Juraj Hromkovič, Richard Královič,
Jan Vahrenhold (Eds.)

ISSEP2010

Proceedings of Short Communications

4th International Conference on Informatics
in Secondary Schools: Evolution and Perspectives

Zürich, Switzerland, January 13–15, 2010

Editors

Juraj Hromkovič
Richard Královič
ETH Zürich
Informationstechnologie und Ausbildung
CAB F16, F13.1
Universitätstrasse 6
8092 Zürich, Switzerland
E-mail: {juraj.hromkovic, richard.kralovic}@inf.ethz.ch

Jan Vahrenhold
Technische Universität Dortmund
Foundations of Computer Science and
Computer Science Education Group
Chair of Algorithm Engineering
Faculty of Computer Science
Otto-Hahn-Str. 14
44227 Dortmund, Germany
E-mail: jan.vahrenhold@cs.tu-dortmund.de

Cover design: Jan Lichtensteiger

ISBN: 978-3-909386-28-4

Published by ETH Zürich, 2009

Preface

The International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP) is an emerging forum for researchers and practitioners in the area of computer science education with a focus on secondary schools.

The ISSEP series started in 2005 in Klagenfurt, and continued in 2006 in Vilnius, and in 2008 in Toruń. The 4th ISSEP took part in Zurich. This volume presents 9 short communications presented at ISSEP 2010.

The ISSEP conference series is devoted to all aspects of computer science teaching. In the preface of the proceedings of ISSEP 2006, R. Mittermeir wrote: "ISSEP aims at educating 'informatics proper' by showing the beauty of the discipline, hoping to create interest in a later professional career in computing, and it will give answers different from the opinion of those who used to familiarize pupils with the basics of ICT in order to achieve computer literacy for the young generation." This is an important message at this time, when several countries have reduced teaching informatics to educating about current software packages that change from year to year. The goal of ISSEP is to support teaching of the basic concepts and methods of informatics, thereby making it a subject in secondary schools that is comparable in depth and requirements with mathematics or natural sciences. As we tried to present in our book "Algorithmic Adventures. From Knowledge to Magic," we aim at teaching informatics as a challenging scientific discipline, full of puzzles, challenges, magic and surprising discoveries. Additionally, this way of teaching informatics is also a chance to import the concept of engineering to schools, by merging the mathematical analytic way of thinking with the constructive work of engineers in the education of one subject.

To underline informatics as well as informatics didactics as scientific disciplines, ISSEP 2010 had two special tracks. The track "Contributions of Competitions to Informatics Education" was based on the fact that taking part in different kinds of competitions provides a valuable contribution to knowledge acquirement and supports the development of problem-solving skills in a creative way. Organizing a competition includes addressing the following questions:

- Which kinds of competitions are especially well suited for achieving which goals?
- How should one create and choose tasks and rules for such competitions?
- What are the achievements of the competition participants, in particular in relation to their training process?
- What is the influence of competitions on the educational processes in secondary education?

The starting point to this track was provided by the invited talk "Sustaining Informatics Education by Contests" by Valentina Dagienė.

The second track, "Empirical Research," pointed out that the community of computer science didactics has to strengthen its effort in empirical research in order to be as serious as the didactics of mathematics and physics are. The main questions posed were:

- What is "good empirical research?"
- Which rules should be followed to produce "good" empirical results?
- Which criteria can be applied to recognize "good" empirical results?
- What are the pitfalls of interpreting empirical results?

To make ISSEP 2010 attractive due to high-quality contributions, we increased the number of invited speakers to six. In addition to Valentina Dagienė (Vilnius), we invited the internationally leading experts Wilfried Bos (Technische Universität Dortmund), David Ginat (Tel Aviv University), David Gries (Cornell University), Allen B. Tucker (Bowdoin College), and Amiram Yehudai (Tel Aviv University) to give talks about different aspects of computer science education.

The program committee of ISSEP 2010 consisted of:

- Peter Antonitsch (University of Klagenfurt)
- Owen L. Astrachan (Duke University)

- Ralph-Johan Back (Abo Akademi University)
- Harry Buhrman (CWI & University Amsterdam)
- Valentina Dagienė (Institute of Mathematics and Informatics, Vilnius)
- Judith Gal-Ezer (The Open University of Israel)
- David Ginat (Tel Aviv University)
- Juraj Hromkovič (ETH Zürich)
- Peter Hubwieser (TU München)
- Ivan Kalaš (University Bratislava)
- Peter Micheuz (University Klagenfurt)
- Roland Mittermeir (University Klagenfurt)
- Wolfgang Pohl (Bundeswettbewerb Informatik)
- Ulrik Schroeder (RWTH Aachen)
- Jarkko Suhonen (University of Joensuu)
- Maciej M. Sysło (UMK Torun, University of Wroclaw)
- Jan Vahrenhold (TU Dortmund)
- Tom Verhoeff (TU Eindhoven)
- Michal Winczer (UK Bratislava)

I would like to express my deepest thanks to all members of the Program Committee for serving and thus contributing to the high standard of the ISSEP series among the conferences devoted to computer science education.

November 2009 Juraj Hromkovič

Table of Contents

From Object-Orientation to Human-Centeredness

Peter K. Antonitsch

Alpen-Adria Universität Klagenfurt
Institut für Informatiksysteme

Peter.Antonitsch@uni-klu.ac.at

Abstract. A particular learning situation is determined by the learning content, the social context of the learning community, and the individual disposition of the learner. Although Informatics didactical research starts to recognize the importance of situated cognition, yet principal interest is taken in development and analysis of artifacts that foster the learning of Informatics. The individual human actor as determining constituent of the learning process is close to being neglected. Proceeding from a self-reflective experience and combining older findings in Mathematics and Informatics didactics with the author's experiences, this article points at ways to direct attention towards the »human factor« in learning Informatics.

In schools, learning processes are supposed to take place permanently. Students are confronted with new material, thoughts, and corresponding activities. Some of the students join in, others do not. In Informatics classes, the reasons for not joining in can be manifold: It could be due to social issues inside the learning group, the potential learner could find the learning content inappropriate, or the provided tools might seem too complex. In most cases, we simply don't know which case holds.

1 A Self-Reflective Approach

Teaching and reflecting should go together ([1], p. 57). Teachers are supposed to reflect upon their own teaching and upon the student's learning progress to coach them best possible. But it is not common practice yet, that teachers care for the student's individual learning process and problems, or guide learners to become aware of their learning habits by themselves. To me, self-reflection revealed the importance of these aspects.

1.1 The Issue: Microsoft Office Excel and Open Office Calc

Combined with Visual Basic for Applications (short: VBA) Microsoft Excel can serve as a scalable learning environment for first programming experiences. Therefore VBA has been my preferred tool to teach programming basics during the past few years, following a grown course-plan with scalability of in- and output as one of its key features: In programmable spreadsheet-environments in- and output can be managed by means of cell-access, predefined GUIs or by user-defined dialogs. Keeping basic operations as easy as possible was the didactical motive [2].

What to do, when literally out of a sudden (but fortunately at the beginning of a new school year) the school-licence for Microsoft Office was cancelled and teachers were advised to use the OpenOffice suite instead? Of course, as Microsoft Excel and OpenOffice Calc seem almost identical from the outside, the adaption of the tried concept was the choice of the moment. But I found out soon, that OpenOffice Basic (short: OOB) provides no predefined command to access cell-content. Furthermore, OOB necessitates the creation of an abstract »Universal Network Object« to display a user-dialog that already has been designed within the visual environment.

Therefore, the intended exchange of the software-tool seemed to replace simplicity for the learners by complex navigation through the hierarchy of objects. This (and some other) »didactical deficiencies[1]«

1 It is fair to note, that the provided structure has shortcomings from the learners' point of view, but makes quite sense from a programmers' point of view (see [3] for

defined a setting which is supposed to be typical for learning situations in Informatics classes:

- There was a (seemingly) well defined task – adapting the course plan to allow a smooth introduction to programming basics with OpenOffice Basic.
- The task had to be accomplished within a certain period of time – at least as soon as the programming units were about to start.
- The tool to accomplish the task was provided – OpenOffice Calc with OOB.
- A secondary problem proved to be the true challenge – getting on well with the tool (to provide software abstractions hiding the complex hierarchy of objects).
- I (the »learner«) had no idea how to deal with the challenge (in the first place).

1.2 The Tool and the Learner

When learning Informatics, it is not uncommon that tools provided to solve a problem become part of the problem. It is also not uncommon to blame it on the tools. Self-reflection started, when I asked whether I myself was a part of the problem, too.

In the situation outlined before, I realized that my programming strategies developed with VBA were not applicable to programming with OOB: VBA makes it easy to deal with spreadsheet-objects of any kind. Below a certain level, programming with VBA requires little comprehension of the underlying hierarchy of objects: Objects appear to be organized within a flat hierarchy, and therefore all relevant VBA-objects and basic properties seem to be accessible »from everywhere and all the time«. On the contrary, adapting OOB meant to make use of (parts of) the object-hierarchy. In short: My mental model had to be updated, and: The »update« enabled me to accomplish the task.

