Junior robot football world cup (Executive Committee for RoboCup-2002, 2002).

The task of creating robots and programming them has proved to be both mentally and technologically challenging. Often the children encountered problems that were not related to their actions. The technical environment, comprising computer hardware, software and robotics kits has been sometimes unreliable, which can disrupt the children's learning process. The staff of Kids' Club have recognised the need for a computer-based environment, which allows rapid and flexible generation of simulated robots, their behaviour and problem-oriented tasks for them to solve. A simulation environment would help the children to construct the mental models of robots and their tasks without being distracted by real-world problems such as non-working connections between the robot and the computer being used to program it. In this paper we describe the outline for a system that simulates the robotics environment.

This paper is in three main sections. The first describes the activities of the Kids' Club and the goals set for it. The second section outlines Empirical Modelling (EM) and how it is suited to the creation of environments for investigative exploration. The third section explains the pedagogical issues surrounding the creation of a computer-based environment to investigate robot behaviours and how EM is an appropriate way to address these issues. A preliminary outline for the proposed system is also described in the third section.

2 The Kids' Club

Kids' Club is an active research laboratory, where children work together with research workers and university students. For children, Kids' Club is a technology club where they can work in an open-minded environment with interesting tools and have lots of room to follow their individual desires. Children not only learn skills in the field of information and communication technology (ICT), but also have an essential role in contributing to the research community. (Eronen et al., 2002a)

The children learn various skills by using different tools of educational technology. The technical environments include visualisation and concretising tools (e.g a visual programming environment (IPPE) (Jormanainen et al., 2002), control technologies (Empirica Control) (Lavonen et al., 2001) and programmable bricks (LEGO Mindstorms) (The LEGO group, 2002). Children solve small scientific problems working collaboratively in a playful, explorative way. For example, through working with IPPE and robots children learn to understand basic control structures in procedural programming when creating the behaviour for a robot, or mathematical formulas and laws of physics when measuring the speed of a robot.

From a research perspective the aim of Kids' Club is to develop novel ways to learn and teach ICT. We would also like to stimulate children's interest in ICT and encourage them to become both active and confident users of information technology, and doers in information technology society.

The pedagogical background (Eronen et al., 2002b) for Kids' Club is based on problem-based learning, creative problem solving and group processes. In the Kids' Club children are designers, who work in pairs in projects without a tight schedule, but in a goal-oriented way within a certain subject. We have not defined a strict curriculum nor do we have any tests. Problem-based learning creates an iterative process from the definition of the problem to presenting the project, through ideas of possible solutions, building, programming and testing the LEGO robot. In this process concretising tools make constructions of mind concrete giving a wider perspective on learning (Papert, 1980). In problem situations children ask help from mentors whose role is not to teach, but to instruct and help children. The mentors are under-graduate and graduate students of computer science and their backgrounds are in computer science and education. To strengthen the learning process children reflect on their learning with the help of a Virtual Reflecting tool, VirRe, at the end of each Kids' Club meeting. Web-form that was used earlier for reflecting was not pleasant to children. Neither it was productive way to collect data for researchers and tutors. Thus, we developed VirRe that records children's answers and web-camera picture into one video file.

According to mentors and children's experiences after the first year, it seems that Kids' Club is a promising environment for developing educational technologies. We can get immediate and constructive feedback from children, which is one of the aims of Kids' Club. Secondly, children's excitement to act in the environment and learn by solving problems shows that visual and concretising tools offer an attractive environment for learning abstract skills like programming.

3 Empirical Modelling

Empirical Modelling (EM) is an approach to computer-based modelling that has been developed at the University of Warwick over many years (The Empirical Modelling Research Group, 2002a). The construction of models using these techniques aims to support learning in situations where the process is difficult to prescribe and approaches to learning need to be flexible to support different styles. The Empiricist Perspective on Learning described in (Roe and Beynon, 2002) underpins the learning process that is typical of building models using EM. Most traditional approaches to program construction emphasise the importance of behaviours. The presumption is that the most fundamental aspect of a system is its overall state-transition model. We believe the concept of 'state-as-experienced' to be the most important consideration when constructing computer-based models. The state is a snapshot of external observed values at a particular moment in time. There is no restriction made on possible future states that the system may go into.

This situation is typical of a spreadsheet. Users are free to make any changes that they desire by redefining the value of an observed quantity or by changing a dependency within the system. This experiential interaction is a key characteristic of the EM approach and gives a flexible environment within which exploratory model construction can take place concurrently with improving domain understanding.

This allows us to engage with the primitive modelling that is required to gain initial understanding of a domain before we can articulate about possible sensible behaviours. This perspective on learning and model construction is more fully explained in (Roe and Beynon, 2002).

We use definitive scripts to represent state-as-experienced. A script contains observables and dependencies. An observable is an element of the state that refers to something we can directly apprehend or indirectly determine in the real-world context we are trying to model. A dependency reflects the way in which these observables are indivisibly linked together. By making redefinitions in a definitive script we can explore transitions between states that may be important.

Our modelling environment called TkEden can be used as a 'black-box' to explore software created by another modeller, as a 'partial model' through the construction of small auxiliary scripts to carry out new tasks or refine the current situation, or as an environment for a modeller to construct models of his own (Roe, 1999). Models created by another modeller can be presented in a layered format. Each layer introduces new concepts to increases the complexity and range of options available to learners. We call this technique 'cognitive layering'.

4 Simulating Robotics

In a simulation the learner is in an authentic environment in which she interacts with elements incorporated in it. Simulations are computerized models, which are designed to teach how an imaginary or real system works. (Roblyer and Edwards, 2000) In the Robotics Simulation Environment (RSE) learners can program and operate virtual robots that resemble real-world Lego robots. The simulation is situational, which means that learner is put in a hypothetical problem situation and she is asked to react (Roblyer and Edwards, 2000). There are two main goals in using the RSE.

The first goal is to help learners build mental models about robotics that can be used when creating real-world robots. When interacting with real-world robots technical problems can hinder understanding of the fundamental principles. In a simulation the learner can concentrate on the essential features. She does not need to think about the physical design of the robot or resolve technical issues. The learning is made easier in the simulation by presenting a predefined learning path, starting with making predictions about ready-made robots behaviour working through to building their own robots and programming them.

A second goal of the RSE is to exceed real world limitations. Lego robots have only a few different sensors (e.g light sensors, touch sensors), but in a simulation environment we can include new sensors, for example to detect robot interactions. Different scenarios can be generated, for example there can be a traffic scenario, where a robot must be programmed to survive in heavy traffic with other cars, pedestrians and crossroads.

In the RSE, learners are in control of their own learning. The learner uses the simulation, and a mentor provides scaffolding when it is required. Learning is a result of the learners' own actions, when she interacts with the environment. The motivation for using the environment stems from the children's tendency to research their own surroundings and make experiments to get feedback. (von Wright, 1993). The process of learning is best described as mental model building.

Mental models are cognitive artefacts, constructions of the mind, which represent, organise and restructure domain knowledge in a way that observable or imaginary world becomes conceivable (Seel, 2001). They are naturally evolving, and develop when people interact with their environment, other people and artefacts. Mental models produce predictive and explanatory power in understanding different interactions (Norman, 1983). According to constructivism, they are adaptable when faced with a conflict situation and grow stronger through interaction in the environment. In the RSE the learner is in a situation where she has to make predictions about the functions of the robots. In the beginning learners' mental models are inaccurate and incomplete. Learners can refine their mental models with feedback from the simulation. The mental model, which is not viable, must be improved in order to succeed in reaching the wanted goals (Norman, 1983) (Seel, 2001). Learners mental models of robotics evolve when interacting with the RSE.