---

hints how to improve the Open Office Basic programming-environment for learning purposes).

What did this experience mean to me? On second thoughts and having the interrelation between the learner and the learning situation in mind, the process of self-reflection suggested that it was irrelevant at that point, which of the both environments is the better choice when teaching programming fundamentals. It was even beside the point, whether the mental model on hand was good or bad, right or wrong. The key point was that the mental »working model« that had proven viable up to that moment did not match the new setting.

1.3 The Human Factor

Usually problems accompanying the learning process become visible at object-level, when the learner is not able or willing to manage the learning content and/or the tools the way he or she is supposed to. To me, self-reflection unveiled a close connection between the learner and the learning: Learning connects two »states of mind« represented by the specific mental models before and after the learning process. Additionally, the actual mental model of the learner can interfere with the learning process. Constructivism confirms this personal finding [4]: The learner's perception of the world determines the individual approach to a posed problem, the learning progress and the errors that accompany learning. Consequently, the learner is the inevitable »human factor« that has to be considered when looking at learning processes. Furthermore, self-reflection reveals the teacher as another component of this human factor. Neglecting (although not underestimating) the learner's private surroundings this yields four main constituents of each learning process (Fig. 1).

2 Considering the Human Factor in Informatics Classes

Didactical literature reports different strategies to focus on the learner's individual approach to a given problem. Among them, interviews are the tools of choice to investigate the process of reasoning, especially when reasoning leads to errors.

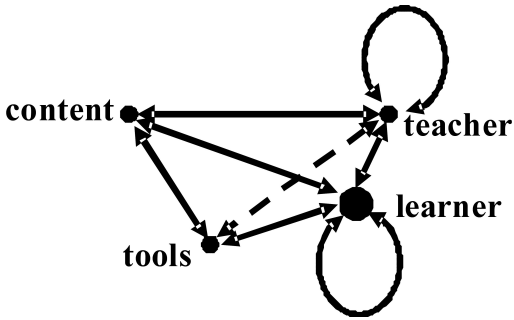


Fig. 1. »Learning tetrahedron« including four main constituents of learning processes at school and laying emphasis on the »human factor«. The arrows denote mutual dependences that (usually) lead to interactions (the dashed connection between content and teacher does not indicate a less important dependence but should produce a three-dimensional impression, placing the learner into the foreground). The reflexive learner-learner and teacher-teacher dependences have a twofold meaning: Learning humans tend to interact with each other, and on the other hand every learning process requires interaction of the learner (and the teacher) with himself!

In 1980 P. Rosnick and J. Clement investigated the types and source of errors at translation of Mathematical texts into formulas by means of a tutoring strategy including interviews [5]. Solving problems like "Write an equation for the following statement: There are six times as many students as professors at this university. Use S for the number of students and P for the number of professors." led to translation errors ("6S = P") that proved robust against explanations by the tutors. A possible source for this error was identified by R. Davis, who pointed at "commonlyshared frames" that might be used by students when dealing with problems of that kind [6].

Similar investigations in the field of Informatics were conducted by E. Soloway et al. concentrating on translation errors by novice programmers. They identified "faulty/incomplete understanding of programming concepts" to be one of the sources for errors when learners translate textual problems into program code [7], but also pointed out that "[...] for most computerized tasks there is some

model that a novice will use in his or her first attempts. We need to understand when is it appropriate to appeal to this model, and, when necessary, how to move a novice to some more appropriate model." [8].

These findings stress the importance of mental models (or »frames of thought«) to understand individual learning paths and, consequently, to focus on the human factor. But how can the teaching practitioner learn about the learners' mental models?

2.1 Predicative and Functional Cognitive Structures

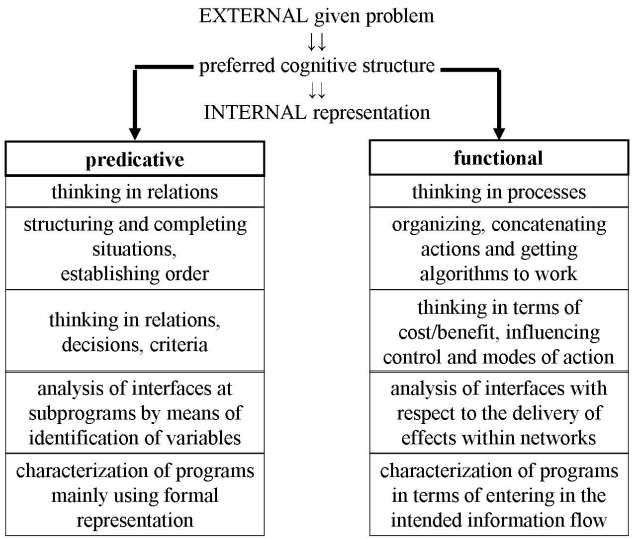


Fig. 2: Survey of predicative versus functional cognitive structures relating to algorithmic thinking (source: [9]).

Researchers in Cognitive Mathematics discovered two different cognitive structures that control the transfer of an external given problem into an internal representation, and that therefore represent (basic) mental models. I. Schwank states [9]: "We distinguish between a predicative structure, which is more concentrated on networks of

relations and structures, and a functional structure, which lays preference on thinking in terms of effects and organizing sequences of actions." (see Fig. 2).

Depending on the preferred cognitive structure, people develop different perspectives onto a given problem and its possible solution. Predicative cognitive structure favours the mutual substitution of single items (»static exchange«), while functional cognitive structure supports to move or add items (»dynamic exchange«) [10]. Due to this correspondence it was feasible to develop easy-to-use tasks helping to decide upon the preferred cognitive structure. Among other diagnostic tools, I. Schwank and her research team used tasks "to find a missing figure, which fits suitably into a set of 8 given figures arranged in a matrix." [11] (Fig. 3).

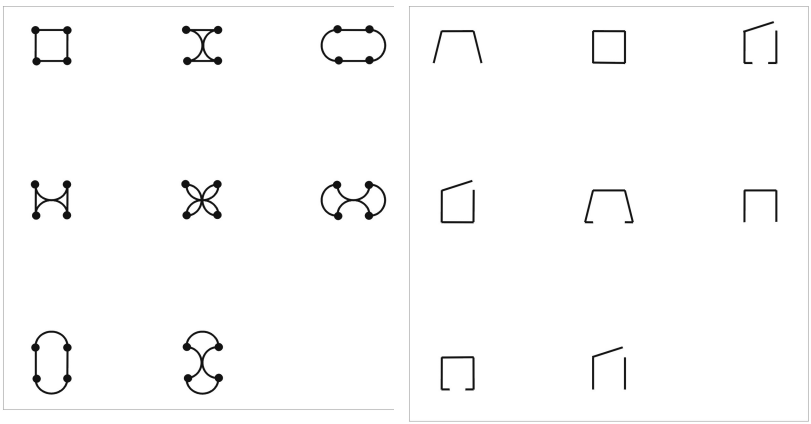


Fig. 3: »Fitting into matrices«-tasks: The left-hand task could be solved using a predicative or a functional analysis, while the right-hand task definitely privileges a predicative one [12].

2.2 A Personal Experience: Cognitive Structures and Databases

Although cognitive structures have been identified with regard to algorithmic thinking, they seem to have attracted almost no

attention among Informatics didacts[2]. I came across the framework of cognitive structures when musing on the question why certain students that perform poorly in programming courses get on well with databases (and vice versa). Then, programming was procedural yet and I considered dealing with procedures as thinking in processes, while representing databases by means of ER- or UML-diagrams seemed to be rather predicative.

I assumed that students, who find it hard to »grasp the structure« of (procedural) programs, possess a rather predicative cognitive structure, and (quite symmetrically) that students, who have difficulties to »find their way« through the static structure of relational databases, possess a rather functional cognitive structure. To put this assumption to the test, I handed out some »fitting into matrices«-tasks and instructed the students not only to fill in the missing figure but also to write down their arguments why it fits in. The presentation and discussion of considered solutions, combined with observations from the preceding programming-lessons and the subsequent »traditional«[3] database-lessons proved the conjecture qualitatively right.

This encouraged me to think of ways to open »usual« representations in Informatics to both the predicative and the functional cognitive structure. The first attempt was to allow a functional approach to databases, which led to a novel pattern for database instruction guiding my database lessons until today: Proceeding from a rather complex but ready-to-use (Access) sample-database, the students are guided to explore its structure by »moving in the relational model« and by »processing relations« in order to collect the information that is needed for a special query, all by themselves (see [15, 16] for a more detailed description). On the other hand, as the sample-database is represented by a (static) UML-

2 Publications that refer to cognitive structures AND consider an Informatics point of view originated within the research group of Cognitive Mathematics at the University of Osnabrück/Germany (see [15], [16], for instance).

3 Here, »traditional« denotes the common sequence of »designing a (static) conceptual model« – »transforming that model into a (static) relational data model« – »querying the database«.

diagram, this pattern allows for the predicative approach as well. Judging form the success of this concept, opening learning situations to a »dual approach« pays off: While solving common problems with the same software tool as ever, Students become more active learners when the design of the problem supports their cognitive structure: The learning results improve.

2.3 Cognitive Structures and Programming

Database lessons can be enriched by adding functional elements to common problem representation. But are there further traces of functional and predicative elements in Informatics classes? I suppose, the answer is "yes": In the last few years »click'n code« programming environments like Lego NXT-G or Scratch have become quite popular. Although not documented, these environments seem to support a predicative and a functional point of view as well! The coding of a program no longer rests on textual representation, but programs are put together with provided »programming-bricks« (see Fig. 4). These programming-bricks might be viewed as static elements that can »stick together« in certain ways, while a single program might be seen as some pattern being made of theses static elements and providing a certain functionality. On the other hand, the programming-bricks might be regarded as dynamic elements, each of which provides a certain functionality. Then, particular »cooperation« of these functional parts constitutes another dynamic element that is called program[4]. This functional point of view is supported by visually represented objects (sprites, robots) that can be animated by means of a program.

4 As both programming environments foster object oriented thinking, it is fair to note, that object-oriented programming per se seems to offer a blend of predicative and functional elements: While a UML diagram used for object oriented design is a rather static (and therefore predicative) representation of the programming project, methods represent the (functional) aspect of dynamic change (of an objects state). But these aspects represent different levels of abstraction! On the other hand, both Scratch and Lego NXT-G mainly focus on methods, adding the predicative aspect at the level of the functional aspect.

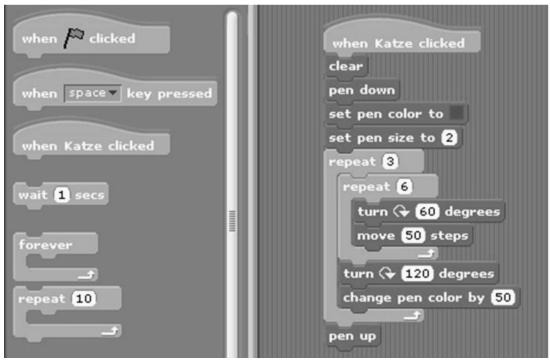


Fig. 4: Programming environment (Scratch) where predefined programming-bricks (left) are combined to build a program (right). The programming-bricks can be seen as static elements that form a pattern providing functionality (predicative approach) or as functional parts that cooperate to produce another, bigger functional structure (functional approach).

2.4 Interaction Patterns in Informatics Classes and Cooperative Learning

From personal experience, the concept of cognitive structures helps to understand learning problems that become visible at »object level« but originate in misconceiving. Furthermore, it enables the teacher to modify learning content to meet the preferences of most learners, or to identify software tools that might have the potential to promote individual learning paths.

These are prerequisites to focus on the human factor in Informatics classes. Enabling the students to tackle problems in an individual way empowers them to solve problems by working in groups and by working on their own (i.e. without constant guidance by the teacher). Such learner-centred classroom organization fosters individual learner-teacher-interaction and the teacher's monitoring of learner-learner-, learner-content- and learner-tool interaction. But we have to be careful! »Learner-centred« is not necessarily equivalent to »learning-centred«. This shall be illustrated by an observation originally published in [17]:

In the course of an Informatics lesson that was given by a pre-service teacher the students had to work on a problem concerning Caesar's cipher. During a learner-centred sequence, when the students had to provide a certain spreadsheet-solution, a student asked for help, pointing and looking at the computer screen: "[...] There, that does not work". The pre-service teacher gave the clue: "Here, [pointing and looking at the computer screen!] you have to reference these two cells".

This incident is remarkable for two reasons:

- The learner-tool interaction »jammed« the learner-teacher interaction: Instead of establishing a real face-to-face situation, the student and the pre-service teacher communicated with each other via the representation on the monitor (see Fig. 5).
- The learner-tool interaction also hampered the learner-content interaction: As the pre-service teacher found out later, the student was caught by the representation on screen. After switching media, the student solved the problem quite soon by herself after »doodling« possible solutions on a sheet of paper!

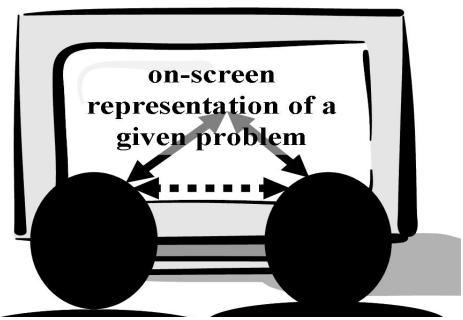


Fig. 5: Dominance of learner-tool interaction over learner-human interaction: In the face of the computer monitor humans tend to establish an »indirect human-computer-human« communication (continuous arrow) instead of (direct) face-to-face interaction (dotted arrow) [17].

Of course, a situation like this can occur at interactions between learners as well. Furthermore, especially when better-informed

students are asked for help, they often »help« by solving the particular problem by themselves (in Informatics classes commonly by means of the used software-tool). It is almost needless to say, that working together this way avoids efficient learning on the part of the asking student. Obviously, considering the human factor in class needs more than just providing an optimal learning environment. Efficient learning in a learner-centred environment needs basic (social) skills on the part of the learners.

Cooperative Learning [18] is a pedagogical framework providing strategies how to acquire these basic skills, promoting (face-to-face) interaction between group members and thus enabling groups to accomplish shared goals. [19]: All students have to occupy with a particular task, get feedback from their group mates (or/and the teacher who is free to help with individual problems), and are encouraged to reflect upon their own work [20]. On the other hand, the teacher has to make preinstructional decisions, to explain tasks and cooperative structure, to monitor and intervene, and to assess and evaluate the quantity and quality of student's learning.

In Informatics didactics a lot of research on Cooperative Learning seems to centre on how to create learner-centred web-based learning environments, thus dealing with Computer- or Web-Based Cooperative Learning[5]. But besides the necessary interaction of learners and tools, face-to-face interactions are still important, even in Informatics classes. There, (pure) Cooperative Learning provides a well-tried framework for classroom organization, but there is little evidence that Informatics didacts pay much attention to this topic yet.

5 Searching the internet for Cooperative Learning in combination with Informatics or Computer Science resulted in about 25 percent of the hits being about Cooperative Learingn on a face-to face basis. Additionally, many of these resources deal with learning at university level.

3 Final Remarks

Considering the human factor in Informatics classes is a multi-faceted effort. In doing so, the teacher has to design the learning environment regarding individual learning preferences of the students and pedagogical patterns that allow for efficient cooperation inside the learning group, including the teacher. Therefore, considering the human factor in Informatics classes goes beyond the scope of Informatics didactics mainstream: To care about technical artifacts alone simply does not serve the purpose – knowing about cognitive structures or planning and guiding interaction processes seem to be important aspects as well. This article assembles personal experiences and findings from different branches of science to point at ways how to start with considering the human factor in Informatics. It is time to substitute discussions like "objects first or objects later" by an attitude, that puts the learner in the first place – not only in pedagogy but also in Informatics didactics.

4 Epilogue

Self-reflection on part of the teacher completes the learning-tetrahedron. Self-reflection on part of the teacher adds instructive loops to bygone classroom situations or individual learning experiences. Hence self-reflection on part of the teacher is an important part of the human factor of the learning process. I have tried to pay tribute to that by switching to first person style whenever writing about personal experiences or conclusions from my process of self reflection. Furthermore, this »semi-scientific« style should emphasize that in every learning process there is always a »me« that is learning.

References

1. Blömeke, S.: Universität und Lehrerausbildung. Klinkhardt, Bad Heilbrunn (2002)

2. Antonitsch, P.: Standard Software as Microworld? In: Mittermeir R. (ed.): From Computer Literacy to Informatics Fundamentals. Lecture Notes in Computer Science Vol. 3422, Springer, Berlin-Heidelberg (2005)
3. Antonitsch, P.: Programmieren mit Open Office Basic? Überlegungen zur didaktisch motivierten Adaption von Open Source-Software zur Lernumgebung. In: I.-R. Peters: Informatische Bildung in Theorie und Praxis. Beiträge zur 13. GI-Fachtagung »Informatik und Schule«, INFOS 2009, LOG IN Verlag, Berlin (2009)
4. Reich, K.: Systemisch-konstruktivistische Pädagogik. Beltz, Weinheim und Basel (2005)
5. P. C. Rosnick, J. Clement: Learning without understanding. In: Journal of Mathematical Behaviour, 3 (1) 3-27 (1980)
6. Davis R.B.: Learning Mathematics. The Coginitve Science Approach to Mathematics Education. Ablex Publishing Corporation, Norwood New Jersey (1984)
7. Joni S.-J., Soloway E., Goldman R., Ehrlich K.: Just So Stories: How the Program Got that Bug. ACM SIGCUE Outlook Volume 17, Issue 4 (1983)
8. Bonar J., Soloway E.: Uncovering Principles of Novice Programming. In: Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (1983)
9. Schwank I.: Cognitive Structures and Cognitive Strategies in Algorithmic Thinking. In: Lemut E., du Boulay B., Dettori G. (eds.): Cognitive Models and Intelligent Environments for Learning Programming. Springer, NATO ASI Series F, Vol. 111, pp 249-259, Berlin (1993)
10. Cohors-Fresenborg E., Striethorst A.: Untersuchung individueller Unterschiede in der mentalen Repräsentation von symbolverarbeitenden Regelsystemen. In: Zentralblatt für Didaktik der Mathematik, Vol. 35 No 3, pp94-101 (2003)
11. Schwank I: On Predicative versus Functional Cognitive Structures. In: Schwank I. (ed.): European Research in Mathematics Education, Vol. II, 84-96, Forschungsinstitut für Mathematikdidaktik, Osnabrück (1999)
12. Schwank I.: Analysis of Eye-Movements during Functional versus Predicative Problem Solving. In: Proc. of the 2nd Conference of the European Society of Research in Mathematics Education-Working Group 5 "Mathematical Thinking and Learning as Cognitive Processes, Mariánské Lázně (2001)