The process of mental model building in RSE is summarised in Figure 1. The process begins with a problem. The learner can have a vague hypothesis for solving the problem, or she can observe the environment in order to form a hypothesis. The learner begins the testing cycle, where she can test her hypothesis and refine it with feedback from the environment. Eventually the testing cycle produces a solution to the problem. The learners' mental model will be refined. The new mental model raises new questions and produces new problems. The refined mental model also increases the learner's ability to predict the behaviour of operations in the environment. The mental model acquired through interaction with RSE can be transferred to the process of concrete robot building.

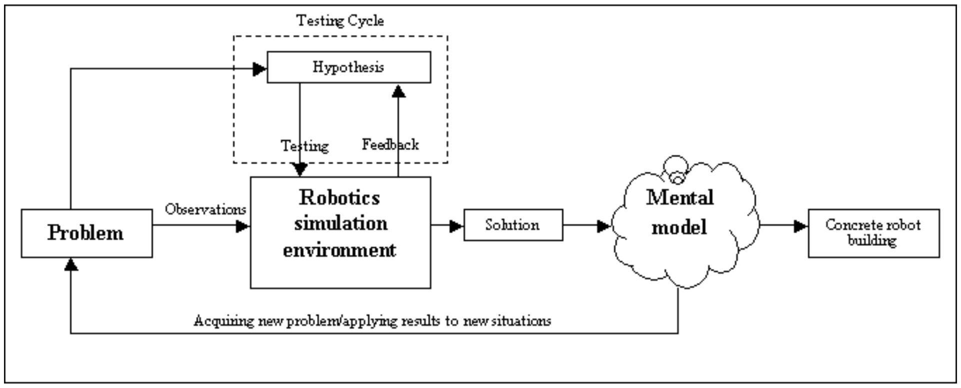


Figure 1: The process of constructing mental models in RSE.

The RSE's design is based on cognitive layering (The Empirical Modelling Research Group, 2002b). By separating concepts into layers the learner can control the pace that she is acquiring new skills and the way those skills are being developed. Our design has five levels to support different kinds of learning.

On the first level the learner is provided with ready-made robots with pre-programmed behaviours. The learner may activate the robot and observe its behaviour. The environment visualises the activities of the sensors, when they are activated, for example when running into wall. This is the "black-box" approach, where the internal functionality of the robot can be conjectured according to the observed behaviour of the robot. This helps the learner to realise the possibilities of the system and see the dependencies between situations and reactions.

The second level gives the learner control over the movement of the robot. The situation is analogous to radio-controlled toys, except the fact that the robots may have internal reaction made ready for example for running into wall. This way the learner may reproduce the situations where the robot's behaviour was especially interesting and observe them more carefully.

On the third level, the learner has to build the behavioural scheme for the robot. This includes creating the dependencies between the observables. For example the dependency between the motor rotation and tyres has to be made or the connection between the activation of touch sensor with the change of direction of rotation in motors. By linking the elements of the robot with dependencies, the behaviour of the robot is created. The third level allows also the learner to introduce new environments for the robot to act in.

The learner is allowed to construct the robot from scratch on the fourth level. She can add sensors of interest into the system and change the qualities of them. For example the rotation speed of the motor or the sensitivity of the light sensor can be changed.

The fifth level is the most open, where the learner can control everything. In addition to controls allowed by earlier levels, the learner can design sensors of her own or change the operation of the system. It is also possible to alter the source code of the RSE itself, whilst it is running. This kind of system "hacking" brings up interesting scenarios, where advanced learners can change the appearance and functionality of the system according to their specific needs.

5 Future Work

The current prototype simulation is a first attempt at creating an environment that is both useful to the children in the Kids' Club and matches their conceptual model of the programming environment for the robots. Our plan is to follow an iterative design methodology such that each cycle increases the number of children it is tested on. There should be a simultaneous reduction in the amount of changes needed to improve the environment (Pratt, 1998).

The initial work on the collaboration of using Empirical Modelling to help simulate Kids' Club activities shows that this approach gives a flexible environment within which exploratory experiments can be carried out, which can be conceptually applied to the task of creating behaviours for real-world Lego robots.

References

Eronen, P. J., Sutinen, E., Vesisenaho, M., Virnes, M., 2002a. Kids' Club - Information Technology Research with Kids. In: IEEE International Conference on Advanced Learning Technologies, ICALT. Kazan, Russia, pp. 331–333.

Eronen, P. J., Sutinen, E., Vesisenaho, M., Virnes, M., 2002b. Kids' Club as an ICT-Based Learning Laboratory. Informatica 13 (4), 1–12.

Executive Committee for RoboCup-2002, 2002. Robocup 2002. Http://www.robocup2002.org/.

Jormanainen, I., Kannusmäki, O., Sutinen, E., 2002. IPPE - How To Visualize Programming with Robots. In: Second Program Visualization Workshop, The 7th Annual Conference on Innovation and Technology in Computer Science Education. Aarhus, Denmark.

Lavonen, J. M., Meisalo, V. P., Lattu, M., 2001. Problem Solving with an Icon Oriented Programming Tool: A Case Study in Technology Education. Journal of Technology Education 12 (2), 21–34.

Norman, D. A., 1983. Some Observations on Mental Models. In: Gentner, D., Stevens, A. L. (Eds.), Mental Models. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Papert, S., 1980. Mindstorms, Children, Computers and Powerful ideas. All about LOGO - How It Was Invented and How It Works. Basic Books, New York.

Pratt, D., 1998. The Construction of Meanings IN and FOR a Stochastic Domain of Abstraction. PhD. Dissertation, University of London.

Roblyer, M. D., Edwards, J., 2000. Integrating Educational Technology into Teaching. Prentice Hall, New Jersey.

Roe, C., 1999. The EMPTY project: Empirical Modelling Principles for Teaching Youngsters. MSc. Dissertation, Department of Computer Science, University of Warwick.

Roe, C., Beynon, W. M., 2002. Empirical Modelling Principles to Support Learning in a Cultural Context. In: 1st International Conference on Educational Technology in Cultural Context. Joensuu, Finland, to appear.

Seel, N., 2001. Epistemology, situated cognition, and mental models: "Like a bridge over troubled water". Instructional Science 29 (4), 403–427.

The Empirical Modelling Research Group, 2002a. The Empirical Modelling Project. http://www.dcs.warwick.ac.uk/modelling.

The Empirical Modelling Research Group, 2002b. The Empirical Modelling Repository. http://empublic.dcs.warwick.ac.uk/projects.

The LEGO group, 2002. LEGO Mindstorms. http://mindstorms.lego.com/.

von Wright, J., 1993. Oppimiskäsitysten historiaa ja pedagogisia seurauksia. Opetushallitus, Painatuskeskus, Finland, Book is in Finnish. Title in English: The history of learning conceptions and their pedagogical consequences.

INVITED SEMINAR

Understanding Network Protocols. A Phenomenographic study

A. Berglund
Department of Information technology, Uppsala University, P.O. Box 325, SE - 751 05 Uppsala, Sweden

Anders.Berglund@docs.uu.se

1 Introduction

The aim of this research project is to explore university students' learning of advanced computer science concepts in an internationally distributed project course with the overall objective of improving learning and teaching of computer networks. This paper summarizes a full report that was previously published as a licentiate thesis at Uppsala University, Sweden (Berglund, 2002) and has been discussed at a licentiate seminar in March 2002.

In this paper, I will start by describing the project the students are taking. In the following section, I will discuss phenomenography and motivate my choice of this research approach. Section 4 briefly presents students' understanding of the network protocol TCP, while the last section discusses implications of the results for learning and teaching.