13. Cohors-Fresenborg E.: Registermaschine as a Mental Model for Understanding Computer Programming. In: Lemut E., du Boulay B., Dettori G. (eds.): Cognitive Models and Intelligent Environments for Learning Programming. Springer, NATO ASI Series F, Vol. 111, pp 235-248, Berlin (1993)
14. Xu B.Y.: Untersuchung zu prädikativen und funktionalen kognitiven Strukturen chinesischer Kinter bei der Auseinandersetzung mit Grundbegriffen der Programmierung. Schriftenreihe des Forschungsinstitus für Mathematikdidaktik Nr. 25, Osnabrück (1994)
15. Antonitsch P.: Databases as a Tool of General Education. In: Mittermeir R. (ed.): Informatics Education – The Bridge between Using and Understanding Computers. Lecture Notes in Computer Science Vol. 4226, Springer, Berlin-Heidelberg (2006)
16. Antonitsch P.: Datenbanken – (etwas) anders gesehen. In: Schubert S. (ed.): Didaktik der Informatik in Theorie und Praxis. Proceedings of 12. GI-Fachtagung Informatik und Schule, INFOS 2007, Köllen Druck+Verlag GmbH, Bonn (2007)
17. Antonitsch P., Lassernig U., Söllei A.: Lehrarrangements in der Informatiklehrerausbildung. In: Schubert S. (ed.): Didaktik der Informatik in Theorie und Praxis. Proceedings of 12. GI-Fachtagung Informatik und Schule, INFOS 2007, Köllen Druck+Verlag GmbH, Bonn (2007)
18. Johnson R.T., Johnson D.W.: Cooperative Learning Homepage. Available at: http://www.co-operation.org/, access on Aug. 19th 2009
19. Integrating New Technologies into the Methods of Education: Cooperative Learning. Available at: http://www.intime.uni.edu/coop_learning/index.htm, access on Aug. 18th 2009
20. Brüning L., Saum T.: Erfolgreich unterrichten durch Kooperatives Lernen 1. NDS Verlagsgesellschaft, Essen (2008)

Beaver, Kangaroo and Classroom Situations: A Promising Symbiosis

Peter K. Antonitsch[1], Andrea Grossmann[2], and Peter Micheuz[1]

[1] Alpen-Adria Universität Klagenfurt
Institut für Informatiksysteme
{Peter.Antonitsch, Peter.Micheuz}@uni-klu.ac.at

[2] HTBL Mössingerstraße Klagenfurt
andrea.grossmann@htl-klu.at

Abstract. During the last few years »Informatics Beaver« has gained recognition throughout Europe. Addressing secondary school students of all standards, this Informatics problem solving contest was designed on the model of the »Mathematics Kangaroo«. This article points at similarities of and differences between these competitions, considers the relationship between the two contests and the corresponding subject-matters and suggests ways to enrich classroom teaching by borrowing from problem solving contests.

1 Problem Solving Competitions

Problem Solving Competitions in the field of formal and natural sciences have a long tradition. The first International Mathematical Olympiad was held in Romania in 1959, followed by the International Physics Olympiad (Poland, 1967) and the International Chemistry Olympiad (Czechoslovakia, 1968). The International Olympiad in Informatics is a rather young high level contest and was conducted in Bulgaria in 1989 for the first time.

Apart from these tournaments for the rather talented, there exist problem solving contests which are intended for students of all

standards, like the Math Kangaroo or the Informatics Beaver. As these contests influence the students' notion of the corresponding subject-matter, they are of particular didactic interest.

1.1 Of Kangaroos and Beavers – Some Historical Remarks

The roots of the Math Kangaroo can be traced back to the late seventies of the 20th century. It was in 1978[1], when the Australian Mathematics Competition was introduced in Australian schools on a nationwide scale [3]. Originally consisting of 30 multiple choice tasks[2], this contest soon spread over Australia and the South Pacific region and inspired two French Mathematics teachers to organize a similar contest in France in 1991, named »Kangourou des Mathématiques« to give honour to the Australian inventors [4]. Within three years the Math Kangaroo became an international contest, attracting more than 5 million participants in 2008. Quite similar to its Australian forerunner, this contest aims at promoting creative thinking and enjoyment in mathematical reasoning by means of multiple choice based mathematical problems allowing for a quick solution. Some of these problems rely on knowledge provided in Mathematics classes, while others require a deeper understanding of the mathematical background or general problem solving strategies.

The main principles of the Math Kangaroo were borrowed by V. Dagiene to establish a tournament "to promote interest in Information and Communication Technologies (ICT) as well as Informatics as foundational science of this area to all school students" ([5], [6]). The »Bebras International Contest on Informatics and Computer Literacy« (short: »Informatics Beaver«) was conducted in Lithuania in 2004 for the first time and has turned an international contest since, with almost 100.000 participants from 10 (European) countries in 2008.

1 There exist even older contests, like the UNSW School of Mathematics and Statistics Competition (starting in 1962) [1], or the Canadian Mathematics Contest (dating back to 1963) [2].

2 The format of the contest was changed several times. By now there are 25 multiple choice tasks and 5 single answer tasks, requiring integer answers between 0 and 299.

1.2 Classifications of Beaver- and Kangaroo-Tasks

Informatics and Mathematics differ in their scientific approach to the world. Modelling is a basic concept of both sciences, but it is Informatics that includes the technological realization of models, caring about aspects like the interface between man and machine or economical issues. While Mathematics is a pure science where the model per se is of value, Informatics is obliged to applicability and realizability of the model as well [7]. This difference between the two sciences affects school curricula of Mathematics and Informatics: There are various cross-connections and interrelations, but the learning goals are different.

The Mathematics Kangaroo and Informatics Beaver have much in common: They share common roots, both of them focus on problem solving and both contests contain tasks that originate in the culture of the corresponding subject-matter at school. Consequently, the set of Beaver-tasks and the set of Kangaroo-tasks should have a non-empty intersection, but the differences of the two sets have to be non-empty, too! A look at the classification of tasks proves this right:

In [8], V. Dagiene and G. Futschek present a list of six task types relevant for tasks of the Informatics Beaver:

- *information comprehension* (including representation, coding and encryption)
- *algorithmic thinking* (including programming aspects)
- *using computer systems* (general principles of standard software)
- *structures, patterns and arrangements* (combinatorics and discrete structures)
- *logical puzzles and games*
- *ICT and society* (including social, ethical, cultural, international and legal issues)

On the other hand, there is no official list of task types for the Mathematics Kangaroo. As a starting point we use a classification provided in [9], [10] to cluster existing Kangaroo tasks:

- *numbers and computations* (including fractions, primes and percentages)
- *equations, inequations and functions* (including linear and non-linear equations, diophantic equations, systems of equations and graphs of functions)
- *combinatorics* (with numbers and shapes)
- *geometry* (including plane geometry and solid geometry)
- *logic, cryptic and magic puzzles.*

The list misses a category *algorithms*, although algorithmic thinking has influenced the development of mathematics throughout history ([11], pp 185). A closer look unveils algorithms/algorithmic thinking being orthogonal to the Kangaroo-categories: Algorithms are the heart of computations and geometric constructions of any kind. Therefore we consider splitting the first category valid, turning *numbers and computations* into *dealing with numbers* and *algorithms* (see Fig. 1).

1.3 Changing the Point of View: Specific, Interchangeable and Related Tasks

Fig.1 suggests that some types of tasks are characteristic either to the Beaver-domain or to the Kangaroo-domain of problem solving. We call this kind of tasks (Beaver- or Kangaroo-) *specific tasks.*

But most of the categories belong to both domains (at least partly). Furthermore, *structures, patterns and arrangements* to some extent corresponds to *combinatorics*, while *representation of information* and *dealing with numbers* share coding as a common ground. And, of course, *algorithmic thinking* and *algorithms* seem to be mere synonyms. Consequently there have to be tasks that could

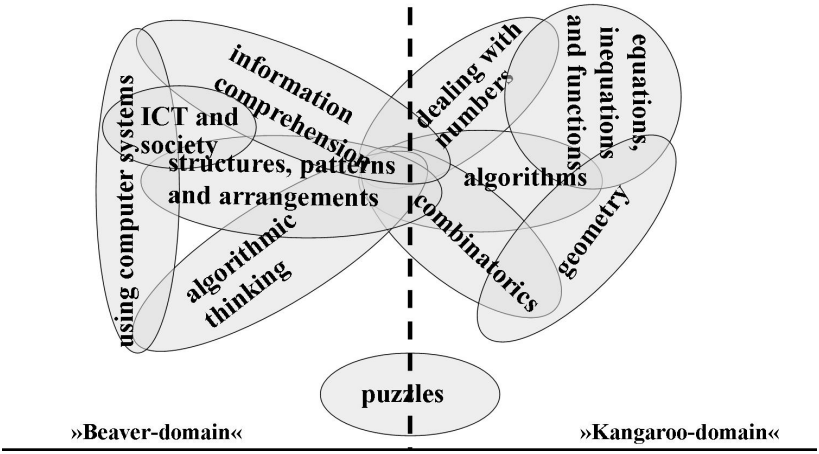


Fig. 1. Task-categories map of the Informatics Beaver- and Math Kangaroo-domains – the categories overlap to indicate that some tasks can not be related to one category alone. Puzzles are kept separate because they play a minor role in classroom situations (at least in Europe).

be items of Informatics Beaver and Math Kangaroo as well. We call this kind of tasks *interchangeable tasks*.

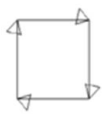
But we have to be careful! Looking at tasks concerning algorithms/algorithmic thinking we notice that seeming synonymy can denote different aspects of a category as well: In 2007 one of the (more difficult) Kangaroo-tasks for the grade 9 and 10 reads as follows ([10], p 44):

> Let a and b be the solutions of the quadratic equation $x^2 - 3x + 1 = 0$. What is the value of $a^3 + b^3$?
>
> **A)** 12 **B)** 14 **C)** 16 **D)** 18 **E)** 24

To find the correct solution (without guessing), the contestant might use the *known* algorithm for solving quadratic equations and calculate the sum of the third powers, or make use of Vieta's formulas, the contestant is supposed to *be acquainted with*.

Fig. 2 depicts a Beaver-task of comparable degree of difficulty for the same grades. Here, the solution of the problem does not necessarily depend on prior knowledge about certain algorithms or the Logo turtle, but can be found by re-inventing the sequence of commands necessary to draw the given picture.

Logo Turtle may perform the following commands:
`forward n` – to move forward drawing a line of n steps long;
`right` α – to turn right making an angle of α degrees;
`left` α – to turn left making an angle of α degrees;
`repeat 5 [forward 30 right 30]` – to move forward drawing a line of 30 steps long and to turn right making an angle of 30 degrees; these actions are repeated for 5 times.
The *turtle* looks up at the beginning.



The figure was drawn using the following commands:

`repeat 4 [forward 100 left … repeat 3 [forward 20 left 120] right … left 90]`

Which of the given degrees have been written instead of the suspension points?

A) 30 B) 45 C) 60 D) 90

Fig. 2 Beaver-task of the category *algorithmic thinking* (2005, grades 9 and 10; [12]).

Obviously, both tasks concern algorithms, but at different levels of action: Solving the Kangaroo-task means to *use* algorithms already known, while solving the Beaver-task means to *create* (or to *process*) a *new* algorithm. Furthermore, dealing with these problems entails to take different views: When applying ready-made algorithms the problem-solver remains outside the problem. On the other hand, finding an algorithm to have something done by an actor means to view the problem with the eyes of this actor, in other words: to step into the problem [13].

Tasks that share a common topic but require different levels of action and/or different approaches to be solved shall be called *related tasks*. The same goes for tasks that take different points of view on a common topic.

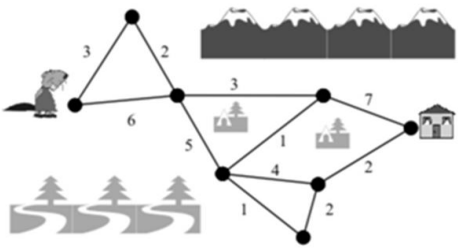
We provide some examples to clarify the definitions given above (Fig. 3 and Fig. 4):

Informatics Beaver 2008 (grades 5 to 7): Fastest Way

Beaver wants to go home as fast as possible. In the drawing you see minutes needed to come from one point to other. What is the best possible time?

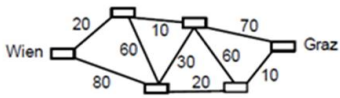
Choose correct answer:

A) 17 minutes
B) 15 minutes
C) 14 minutes
D) 16 minutes



Math Kangaroo 2006 (grades 3 and 4): Cheapest Ticket

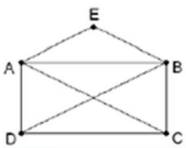
The numbers along the lines give the price of a ticket between two cities. Petra wants to travel from Wien to Graz at lowest price possible. What is the minimum amount she has to pay?



A) 80 B) 90 C) 100 D) 110 E) 180

Informatics Beaver 2005 (grades 11 to 13): Eulerian Path

A graph is a figure consiting of points (named vertices) and segments connecting each pair of vertices (named edges). The sequence of segments, which connects any two vertices of the graph, is named the path. If the path consists of all edges of the graph repeated only once, such path is called the Eulerian path. From which vertex of the graph should you start in order to draw the given Eulerian path?



A) D or C B) A or B C) E D) It's impossible to draw the Eulerian path

Fig. 3. Three tasks concerning graph theory (category *structures, patterns and arrangements* resp. *combinatorics with shapes*). The two Beaver tasks represent *related tasks*, while "Fastest Way" and "Cheapest Ticket" are *interchangeable* Beaver- and Kangaroo-*tasks*.

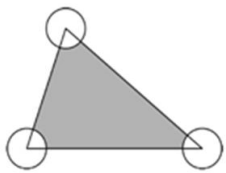
Informatics Beaver 2008 (grades 5 to 7): Formatting

Suppose you are writing a letter using text processing software and you want to centre some text in it. What is probably the best way to do the formatting?

A) To use necessary amount of spaces **B)** To use a few tabs
C) To use the alignment function **D)** To shift document margins

Math Kangaroo 2009 (grades 9 and 10): Geometry

The area of the shown triangle equals 80m². Each circle has a radius of 2m and its centre is in one of the vertices of the triangles. What is the area of the grey shaded region (in m²)?



A) 76 **B)** 80 - 2π **C)** 40 - 4π **D)** 80 - π **E)** 78π

Fig. 4. *Specific* tasks from the categories *using computer systems* and *geometry*.

1.4 A Concluding Interlude

In spite of their common roots and goals, Mathematics Kangaroo and Informatics Beaver have become well-distinguishable problem-solving contests that complement each other. The common stock of interchangeable tasks serves as a link between mathematical reasoning and strategies to tackle problems in the field of Informatics. Therefore, these two competitions cover a wide spectrum of problems and, what counts even more, they manage to motivate young people to engage in problem-solving activities. Judging from personal experience, competitions manage to motivate much better than typical classroom situations in Mathematics or Informatics.

We have to ask: What makes the difference? And: Can we borrow from problem-solving contests for students of all standards in order to enhance learning in class?

2 Kangaroo, Beaver and the Corresponding Subject-Matters

2.1 The Core of Subject-Matters: Fundamental Ideas and Standards

The concept of fundamental ideas was introduced into didactics in 1960 by Jerome S. Bruner who asked: "What are the implications of emphasizing the structure of a subject, be it mathematics or history – emphasizing it in a way that seeks to give a student as quickly as possible a sense of the fundamental ideas of a discipline?" ([14], p. 3). Fundamental ideas of a discipline provide an abstract framework for the corresponding subject matter and have become the basis for educational standards that focus on the interface between the subject and teaching/learning practice. When looking for connections between Mathematics Kangaroo and Informatics Beaver on one hand and the subject matters Mathematics and Informatics on the other, "the standards" seem to be an appropriate starting point.

2.2 Kangaroo Categories and Mathematics Standards

In 2000 the National Council of Teachers of Mathematics (NCTM) published »Principles and Standards for School Mathematics« which has become the mother of all standards in the field of Mathematics[3] and Informatics. The NCTM-collection lists five content standards and five process standards, each of which "consists of two to four specific goals that apply across all the grades" [15]. The content standards are[4] *number and operations*, *algebra*, *geometry*, *measurement*, and *data analysis and probability*, accompanied by the process standards *problem solving*, *reasoning and proof*, *communication*, *connections*, and *representation*.

3 For instance, the Austrian content-standards for the subject matter Mathematics are akin to the NCTM.standards, with the single exception that in Austria »analysis« is a content standard for grades 9 to 12/13 while the NCTM-standards miss this branch of Mathematics completely. (see [16], [17] for a synopsis of current Austrian Math-standards).