2 The student project

The learning that is investigated in this research takes place in a course in distributed computer systems and real-time programming, which is the centre of a joint research and development project: the Runestone initiative (Daniels, 1999). The students, who are majoring in computer science, take the Runestone course during their third or fourth year at Uppsala University, Uppsala, Sweden and Grand Valley State University, Allendale, MI, USA.

During the course, the students work in internationally distributed teams to jointly develop a software system that gives an end-user the possibility to play with a Brio labyrinth. The labyrinth is a Swedish wooden toy, the aim being to manoeuvre a steel ball from a starting point to a final point on the board, by tilting it so that the ball moves without falling into any of the holes. The original labyrinth has, as is shown in the left picture of Figure 1, knobs that are used to control the angle of the board. The labyrinth that was used was modified to have motors to control the board and a camera to give feedback to the controlling software system, as in the right picture of Figure 1. There were 14 groups of five or six students, each group comprising students from both universities, collaborating mainly by e-mail and Internet Relay Chat, IRC.

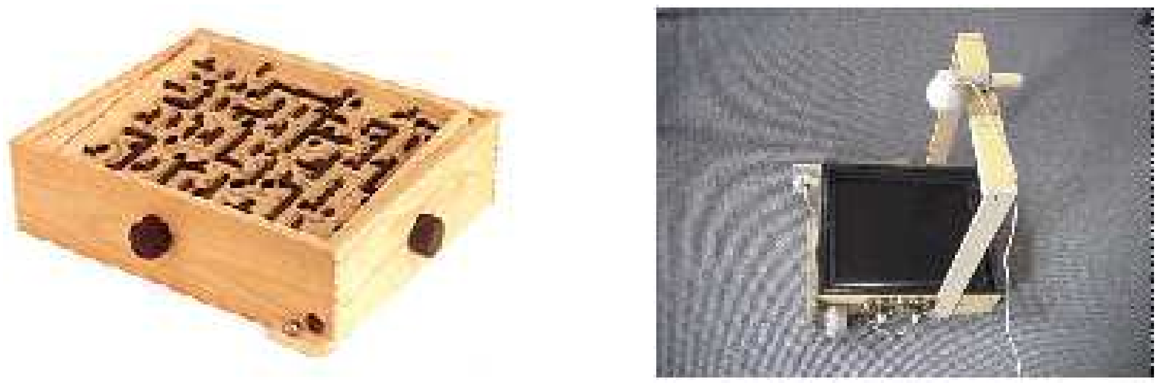


Figure 1: A Brio labyrinth, and a modified version with a camera and motors added (Berglund, 2002).

3 Phenomenography as a research approach

A phenomenographic research project aims at analysing and describing the variation in ways in which central concepts of the subject matter are understood or experienced by the learners (Marton and Booth, 1997). Phenomenographic research has been successful in studying learning in higher education, where the work of Booth (1992) concerning first year university students who learn to program, and of Cope (2000) concerning learning information system are relevant to this project. In my study, university students' experience of computer networks is in focus, and phenomenography offers the possibility for me, as a researcher, to investigate the students' own experience of network protocols.

One of the keystones of phenomenography is that the researcher can arrive at a limited number of qualitatively distinct categories of description which succinctly and adequately cover the countless ways in which a phenomenon can be experienced, or understood. The results are articulated in a set of qualitatively distinct categories of description that express the variation in how a phenomenon is experienced by the learners, called the outcome space. The outcome space that I arrive at thus gives me, as a computer scientist and phenomenographer, the possibility to relate the students' understanding of computer network to the goals of the education, as it is expressed in the course descriptions and as I, as computer scientist, understand the protocols. The strong focus on the object of the students' learning is an important feature of phenomenography as a research approach in the project that I present in this seminar, maintaining as it does the subject matter.

Phenomenographic research is not performed in controlled experiments or as quasi-experiments, but in "close proximity (both spatial and conceptual) to the learning situation [in which the students] find themselves" (Booth, 2001). She continues by arguing that it is not a quantitative methodology:

> The results are descriptive and lie at a collective level, in the sense that individuals are seen as contributing fragments of data that together constitute a whole and collective experience [...]. (p. 172)

The outline above indicates the roles that are given to the phenomenon (here: network protocols) and the experiencers (students taking an internationally distributed course in computer systems) in a phenomenographic research project. The object of the research is the relationship between these two entities; that is, the students' experience of network protocols. Neither the student nor the protocol in focus can alone serve as study objects. This perspective, a second order perspective, is illustrated in 2, where arrow (a) indicates the relationship between the learner and the object of their learning. At the risk of oversimplifying, the researcher investigates this relationship, and is in the same way a learner who learns about the students' experience of a particular phenomenon, as is indicated by arrow (b).

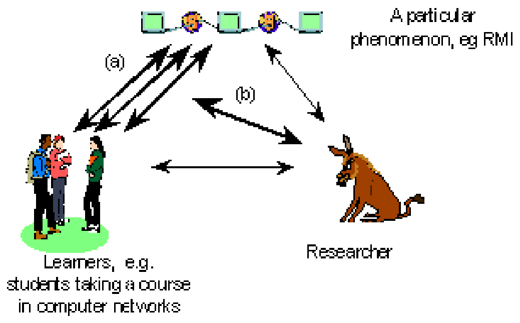

A particular phenomenon, eg RMI
(a)
(b)
Learners, e.g. students taking a course in computer networks
Researcher

Figure 2: The second order perspective, used in phenomenography (Berglund, 2002).

There are several reasons why this research is performed with a phenomenographic approach. The strong focus on the object of the students' learning is important to me, since it gives me, as a computer scientist, the possibility to learn about the students' understanding of computer science. The results of a phenomenographic research project, such as mine, also reflect and express the students' experience in a relatively direct way. Results are not described in abstract ways such as for example cognitive models or statistical correlation. I find that this closeness to the students' world and to the subject area offers rich possibilities to give results that can be used to obtain my initial goal, which is to improve education.

4 Students' understanding of TCP

Through the phenomenographic investigation three qualitatively different ways of experiencing TCP have been identified. An analysis of the different aspects of the categories of description for TCP is summarised in 1. The first column indicates what TCP is experienced as. The second column focuses on the world or framework in which TCP is experienced. The next column shows the technical characteristic that is understood for TCP. Finally, the last column indicates in what way TCP is described.

Differences between the categories, numbered 1 through 3 in the rows, are found in the framework of which the protocol is experienced as a part, as what the protocol is experienced and in what way it is described.

The first category describes an understanding where TCP is related to an experienced framework that consists of two specific computers, where data is transferred between these two computers. The network only exists as a background to this transfer. In the second category the framework is broader: TCP is experienced as integrated with and a part of an internet. Finally, in the third category, the framework has its limits outside an internet and the world of computers, and also takes human decisions into account. Here TCP is the result of decisions taken by a committee.

Table 1: Aspects of the different categories of description of TCP (Berglund, 2002).

	As what is TCP experienced?	As a part of which framework is TCP experienced?	What is the technical character of TCP?	How is TCP described?
1.	Safe communication	A framework of two specific computers	TCP is a protocol with acknowledgement	In concrete terms
2.	A connection	A framework of an internet		In an abstract way
3.	A standard for communication	A framework of a world outside the network		On a meta-level

TCP is, in all the three categories, experienced as an inseparable part of the framework to which it belongs. The protocol is integrated with specific computers, the network, or the world outside the network. A protocol needs its surroundings for its existence, and could thus not exist without the world of which it is a part. Neither would its surroundings be the same without the protocol.