4 see [15] for complete information.

Comparing the content standards with the original list of Kangaroo-categories taken form [9], [10] we see little difference: The category *equations, inequations, functions, and graphs* can be subsumed under the content standard *algebra*-and the category *puzzles* can be related to the process standards *reasoning and proof* and *problem solving*[5], where the NCTM-standard reads as follows: "[...] Students need to develop a range of strategies for solving problems, such as using diagrams, looking for patterns, or trying special values or cases. [...] Teachers play an important role in developing students' problem-solving dispositions. They must choose problems that engage students [...]." [15]. Puzzles are a good choice of problems to engage students. Obviously, Kangaroo-tasks are quite compatible with standards for the subject matter Mathematics.

2.3 Beaver Categories and Informatics Standards

After the model of the NCTM-standards for Mathematics a working group of German-speaking Informatics didacts elaborated a recommendation for Informatics standards for grades 5 to 10 [18]. Published in 2008, this recommendation – like its forerunner – lists five content standards and five process standards. There are *information and data*, *algorithms*, *language and automat*, *Informatics systems,* and *Informatics, man, and society* for the content standards[6], and *modelling and implementing*, *explaining and appraising*, *structuring and linking*, *communication and cooperation*, and *representation and interpretation* for the process standards.

Quite similar to the Kangaroo-categories, we find most of the Beaver-categories match the Informatics content standards: *Information comprehension* and *algorithmic thinking* correspond to *information and data* and *algorithms*, touching even aspects of the process standard *modelling and implementing*. Furthermore, *using computer systems* and *ICT and society* represent subsets of *Informatics systems* and *Informatics, man, and society*, while

5 In Austria »problem solving« is a process standard for grade 4 only!

6 see [18] for complete information

structures, patterns and arrangements intentionally touches aspects of *language and automata*.

And, again, we miss a notion of *logic puzzles and games* within the list of standards[7],

2.4 What makes the difference?

We have learned that there is hardly any difference between task-categories of problem solving contests and educational standards of corresponding subject matters. In other words: It is very unlikely that the motivation of problem-solving contestants is (solely) triggered by the content of the tasks. But the absence of puzzles and games might give a clue.

Puzzles are considered unstructured problems that have to be framed by the solver before solving [19]. Furthermore, games are considered organized play structured by the six key elements ([20], p118) *rules*, *goals and objectives*, *outcomes and feedback*, *conflict/competition/challenge/opposition*, *interaction*, and *story or representation*.

Informatics Beaver and Math Kangaroo incorporate most of these characteristics: The contestants' goal is to reach as many points as possible and they get feedback about their performance by means of result lists. Many tasks are put into context, some of them are challenging, and some of the tasks are even unstructured at first sight. But are the contests play in the sense of "[...] play is a free activity that is consciously outside of ordinary life". [...] play is utterly and absorbing" ([20], p 112)? We think the answer is "yes": Students are free to enter the contest – those who refuse to participate are free to skip all the tasks without negative consequences[8]. Moreover, Kangaroo and Beaver tasks have the quality of "haunting thoughts": The tasks stem from the mindset of

7 The situation is even worse than with Kangaroo-categories and NCTM-standards: The Informatics process standards do not include items like *problem solving* and *reasoning* explicitly (although reasoning might be part of *communication and cooperation*)!

8 See [6] for an outline of Beaver contest regulations; the rules for Math Kangaroo are alike.

Informatics/Mathematics alone and each of the tasks covers just a narrow domain inside Informatics/Mathematics. Therefore, most of the problems are easy to survey and quite often the combination of textual and graphical representation contains a first notion of the solution. To have an idea where to begin is a necessary prerequisite and a strong motive to take up – one begins, one tries this and that, gets involved (almost "absorbed") and feels a touch of "flow" when the solution is puzzled out at last. Furthermore, the multiple-choice style of the contests helps to check whether ones own solution might be correct[9].

Consequently, when students participate in the Informatics Beaver or Math Kangaroo they *actively* play a kind of game. This really makes a difference to common classroom situations. M. Prensky states that "the majority of our education has become a series of informational or logical presentations or readings, followed by some sort of quiz or examination. [...] it bores the [learners] to tears" [20, p71]. Learners who are used to be active at multi-medial private entertainment become mere listeners at school, supposedly most of them, most of the time. But with problem-solving contests the problem-solvers have to be active! They have to think, they have to try by themselves, they can tinker at the problems and they even can err without consequences for their career at school!

That makes another good point: The Informatics Beaver and Math Kangaroo are no part of daily routine at school. These contests are something special *outside ordinary life*, so from the students' viewpoint the contests are really sort of (organized) play. "People enjoy difficult tasks more when presented as play rather than work, and their minds wander less." [20, p 115]

Yet there is another, a social aspect that must not be neglected: Although the »Informatics Beaver Games« and the »Math Kangaroo Games« are outside school, being a good »player« pays off in school:

9 Of course we are aware that the provided possibilites for correct answers are used the other way around as well (and, supposedly, even more often): Contestants might simply choose the answer that is correct most likely, sometimes without any reasoning.

It is quite common that the best »beavers« or »kangaroos« (of a school, a district, a country) get prizes and are honoured within a ceremonial act. The motivating power of gaining recognition inside one's social group should not to be underestimated!

3 Outlook and Résumé

Informatics Beaver and Math Kangaroo are similar, but not the same. Further research might focus on distributing a greater sample of tasks to the categories of interchangeable, related and specific tasks, possibly arriving at the conclusion that a refined categorization will be more appropriate.

Investigating the influences of Informatics Beaver and/or Math Kangaroo on »ordinary« Informatics/Mathematics lessons outlines another promising field of research. Two aspects might be of specific interest:

- It has been noticed that problem-solving competitions can rouse the learner's interest in the corresponding subject matter. Questions like: "Can we discuss the solution to that particular problem?" or: "These problems were so different from what we deal with in our Informatics/Mathematics lessons! Does this really belong to Informatics/Mathematics?" point at the potentiality of the contests to widen the learner's horizon and to continue engaging the learners in solving problems within or even beyond the scope of a particular curriculum.
- We witness that inspired teachers already use Kangaroo-like problems successfully to motivate their students in Mathematics lessons, especially in grades 5 to 8, but until now these individual advances seem to be beyond the scope of scientific research.

There are further questions concerning the adaptability of contest-style tasks for daily learning, like:

- What about the process-level of Beaver- and Kangaroo-tasks? Are there specific Beaver-skills and/or Kangaroo-skills, and if there are, do they correspond to the process standards listed above?

- Can Beaver- and Kangaroo-tasks be expanded so that problem-solvers not only have to choose the correct solution but also have to explain how they found it? Or will the inevitable shift from multiple choice tasks to short answer questions diminish the motivating effect for the majority of students?
- Can we exploit the knowledge about the power of games and the importance of social ties as well to enhance learning in class? In [21] T. Verhoeff states his personal opinion, "that competitions have much to offer in education". We quite agree with him, but does this apply to (imaginable) micro scale contests as well? Do intra-school (or even intra-class) contests change the student's general attitude towards learning? Might these contests help to set up a competitive or rather a cooperative climate in school/class? And finally: What does it take to establish a common spirit of esteem for (academic) success in schools?

All of these questions point at an inspiring, a vital, and a very promising symbiosis between school and problem-solving contests designed for all students. Most answers have still to be given, though.

References

1. http://www.maths.unsw.edu.au/highschool/unsw/highcomps.html
2. http://cemc.math.uwaterloo.ca/contests/contests.html
3. http://www.amt.edu.au/amcfact.html
4. http://www.mathkang.org
5. http://www.bebras.org
6. Dagiene V.: Competition in information technology - learning in an attractive way. Submission to the workshop "Perspectives on Computer Science Competitions for (High School) Students" at Schloss Dagstuhl, Germany, 2006.
   http://www.bwinf.de/competition-workshop/papers.html
7. Mittermeir R.: Mathematik und Informatik. Halbbrüder oder Geschwister unterschiedlichen Geschlechts. In: Kadunz G. et al. (eds.): Mathematische Bildung und neue Technologien. Teubner, Stuttgart (1998)
8. Dagiene V., Futschek G.: Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks. In: Mittermeir R., Sysło (eds.): Informatics Education – Supporting Computational Thinking.