The experiences of what TCP "is" or "means" differs between the three categories. The "meaning" of the protocol is closely related to the experienced framework of which the protocol is a part. When TCP is related to two specific computers, it is understood as a reliable way of communicating, that is, a user can know that no data is lost during the transfer. In the second category TCP is understood as a connection over the network. A connection has to be created or "set up". Once the connection is there, it offers safe communication. In category three TCP is experienced as a part of a framework that reaches outside the world of computers. TCP is a standard tool for communication; as a standard it is decided by a committee. The fact that it is a standard is what makes it useful.

When talking about TCP, the students frequently referred to the technical characterisation, or technical properties, of the protocol, telling the interviewer "how the protocol works". No variation in the understanding of this technical characterisation for TCP has been found in the data. TCP is experienced as a protocol with acknowledgement in the three categories. Rather, the technical characterisation is what gives a specific protocol its character; that makes it possible to recognise TCP as TCP, or UDP as UDP etc.

5 Understanding the results

The way(s) a certain student experiences a specific phenomenon, such as TCP, can change. A student does not have a given limit for the capacity to reach an advanced understanding. Rather, the student interacts with the phenomenon. His understanding of the phenomenon is then shaped by the phenomenon in the context in which the phenomenon is experienced, in the environment where the learning takes place, and the student him- or herself with his or her interests and previous understandings. Thus, it is worth studying what constitutes a good understanding, and how the universities can act to promote good understanding among their students.

Marton and Booth (1997) argue that relevant or meaningful learning is a change in the learner's capability of experiencing something into a new or different way. This definition of learning does not only indicate that some learning is meaningful, but also points out that there are less relevant forms of learning, as for example pure rote-learning. They also discuss good learning and argue that the ways in which learning is experienced "differ in richness (different aspects of learning that are discerned and held in focus simultaneously) and situational appropriateness (which particular aspects are held in focus under the prevailing conditions)." (p. 55). I will take this as a starting point for a discussion about situational appropriateness and richness of the students' experience of network protocols.

A conclusion that can be drawn from the argument of Marton and Booth is that the task at hand indicates which way(s) of experiencing a protocol are the most fruitful. For example, experiencing a protocol in a framework of two computers and described in a concrete way is closely related to programming, while a framework where the protocol is experienced as an integrated part of a network is useful for discussing the properties of protocols or which protocol to use in a particular situation.

Marton and Booth also argue that an understanding expressed in a higher category of description offers the broader perspective needed to inspect and evaluate an understanding expressed in a lower category of description. An example can serve to illustrate and concretise the reasoning. To evaluate the solution to a problem solved in a concrete way concerning two interacting computers, e.g. the coding of a program using TCP, it is necessary to shift to an understanding where the program is experienced in the framework of a network, and is discussed in an abstract way. By "stepping outside" the original reasoning to look at the problem as an issue of design instead of as an issue of coding, questions about the relevance of the solution can be discussed. A point of departure for such a discussion is Adawi et al. (2001), where we argue that variation in the context in which a phenomenon is experienced during a research situation supports that the phenomenon is experienced in new or different ways. A similar reasoning could be applied to teaching. Good teaching should then propose different contexts for a phenomenon, encouraging the students to shift between different ways of understanding a phenomenon.

References

Adawi, T., A.Berglund, Booth, S., Ingerman, A., 2001. On context in phenomenographic research on understanding heat and temperature. In: EARLI 2001. Fribourg, Switzerland, also available at http://www.docs.uu.se/˜andersb/lic/.

Berglund, A., 2002. On the Understanding of Computer Network Protocols. Licentiate thesis, Uppsala University, Uppsala, Sweden, Also available at http://www.docs.uu.se/˜andersb/lic/.

Booth, S., 2001. Learning Computer Science and Engineering in Context. Computer Science Education 11 (3), 169–188.

Cope, C., 2000. Educationally critical aspects of the experience of learning about the concept of an information system. Ph.D. thesis, La Trobe University, Bundoora, Victoria, Australia.

Daniels, M., 1999. Runestone, an International Student Collaboration Project. NyIng report 11, Linköping University, Linköping, Sweden, also available at http://www.docs.uu.se/˜matsd/NyIng.html.

Marton, F., Booth, S., 1997. Learning and Awareness. Lawrence Erlbaum Associates, Mahwah, NJ, USA.

DEMO SESSION

Virtual Teaching Environment for Basic Studies in Telecommunication

M. Kuusinen, P. Hiirsalmi, and A. Kaarna
Lappeenranta University of Technology
Department of Information Technology
P.O. Box 20
FIN-53851 Lappeenranta
{Miika.Kuusinen,Petri.Hiirsalmi,Arto.Kaarna}@lut.fi

Abstract

Virtual education methods have become more important in today's education. In this paper we present virtual teaching environment for the basic studies in telecommunication. Our viewpoint is that virtual teaching in basic courses should not replace traditional teaching methods completely but it should offer better possibilities for the students to learn and to be prepared for the examination. We are in process of developing virtual material for two basic courses in telecommunication. At the same time we are gathering information about the use of the material and about the effectiveness of the new process to the students' learning and exam grades.

1 Introduction

All universities in Finland are members of the Finnish virtual university (FVU) which aims to achieve structural prerequisite for large-scale and international virtual university action. The Finnish virtual university provides university students, teachers, researchers and other staff with a virtual campus through a common portal and net services including: information about on-line education; new opportunities to study through the net; training and courses for staff; tutoring, guidance and support; digital learning materials and access to the most modern learning environments; research services; various tools supporting study and research; carefully selected connections with the world's virtual campuses; business opportunities for companies wishing to take part in constructing the virtual campus. It can be said that the virtual university is a new method of networking between universities (Finnish virtual university , Suomen virtuaaliyliopisto).

Anyhow Finnish universities may not want to virtualize themselves completely. Teaching and learning are still based to interpersonal educational occasions between students and lecturers. Different kind of virtual teaching environments are developed more widely and rapidly for supplementary education courses and for open university education than for basic degree studies (Opintoaineistot verkossa-hanke, 2002).

At Lappeenranta University of Technology the basic studies in telecommunication are provided in two courses offered by Department of Information Technology: Introduction in Telecommunication 1 and Laboratory Course in Telecommunication. Both courses are compulsory for every student in information technology. They also form a cornerstone in studies for students whose major is datacommunication. Besides students in information technology these two courses are taken by many students from other departments as well as they are given in open university for adult students. Every year roughly 500 students in total enroll in these two courses. Basic knowledge and motivation among students about these subjects have been noticed to vary a lot. All this makes it quite a challenge for lecturers and laboratory assistants.

In the future we can also offer our virtual learning environment to other Eastern Finland Virtual Universities. As a part of the Finnish Virtual University project, the Eastern Finland Virtual University Network is a three-year network project (2001 - 2003) financed by the

Finnish Ministry of Education. The project partners are University of Joensuu coordinating the project, University of Kuopio and Lappeenranta University of Technology (The Eastern Finland Virtual University, 2002).

2 Purpose

The theory course consists of 28 hours of lectures and of an exam and it introduces the principles of telecommunications, different networks and telecommunication medias. The course also offers students theoretical knowledge that is required at the laboratory course. The course is mainly based on William Stallings's book "Data and computer communications" (Stallings, 2000).

Traditionally lecture transparencies, materials as well as instructions and other useful information have been available at courses' web pages. During the summer 2002 with the aid of Virtual University's funding we decided to make the material a bit more challenging and interesting for students. The purpose of our virtual material is to never replace the actual lectures or practical laboratory works but to offer students better chances and possibilities to study these topics. Our opinion is that virtual teaching can't replace completely traditional lectures of these basic courses. Most students don't have potential to study these courses on their own because the lack of know-how. All students at the university have opportunity to attend actual lectures and those whose level of knowledge is lower can get better chances to learn all important issues and get stable foundation for all future studies at Department of Information Technology. We have seen that students who are or who assume that they are familiar to subject matters of the courses do not have best motivation to attend the lectures. Unfortunately, this leads quite often to poor learning and exam grades.