Lecture Notes in Computer Science Vol. 5090, Springer, Berlin-Heidelberg (2008)
9. Noak M., Geretschläger R., Stocker H. (eds.): Mathe mit dem Känguru. Die schönsten Aufgaben von 1995 bis 2005. Carl Hanser Verlag, München (2008)
10. Noak M., Geretschläger R., Stocker H. (eds.): Mathe mit dem Känguru. Die schönsten Aufgaben von 2006 bis 2008. Carl Hanser Verlag, München (2009)
11. Davis P.J., Hersh R.: The Mathematical Experience. Birkhäuser Verlag, Basel (1982)
12. CD.version of the 2004- and 2005-Beaver tasks, downloadable at: http://www.emokykla.lt/bebras/download/bebras.zip
13. Duchâteau C.: From "DOING IT..." to "HAVING IT DONE BY...": the heart of programming. Some didactical thoughts. In: Preproceedings NATO ARW Cognitive Models and Intelligent Environments for Learning Programming. S. Margherita Ligure, Genova 1992
14. Bruner J.S.: The Process of Education. Harvard University Press, Cambridge Mass. (1960)
15. NCTM Standards Online; available at: http://standards.nctm.org/document/
16. Bildungsstandards und Kompetenzmodelle für die 4. und 8. Schulstufe – Austrian educational standards and competency models for grades 4 and 8 (in German): http://www.bmukk.gv.at/medienpool/17534/bgbl_ii_nr_1_2009_anlage.pdf.
17. Bildungsstandards Angewandte Mathematik BHS – Austrian educational standards for applied Mathematics in vocational schools (in German); available at: http://www.berufsbildendeschulen.at/fileadmin/content/bbs/AGBroschueren/ PilotbroschuereMathe-jan09.pdf.
18. Arbeitskreis »Bildungsstandards«: Bildungsstandards Informatik für die Sekundarstufe 1 – Educational standards Informatics for grades 5 to 10 (in German). Beilage zu LOG IN, 28. Jg. (2008), Heft Nr. 150/151.
19. Michalewicz Z.., Michalewicz M.: Puzzle-Based Learning. In: Proceedings of the 2007 AaeE Conference, Melbourne, published as a CD-ROM available online at http://www.cs.mu.oz.au/aaee2007/papers/paper_25.pdf

20. Prensky M.: Digital Game-Based Learning. Paragon House, St. Paul MN (2007)
21. Verhoeff T.: The Role of Competitions in Education. Presented at Future World: Educating for the 21st Century, a conference and exhibition at IOI, available at:
http://olympiads.win.tue.nl/ioi/ioi97/ffutwrld/competit.pdf

All links have been accessed on Aug. 24th 2009

Maturity Exam in Programming for a High School: Tasks Developing and Evaluation Approaches

Jonas Blonskis[1] and Valentina Dagiene[2],

[1]Kaunas University of Technology
Sukileliu str. 112–34, Kaunas, LT-49240 Lithuania

jonas.blonskis@ktu.lt

[2]Vilnius University, Faculty of Mathematics and Informatics
Naugarduko str. 24, Vilnius, LT-03225, Lithuania

dagiene@ktl.mii.lt

Abstract. Two models of maturity exams in information technologies and computer science have been applied in Lithuania at high schools [2]. The first one is intended to evaluate students' competencies in information technologies. The other one is focused on programming skills and is intended for promoting the professional studies of informatics in higher education. The first national exam in information technology was launched in 2006. The exam consists of a set of tests (questions) and two programming tasks to write programs named as practical tasks. The goal of the practical tasks is to develop programs for given tasks. Developing programs is the most important part as well as one of the most difficult tasks for students. The paper deals with objectives, tasks, and evaluation of the maturity exam in programming for high school students.

Keywords: Exam in informatics, programming, task developing, writing programs, data structures.

1 Introduction

Informatics (information technologies) as a separate subject was taught in Lithuanian high (secondary) schools. To establish the maturity exam in informatics was quite a purposeful process. Discussion on the maturity exam in informatics has been presented in the second ISSEP conference and later [2; 3; 4].

Students can choose a module of programming basics in the 11-12th form. The objective of this module is to familiarize students with programming constructions, encourage them to choose informatics studies in universities and become programmers. In this module, students are familiarized with solution methods of simple tasks, data structures and algorithm modification [5].

Programming skills hold quite a big part of informatics studies. Informatics study programs are being improved and expanded. Students who are familiarized with programming concepts and who want to program are required. National and international informatics olympiads are intended basically for creating algorithms and developing algorithmic thinking, moreover, they encompass only a small part of the most talented students. Researches show that the most talented programmers have written their programs at the age of 11-13. Therefore, the need to evaluate the acquired knowledge and skills with one accord appeared in Lithuania. Since 2006 the national exams in information technology and programming (in abbreviation, programming) have been carried out and their results are a part of competition grade when informatics or contiguous studies in higher education are chosen. Those who pass the national exam in programming successfully, have wider possibilities to become students of the desired trend of studies, *i.e.* informatics. At the same time it is a test whether a student is apt for studying informatics: there are quite many first-year students who quit their studies since they find programming a hardly understandable and uninviting occupation for themselves.

The exam may be approached in two ways: on the one hand, it is the evaluation of the results achieved by a student; on the other

hand, it could heighten the motivation to learn. Both must be considered when planning the exam. The exam should be prepared so that it measured the competences needed for studying in universities.

2 Objectives and Scopes of the Exam

Developing a national recognized exam is a responsible job. The exam not only evaluates students' knowledge and makes way for the students aligned to enter universities, but also teaches younger students and teachers [2].

An exam is usually the final method used to evaluate the learning levels reached by students as well as the quantitative expression of achievements according to the educational standards. The main function of an exam is the evaluation of learning results [8]. The content of the exam is closely related to the subject's curriculum. The proper selection of the exam's goals, and the emphasis (or lack thereof) on one or another aspect of the subject, have a strong impact on the quality and content of learning as well as on the students' motivation to learn the discipline. Lithuanian teachers and students pay great attention to exams and therefore this situation should be exploited. By creating the content of the exam, a double goal could be achieved: to evaluate the students' knowledge and to encourage a student to cultivate his or her skills in the chosen field.

The goal of the programming exam is to encourage skilful students to engage in software design and thus to develop their skills. Programming is one of the most essential intellectual resources of our country. However, programming is not an easy job: it requires much effort and certain specific skills. Programming is a creative process that encourages thinking and the integration of knowledge from various fields. It helps form a professional attitude to application programs and prompts an impact on their implementation in a more efficient way. It is assumed that the programming exam will help some students to become interested in this activity and they will pursue programming as a profession.

When developing the content, the recommendations of world experts were taken into consideration [8]. The maturity programming exam is based on the optional module of the basics of programming [5; 6] which consists of four parts: 1) introduction – basic elements of programming; 2) data structures; 3) developing algorithms; 4) testing and debugging programs.

The exam consists of two parts: the larger part (75%) is allocated to programming, while the rest part (25%) concerns the issues of computer literacy. The programming part consists of a test (25%) and two practical tasks (50%). The aim of the programming test is to examine the level of students' knowledge and understanding of the tools required in programming (elements of the programming language, data types and structures, control structures, basic algorithms). The national exam focuses on: knowledge and understanding – 30%, skills – 30%, and problem solving – 40%. The problems are oriented towards the selection of data structures and application of basic algorithms to work with the created data structures.

The practical part has two tasks, – students have to write programs for the given problems. The main aim is to examine the students' ability to master the stages of programming activities independently.

The first tasks are intended to examine the students' abilities to write programs of the difficulty described in educational standards. The abilities of students to use the array data type for work with integers, to realize the algorithms for work with data structures as well as the abilities to manage with input and output in text files are examined.

The second tasks are intended to examine the students' understanding and abilities of implementation of the record data type. The core of the task is to develop the appropriate structures of records together with arrays. The abilities to input data from the text file to arrays containing the elements of record type, to perform operations by implementing the analyzed algorithms, and to present the results in a text file are being examined. The operations are to be

performed only with numerical values. The curriculum does not mean operations with character strings, only reading and derivation of such strings are applied.

Further, we will deal with the practical tasks of the exam: their aims, complexity, evaluation, students' solutions developed, approaches and indentified problems. The national exam curriculum and tasks are presented in National Examination Centre of the Republic of Lithuania (URL: http://www.nec.lt) [5].

With the participation of UK experts, the exam analysis, was made under the project [8], while its results and conclusions on how to prepare the exam were checked by means of a pilot exam in December 2007. A great attention was paid to the development of practical tasks. They must meet three main requirements: to test knowledge, practical skills, and the ability to solve problems. The first two points are not difficult to evaluate and possible to formalize, while the evaluation of the third one, the creative aspect, is problematic.

The task meets the exam requirements if there is an opportunity to choose from the following ways of solving it: 1) a method of solution; 2) data structures; 3) realization of an algorithm. The student's purpose is to find a suitable compatibility of the whole (Fig. 1).

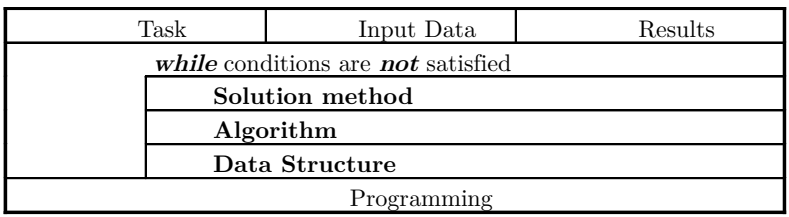


Fig 1. Algorithm of the main component parts of task analysis

If there are more than one mathematical solutions of the task, there may be more than one algorithm solution to each of them. The student has to evaluate possible variants on his level of knowledge in respect of data structures known to him. Only properly combined

those three aspects indicate the complexity and size of the future program, which is very important since the exam time is limited. The fact that realization of an algorithm depends on the chosen data structures should be considered additionally.