There were three main ideas in developing the material, see Figure 1. Firstly, the students are able to study the material about the current subjects beforehand and so they have better state of readiness for the actual lectures. Secondly, it is easier for the student to motivate himself or herself to studying and prepare oneself for the exam taken after the lectures. Last but not least, one goal is to reduce the amount of work of assistants by replacing written exercises with automatically checked quizzes done online.

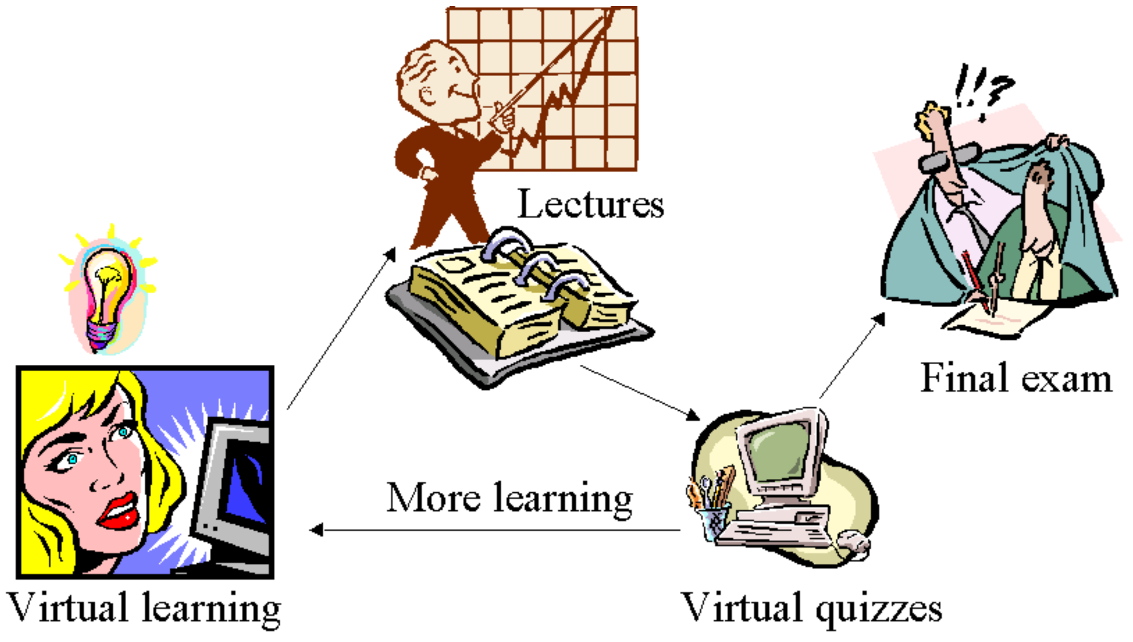


Figure 1: Learning process for the theory course consisting of virtual learning, lectures, virtual quizzes and an exam.

On the laboratory course each student chooses three different practical works out of six possibilities. The learning process is illustrated in Figure 2. Every topic contains three steps: pre-report, practical work and post-report. After the pre-report each student should have the basic knowledge about concepts and terms of the subject matter. The pre-report can be done either as a written report or by computer-aided quiz. The purpose of the computer-aided quizzes is to offer students an alternative way to measure their knowledge and to get instant feedback. This should also reduce assistants' routine work like checking of written reports. The laboratory work itself and the preparation of the post-report remain the same as earlier. The principles of the practical works are mostly based to the following books: Douglas E. Corner "Internetworking with TCP/IP volume 1: Principles, Protocols And Architecture" (Corner, 2000), W. Richard Stevens "TCP/IP Illustrated, Volume 1: The Protocols" (Stevens, 1994), Esa Kerttula "Tietoverkkojen tietoturva" (Kerttula, 1998).

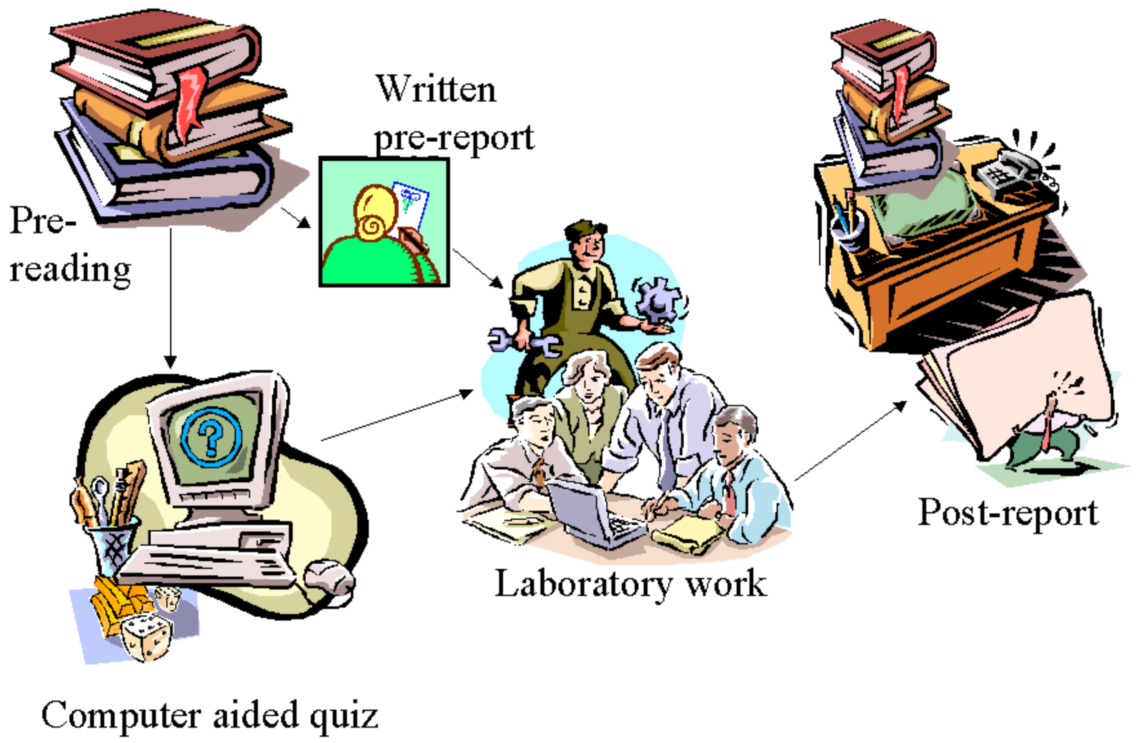


Figure 2: Learning process for the laboratory course consisting of reports and practical works.

3 Implementation

The background of designing the course was to keep the material as simple as possible so that there wouldn't be too much need for special network and computer equipment such as special software or browser add-ons. The body of the course is created in WWW-environment (World Wide Web) by using standardized HTML (Hyper Text Markup Language) (The World Wide Web Consortium, 2002). Actual material contains the same lecture slides used in lectures but they are enhanced with links to related material as well as with pictures, animations and short audio samples further explaining more complicated topics. Pictures consist of different kinds of diagrams, drawings and photos. Animations are created with animated GIFs (Graphics Interchange Format) or Macromedia's Flash (Macromedia Flash, 2002). Audio samples are made with RealNetworks' encoder producing commonly used RealMedia format (RealNetworks, 2002), (Karjalainen et al., 2002).

For each topic there are also voluntary quizzes available. The purpose of these quizzes is to measure student's knowledge and understanding about the current subjects as well as to prepare the student for the exam. Quizzes were made with WebCT as it turned out to be the most practical way (WebCT, 2002). WebCT environment contains support for providing the quizzes, checking the correctness of the answers and collecting information of the progress of the each student for the teacher.

At the moment the appearance and layout of the virtual material might not look that appealing, in fact many people might find it quite dull. We have tried to maintain readability and usability while designing the material. Even latest HTML-versions and computer equipment makes it possible to use a wide range of colors and background images etc. People are accustomed to read black on white or other high contrast color-combinations. For example people with various eyesight problems such as long-sightedness or colorblindness should be able to use the material without any problems (Follansbee, 2002). This is achieved by using big enough commonly used fonts such as Arial and different kind of font styles such as italic and bold where emphasis is needed.

4 Conclusions and Future Work

Most of the content is already finished but some additional enhancements such as animations, photos and audio samples will still be added in near future. As both courses are given in autumn 2002, we do not have any results yet whether this virtual environment will actually help the students to learn the topics better and to get better grades. However, we plan to monitor the results and student's experiences after the courses with a feedback system. We are also planning to research the relation of grades between those students who have been actively used the virtual material offered including taking the quizzes and those who have used more traditional ways of studying. Based on the research results we can improve the virtual material towards higher usability. Thus, it is essential to receive feedback from as many students as possible. Increasing interaction between students and lecturer at actual lecture situations is also something what we expect to happen and what we would like to experience.

We genuinely believe, that the provided virtual material will encourage the students to learn more about telecommunications and increase their motivation and knowledge about the available issues in these two courses. This applies especially to those students whose major is not software engineering or datacommunications. What we do expect is, that the students will get significantly better grades if they use our virtual material actively. However, we must also think carefully how much time and effort it is worthwhile to allocate on the preparation of the virtual material.

References

Corner, D. E., 2000. Internetworking with TCP/IP, Volume 1: Principles, Protocols And Architecture, 4th Edition. Prentice Hall.

Finnish virtual university (Suomen virtuaaliyliopisto), 2002. Available: http://www.virtuaaliyliopisto.fi/index.php?language=eng. Accessed: 26.09.2002.

Follansbee, T., 2002. Colorblindness and usability. Available: http://webword.com/moving/colorblindness.html, Accessed: 26.09.2002.

Karjalainen, A., Lehto, T., Nykänen, O., Ala-Ranta, M., 2002. Etäopetus multimediaverkoissa. Available: http://matriisi.ee.tut.fi/kamu/opekoulutus/verkkokurssi/index.htm, Accessed: 26.09.2002.

Kerttula, E., 1998. Tietoverkkojen tietoturva. EDITA.

Macromedia Flash, 2002.

Opintoaineistot verkossa-hanke, 2002. Yliopisto-opetus ja opintoaineistot verkossa loppuraportti. Available: http://www.opiskelijakirjasto.lib.helsinki.fi/koodi/rap.htm, Accessed: 26.09.2002.

RealNetworks, 2002. Available: http://www.real.com, Accessed: 26.09.2002.

Stallings, W., 2000. Data and Computer Communications. Prentice-Hall.

Stevens, W. R., 1994. TCP/IP Illustrated, Volume 1: The Protocols, 1st Edition. Addison-Wesley Pub. Co.

The Eastern Finland Virtual University, 2002. Available: http://www.joensuu.fi/isvy/isvyeng/index.htm, Accessed: 26.09.2002.

The World Wide Web Consortium, 2002. Available: http://www.w3.org/, Accessed: 26.09.2002.

WebCT, 2002. Available: http://www.webct.com/, Accessed: 27.09.2002.

Course Management System For Large Scale Courses

L. Malmi
Helsinki University of Technology
Department of Computer Science and Engineering
P.O.Box 02015, Finland

S. Ensio, T. Riski
Innofactor Ltd
Tekniikantie 12, FIN-02150, Finland

lma@cs.hut.fi,{sami.ensio,tuomas.riski}@innofactor.com

1 Introduction

Organizing large-scale courses with several hundreds of students requires strong managerial skills from the teacher. Actually, faculty members often spend more time in management operations, such as registration, calculating course results, receiving and classifying submissions, than in actual teaching. A similar problem may occur in virtual courses. A typical example is a case where 50 students submit solutions to 10 different exercises by email. Managing 500 email messages with different formats and attachments takes considerably time from the teacher. Incidentally, it might be easier to request all submissions to be printed on paper.

The Laboratory of Information Processing Science in Helsinki University of Technology (HUT) gives several computing and programming courses with 200 to 1500 students. Due to the masses all courses have developed some kind of automation to ease up the organization. Typically these include different Unix scripts, but several more elaborated systems exist, as well. Since these systems are often quite course specific and difficult to maintain, many similar operations are implemented again and again. In order to avoid this, a new project was launched to create a general purpose course management system in spring 2001. The project was established as a joint project with Helsinki University of Technology and Innofactor Ltd.

2 System overview

We listed the following key design issues.

1. The system should be web-based and work on most common browsers.
2. We should not build Yet Another Integrated Learning Environment like WebCT or FLE (WebCT Inc., 2002),(FLE3, 2002). On the other hand, the system should be more general than systems build around automatic assessment of exercises, like CourseMaster or Web@ssessor (Foxley et al., 2001),(Web@ssessor, 2002), or document sharing systems like BSCW (BSCW, 2002). Essentially the project concentrates on building general tools for management operations, not tools for presenting or assessing information nor CSCW tools. There is no pedagogical framework involved. Instead, the system could be used in organizing courses with quite different pedagogical designs.
3. The system should be an open environment which does not always need logging in with a user id and a password. The system should be able to send and receive email messages, as well as submit messages to newsgroups and allow links to external web-sites.
4. The system should be general enough to allow many kinds of courses which means strong customization facilities.
5. The system defines several roles for the users. *Visitors* can only see course information, but cannot do any actions. *Students* can register on a course, book assignments, submit solutions, view their marks *etc.* *Tutor assistants* typically use the system to view and grade submissions, check for students' status and change student data. *Course managers* set up the courses, define exercises, submission deadlines, course schedules *etc.*

Below we list briefly the chief functionalities available in the system.

- *Registration.* Students register on the course when logging in. They can self manage personal data.
- *Exercises and Course parts.* Course manager can set up any number of different exercises. Each exercise can be selected for a certain course part, and a course may have several course parts, recursively. For each exercise, submission deadline and accepted submission format (plain text, upload files, web URL) is defined. Students should conform to these settings, and for example, late submissions are automatically recorded.
- *Grading.* Course staff can view submissions on screen or download them as files. Grades with comments can be send to each student, and new deadlines can be given for rejected submissions. Students can view their own status and staff members can view the whole course status. Grades of single assignments and course parts can be combined in versatile ways to calculate the final grade of the course.
- *Managing classroom sessions.* Students can choose, which of available classroom sessions they would attend, or they can be ordered or randomly ordered to different sessions. Students are self allowed to exchange reservations.
- *Managing working groups.* Students can build new working groups, call for and accept new members to the group, or apply to a group. Groups can submit joint assignments and grades are recorded to all group members.
- *Examination.* Course manager can define sets of questions or multiple choice questions to be answered by the students. The system checks the answers and reports the results.

In addition, a number of other features, such as using peer reviewing, sending SMS messages and group mails to students, automatic building of student calendar, are supported.

3 Piloting

The system has been piloted in spring 2002 in a few courses of HUT. On the course of general computing tools (with over 400 students) a successful feature was that students could change themselves the classroom group they were assigned to. Attendance to classroom exercises are compulsory and managing the exchange of students between groups of limited size has been a permanent problem earlier. In the course of data structures and algorithms, peer reviewing of design exercise reports was carried out successfully among about 200 groups of students. Earlier this has been carried out by a number of scripts.

The main problem encountered during piloting, in addition to discovering quite a number of bugs and unwanted features, was that the user interface was too complicated both for students and teachers. Based on this the whole user interface was redesigned, simplified and reimplemented during summer 2002. The pilot tests in fall 2002 concentrate in tuning minor issues related to the new user interface. In principle, the system is then fully functional.

References

BSCW, 2002. BSCW home page. http://bscw.gmd.de/.

FLE3, 2002. Future Learning Environment. http://fle3.uiah.fi/.

Foxley, E., Higgins, C., Hegazy, T., Symeonidis, P., Tsintsifas, A., 2001. The Coursemaster CBA System, LTR report. Computer Science Department, The University of Nottingham, UK. http://www.cs.nott.ac.uk/CourseMaster/.

WebCT Inc., 2002. WebCT Company home page. http://www.webct.com/.

Web@ssessor, 2002. Drake Kryterion Inc. home page. http://www.webassessor.com/main.htm.

An overview of Virtual Approbatur

S. Torvinen
Department of Computer Science
University of Joensuu, Finland
P.O. Box 111
FIN-80101, Joensuu, Finland

sirpa.torvinen@cs.joensuu.fi

1 Introduction

The Finnish Ministry of Education is funding a three-year project during 2001-2003 to establish the Virtual University of Finland. One of the particular goals in the project is to develop new Computer Science education methods (Haataja et al., 2001). In the fall of 2000, the Department of Computer Science at the University of Joensuu began offering, within the framework of that project, a university-level Introductory Computer Science Curriculum (15 cu, 22.5 European Credit Transfer System credits, cps) to high school students in the surrounding rural region of the province of Pohjois-Karjala. Almost one half of the Curriculum concerned programming using the Java-language. In the fall of 2001 we extended this teaching experiment to a rather similar rural region in the neighboring province of Etelä-Savo and in the fall of 2002 the first university level students started to study Virtual Approbatur Curriculum. We have now students from a total of 19 rural high schools and from the Department of Teacher Education at the University of Joensuu (Savonlinna campus). In this report, we present the contents and arrangements of this distance learning program and resume the tools used in Virtual Approbatur.

2 Content of Virtual Approbatur

The Curriculum of the Virtual Approbatur Course (22.5 cps) includes eight (at the first year nine) different courses that will give students a basic knowledge of three main domains: Introduction to Computer Science (9 cps), Basics of Programming (9 cps) and Preliminaries of Computer Architecture and Operating Systems (4.5 cps). Students study independently over the Web using WebCT-learning environment. While each student learns in his or her own personal way, the learning environment should offer a rich assortment of learning tools for different kinds of students (Ellis et al., 1999). Our Web material is composed mainly in HTML and it supports the printed material and is acting like guide to the study process showing the paths of studying. The Web material brings the most essential parts of the course to the learners and quickly shows the student the structure of the domain. Additionally, the Web material includes examples and optional exercises, visualizations using MacroMedia Flash 5.0 or Jeliot2000 (Ben-Ari et al., 2002). We have used also some interactive Java-applets in order to deepen study process (students can e.g. build binary trees by Java-applet).

3 Implementations of courses

There is a wide experience that far more effort is needed in designing distance education courses than in contact-lessons in classroom (Cordani and Tucker, 1998). Most of the courses have a similar structure: students work from both printed and web-material, they do weekly assignments, and their learning outcome is evaluated through an exam. One of the design principles was to link the printed material with activating learning environment on the Web. However we have used following teaching methods in order to rich and vitalize the learning project (Haataja et al., 2001): *Group activities*: In a few courses students work in small groups. With this we encourage students to work collaboratively over the web. Members of the groups are studying in different High Schools and group is working using teamwork tools available in WebCT. *Learning by writing*: In the course of "Introduction to the Ethics of Computing" students compose an essay on a certain subject. In additionally students write a reflective portfolio concerning the development of own ethical knowledge during the course. *Days at campus*: To make the students experience themselves as members of the academic community, they are invited to the campus on four days during their studies. First meeting is before the web-based studies are beginning, two of the meetings are during the study-process (at the end of the first Fall semester as well as at the end of the first Spring-semester), and last one is the celebration at the end of Virtual Approbatur studies. *Summer School*: The theoretical course on algorithms will be arranged on campus, as a one-week intensive period at the beginning of the second Fall semester. During this week students get familiar with each other and do lots of group work based on the face-to-face lectures.

4 Experiences

Our main experiences with this project are the following (Meisalo et al., 2002):

- Student needs to be ready at work independently via Internet: if (s)he is not, (s)he will very probably be failed in this project and drop out.
- Plenty of time is needed: The main reason to drop out both in Year 2000 and Year 2001 was the lack of time.;
- Learning programming is the most difficult in Virtual Approbatur -project.
- High school students need more specific instructions for what to do.

In Table 1 we can see the amounts of participants in Virtual Approbatur during years 2000-2001. In Table 1 is also seeing the relative part of female and male students in each control date. Control dates are: 25.8. at the beginning of the Virtual Approbatur studies, 1.1. right after passing Programming, part 1, 15.3. at the end of the course of Programming, part 2 and 16.12. at the end of Virtual Approbatur.

Table 1: Participants in Virtual Approbatur.

Control date	Year 2000			Year 2001		
	Female	Male	Total	Female	Male	Total
25.8.	15.7%	84.3%	89	12.5%	87.5%	184
1.1.	14.3%	85.7%	56	10.9%	89.1%	137
15.3.	8.8%	91.2%	34	12.0%	88.0%	100
16.12.	10.0%	90.0%	20	NA[1]	NA[1]	NA[1]

[1] Data not available yet.

Our experiences in dropouts are very similar to earlier findings (Cornell and Martin, 1997) and course-level dropouts rates in Virtual Aporobatur varied from 0 to 42.9% on Year 2000 (Meisalo et al., 2002). Data from year 2001 are not completed while part of the courses are still ongoing (these years in table I means the year when students started to study Virtual Approbatur). According to Cornell and Martin (Cornell and Martin, 1997) it is very common that as many as 30 to 50% of students drop out during virtual courses. We have evaluated our courses by two methods: by the analyzed feedback given by students and tutor teachers and on the other hand by action research type of case study via questionnaires and interviews. We have immediately improved our courses based on the analyzed feedback and the results of our studies.

5 Future

In future we are going to improve our courses furthermore and concentrate our research especially into the following topics:

- Semi-automatic assessment Tools in web-based learning environment: especially use programming courses to help the assessment process with large number of students (Higgins et al., 2001).
- Visualizations tools: in order to help especially mid-performers on their study-process (Ben-Ari et al., 2002).
- Thinking tools for the net: we are interested in to study how we could use different kind of thinking tools, like concept maps or intelligent portfolios, in order to help students with their study process over the web.
- We are also interested in to apply new techniques like woven stories (Gerdt et al., 2001) into our Virtual Approbatur -program, e.g. in the courses that we use teamwork methods.

References

Ben-Ari, M., Myller, N., Sutinen, E., Tarhio, J., 2002. Perspectives on Program Animation with Jeliot. In: Diehl, S. (Ed.), Software Visualization. No. 2269 in Lecture Notes in Computer Science. Springer-Verlag, pp. 31–45.

Cordani, J. R., Tucker, R. J., 1998. Tools for Higher Education Distance Teaching. In: Proceedings of the 26th SIGUCCS Conference on User Services. Bloomington, USA, pp. 71–76.

Cornell, R., Martin, B. L., 1997. The Role of Motivation in Web-Based Instruction. In: Khan, B. H. (Ed.), Web-Based Instruction. Educational Technology Publications. pp. 93–106.

Ellis, A., Lowder, J., Robinson, J., Hagan, D., Doube, W., Tucker, S., Sheard, J., Carbone, A., 1999. A collaborative strategy for developing shared Java teaching resources to support first year programming. In: Proceedings of the 4th annual ITiCSE 1999. ACM Press, pp. 84–87.

Gerdt, P., Kommers, P., Looi, C., Sutinen, E., 2001. Woven Stories as a Cognitive Tool. In: Cognitive Technology 2001, LNAI 2117. Springer-Verlag, pp. 233–247.

Haataja, A., Suhonen, J., Sutinen, E., Torvinen, S., 2001. High School Students Learning Computer Science over the Web. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ) 3 (2), Available in http://imej.wfu.edu/articles/2001/2/04/index.asp.

Higgins, C., Suhonen, J., Sutinen, E., 2001. Model of a Semi-Automatic Assessment Tool in Web-Based Learning Environment. In: Proceedings of the 9th International Conference on Computers in Education (ICCE/SchoolNet2001). Seoul, Korea, pp. 1217–1224.

Meisalo, V., Sutinen, E., Torvinen, S., 2002. How to improve the virtual programming course?, Accepted in Frontiers in Education Conference (FIE), Boston. Available in http://fie.engrng.pitt.edu/fie2002/papers/1180.pdf.

Old-Fashioned Teaching Using the Net

A. Wikla
Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 UNIVERSITY OF HELSINKI

`Arto.Wikla@cs.helsinki.fi`

Today there are many fancy ideas of *"virtual"* teaching and learning using the Internet or some more restricted net.[1] *"Automating the university"* seems to be the goal...

Being a university teacher I have a strong prejudice that personal, *human-to-human* teaching has many important factors that are not present in *man-machine* communication. There are also many *"non-technical"* aspects in technical subjects: understanding what you are doing and why you are doing it, goes beyond *"knowing the syntax"!* This is important even in learning programming, for example.

I have been using www-pages as *"The Place"* where all my teaching material exists as long as from the spring 1997. Our courses consist of weekly lectures (2x2 hours) and weekly excercises (1x2 hours). The number of students in each course have often been several hundreds.

My courses' pages normally include:[2]

- important messages and news
- general information about the course
- all the lecture material as www-pages; the material used in the lectures is *exactly* and even physically the same that the students may read at home or in an *"Internet Cafe"*
- excercise page; the problems appear weekly, and so do the example solutions - the latter of course only after the weekly exercises
- the result page: this page appears at the end of the course
- other useful information.

In some courses using a Usenet group has also been successful - especially in courses for 2nd year students: wild and free talk! The teachers have tried to hold back before answering, and often the students have been able to help each other.

I have used the Net this way for quite long already:

- Java-ohjelmointi (Java-programming, first version) (spring 1997)
- Johdatus ohjelmointiin (Introduction to programming) (in Java) (autumn 1997 - summer 1999, autumn, spring, summer) (6 times)
- Tietorakenteet (Data Structures) (spring 2000, 2001, 2002) (3 times)
- Ohjelmoinnin perusteet (Introduction to Programming) and Java-ohjelmointi (Programming in Java) (autumn 1999 - autumn 2002, autumn, summer) (7 times).

And the results have not been bad. My course pages can be seen starting from the page http://www.cs.helsinki.fi/u/wikla/. On practically every course page there is also the students' course evaluation, which might be interesting reading.

[1] See for ex. http://www.virtualuniversity.fi/.
[2] See for ex. http://www.cs.Helsinki.FI/u/wikla/JohdOhj/OhPe/indexS02.html.

Reintroducing Exactness into High-School Mathematics - Remarks on the History and the Future of Quantification

T. Kavander, S. Pyöttiälä, and T. Salakoski
University of Turku and TUCS
Lemminkäisenkatu 14-18A 20520 Turku

M. Peltomäki
Kupittaa High School and TUCS
Sireenikuja 1 20720 Turku

`sami.pyottiala@it.utu.fi`

1 The demands of information technology on mathematics education in high school

Information technology has a great social meaning in Finland. There are IT-professionals as well as large companies that have a big influence on our economy through for instance offering many jobs here. In order to keep up this development it is important to make sure that the standard of information technology teaching keeps getting better or at least stays at a reasonable level.

Information technology is closely related to discrete mathematics, and for example programming requires the active development of ones constructive thinking. These matters are generally present in the society, but still their learning does not begin until after high school. Therefore it would be appropriate to adapt the contents of mathematics education in high school to make discrete mathematics, constructive thinking and mathematical proving familiar to students already in high school. This would help raise the standard of mathematical sciences education in the university level without lengthening the time spent in the studies.

In order to present mathematical phenomena accurately and formally quantifiers are needed alongside natural language. Nowadays quantifiers are not used in high school mathematics but instead matters are presented using natural language or even left to the students intuition.

2 Presenting quantifications in the past and nowadays

Even though quantifiers are not used in present day textbooks they have been used mathematics education in high school in past decades. Quantifiers were introduced to textbooks soon after the Finnish National Board of Education confirmed the new curriculum for high school mathematics in 1973. The new curriculum was based on the suggestions of the Nordic mathematics education reform committee from 1967. The committees suggestions included a statement on limit in classical analysis stating that the formatting of definitions can be facilitated and their understanding can be improved by using quantifiers. (Anon, 1967). In addition the board wanted to include logical symbols for the words ‘and’, ‘or’, ‘implicates’ and ‘equivalent’ as well as the concept of equivalence relation in the curriculum for the tenth school year.

In 1985 a new curriculum was asserted. As a result especially theoretical aspects and formalism were reduced. In textbooks expressions with quantifiers were substituted with natural language, pictures and intuition. The curriculum of 1994 has continued the same trend. This development is generally considered favourable because now students can concentrate on understanding the actual mathematical matters instead of paying too much attention on interpreting concise expressions with quantifiers.

3 Combining the exact and the illustrative

One of the goals mentioned in the 1994 curriculum was that students should learn to read mathematical texts and appreciate the exactness and clarity of the presentation (Opetushallitus, 1994). Also in the 1985 curriculum one of the mentioned goals for higher level mathematics was to get used to using exact and correct markings as well as to read mathematical texts (Kouluhallitus, 1985). These goals are, however, somewhat contradictory to the current situation. When observing current textbooks, it is obvious that quantifiers, a significant aspect of mathematical texts, are not present. In addition, it can be said that quantifiers bring the kind of exactness to mathematics that cannot be achieved through the use of natural language. Clarity, then again, has been achieved in textbooks by using natural language, hidden quantifiers and illustrative pictures concurrently.

Quantifiers should be returned to high school mathematics educations in order for the abovementioned goals to be achieved. As a difference to 1973–85, natural language and illustrative pictures should be used alongside quantifiers. Exact and visual presentations do not exclude but actually support each other. Information technology can be used to aid learning by for example by using an editor that gives instant feedback and visually presents, for example with colours, how the quantifiers function. This would also be in line with the constructive learning theory. In addition, structured derivations (Back and von Wright, 1999) that support the combining of formal and informal presentations could be used as a learning device.

References

Anon, 1967. Pohjoismainen koulumatematiikka. Mietinnön suomenkielinen lyhennelmä. Statens tryckericentral, Stockholm.

Back, R.-J., von Wright, J., 1999. Structured Derivations: a Method for Doing High-School Mathematics Carefully. Turku Centre for Computer Science: TUCS Technical Report, Turku.

Kouluhallitus, 1985. Lukion opetussuunnitelmien perusteet. Valtion painatuskeskus, Helsinki.

Opetushallitus, 1994. Lukion opetussuunnitelmien perusteet. Painatuskeskus, Helsinki.

Author index