The task consists of the main task formulation, initial data, and requirements to results. Data structures are created in order to keep data and results conveniently. At the same time, it is possible that data structures are created separately for data and results. It is essential to evaluate the necessity to save intermediate results during the proceedings of task solution (e.g., it is enough to save only the meanings of the last two sequence members while searching for the nth Fibonacci sequence member). Moreover, while creating data structures, it is necessary to consider a convenient way to perform actions in them.

3 To Solve a Task – Write a Program

Students often understand the syntax of statements of a programming language (e.g., repetition, procedure), but they do not know when and how to use them. Using Bloom's Taxonomy [9] to analyze this problem, our students have difficulty moving from knowledge recall (level 1) and knowledge comprehension (level 2) to knowledge application (level 3). That is, students understand the syntax of basic statements, however they cannot reach the level of applying the knowledge and writing programs to solve problems.

We understand that an exam is not the best way of teaching students, – it seems to be late. We have noticed something different. The students who intend to take the programming exam choose the programming module a year ago and try to follow the exam model while studying. In other words, if a lot of attention is paid to writing programs, if there are many tasks of algorithms and data structure selection in the exam, the students pay much attention to the mentioned points while learning. Therefore, the exam performs an educational function.

For this reason, after every exam a quantitative analysis, which emphasizes weakly acquired knowledge, the most complicated fields of knowledge and the exam task suitability for ranking students, is made. In addition, a qualitative analysis, which looks at solution methods, data structures chosen by students and their ability to develop algorithms, is made as well. Both of them show strong and weak points and provide most rational solutions. Typical mistakes are analyzed and the means to avoid them are indicated. Therefore, this is the main educational means for those who prepare for the exam and it is a source for teachers to raise their qualification.

Teachers act as program evaluators. The evaluation is performed in two aspects: an automatic test to determine the correctness of program work and a visual testing which aims to evaluate the effectiveness of the written program. To this end, an interactive evaluation system that is related to a data base of collecting and processing the evaluation results has been created.

During the process of evaluation, different program variants are analyzed and discussed, the effectiveness of chosen data structures and algorithms is evaluated. That is the main source for preparing the methodological educational material which is used in seminars and teachings of various levels.

4 Peculiarities of Practical Tasks

Solution of tasks displays students' abilities to make decisions and implement them. It consists of four steps: 1) selection of the solution method, 2) algorithm formation, 3) data structures development, 4) algorithm modification for proper work.

Selection of the solution method is important when there exist several possible task solutions exist and most acceptable should be chosen. The process of algorithmization is one of the most important ones because it determines not only the size of a program, but also the complexity, while task analysis, separation of initial data and results govern the complexity of data structures.

During the process of developing data structures, it is essential to consider a convenient way to do both: to save initial data and calculation results and to perform actions in order to realize the created algorithm. At all the stages of program creation, a student's ability to be independent and to refuse pattern solutions is the main factor. In this way, the student's ability to look for indirect ways of algorithmization is disclosed.

Giving a task, it is most important to evaluate the student's ability to create appropriate data structures considering the selected solution algorithm, and to modify the algorithm to work with data in those structures. An opportunity to select one of several alternative solution methods, even though ambiguous, must be allowed in the task. However, it is much more difficult to determine the rating scale.

When evaluating the students' programs, two aspects are distinguished: program correctness and program rationality. The fist one is assessed precisely according to a point rating scale prepared beforehand. The evaluation consists of two steps: testing and program text review. Whereas the second evaluation is rather subjective, because there are no means to specifically describe the most rational solution from all the possible ones. This is why the work is evaluated by two independent assessors. If a great difference between the evaluations occurs (e.g., the difference of 5%), the head assessor evaluates the work.

Example: Collection. A full collection of chocolate egg toys consists of 100 toys. Each toy has its number in the collection. Lina and Jurgis have been collecting the toys for the whole summer. At the beginning of September, they decided to exchange toys so that both collections were supplemented with the new toys. Only those toys may be offered for exchange that appear in children's collection more than once. However, the toys that are owned by the friend do not suit for exchange. Several identical toys also cannot be offered to exchange [7]. Write a program that would select the numbers of toys, chosen by Lina, to exchange with Jurgis, and the numbers of toys, chosen by Jurgis, to exchange with Lina, would comprise a list of toy numbers in a combined collection. The numbers cannot be repeated.

Date. The text file has three lines. The first line has n amount of Lina's toys, and m amount of Jurgis' toys. In the second line, there are toy numbers of Lina's collection, while in the third line, there are toy numbers of Jurgis' collection (Table 1).

Results. Print the toy numbers offered for exchange in increasing order in the text file. Print the toy numbers, offered by Lina for exchange, in the first line, and the toy numbers, offered by Jurgis, in the second line. If they both have no toys for exchange, then print 0 (zero) in the appropriate line. In the third line, print a list of toy numbers in Lina and Jurgis' combined collection in increasing order. The numbers cannot be repeated (Table 1).

Table 1. Example of data and results

Input file	Output file
8 12 5 6 6 9 14 6 8 16 5 12 6 7 13 7 9 10 12 5 16	0 7 12 5 6 7 8 9 10 12 13 14 16

Most of the students chose a consistent task solution method. They wrote down the initial data (toy numbers) into two separate arrays and ranked in increasing toy number order. Then they formed two arrays where they wrote down each person's toy numbers for exchange. Here these numbers which repeated in the data array at least twice were written down. Afterwards, these numbers which were present in the data list of another person were eliminated from the data array, and this is data lists for exchange. Then the students formed a list of the combined collection in the following way: they wrote down the meanings of both initial arrays into a new array, ranked the meanings of the array and eliminated the repeated toy numbers. That is a simple sequence of actions which required even five arrays. At the same time, three algorithms had to be used, such as: ranking, searching for repeated meanings, and eliminating the meanings from the array.

A more rational solution can be presented by saving data in two arrays A (Lina's) and B (Jurgis'): the array element index marked the toy number, while the meaning of indexed variable marked the amount of a particular toy number (to read data from such a file the

following action was used: A[no] := A[no] + 1, when no – toy number, A – Lina's collection list; analogous B array for Jurgis' collection). An example of writing data into arrays is shown in Table 2.

Table 2. An example of writing data into arrays

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	0	0	0	0	1	3	0	1	1	0	0	0	0	1	0	1	0	0
B	0	0	0	0	2	1	2	0	1	1	0	2	1	0	0	1	0	0

For example, if A[k] = 0, it means that Lina does not have the toy number k. Algorithm ranking and meaning elimination became unnecessary. The toys with the meaning higher than 1, are for exchange. The toy number suitable for Lina's exchange is the one which is A[no] > 1 ir B[no] = 0. The list of Jurgis' exchange is formed in a parallel way. The list of combined collection is made up in the same way as that of personal collections. Then, the numbers which match the meaning unequal to zero in the array, have to be printed. This method is simpler and the program is much shorter. Similar solutions are also possible.

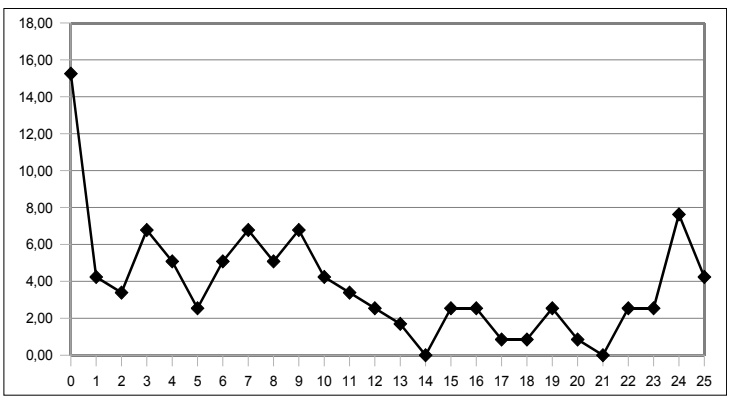


Fig 2. Task evaluation results

This task distinguishes more skilful students from the beginners and does not constrict the creativity and independence in selecting a solution. However, evaluation of such programs is a difficult process since very diverse solution methods are used and the size of programs differs. The evaluation results are presented in Fig. 2.

The task was solved by 118 students; its difficulty – 27% and resolution – 69%. The numerical value of task difficulty is a percentage of all the points get by students and the amount of points theoretically possible to collect. According to the statistical test theory, the best tasks are of 50% difficulty, very easy ones $> 80\%$, and very difficult ones < 20 %.

The task resolution shows how a separate task distinguishes the best and the worst students. Task resolution is a difference between difficulties of 10% of the best students who passed the exam and 10% of the worst students. If the task is very easy and almost all the students, the best and the worst ones, completed it successfully, the resolution of such a task is small. A very difficult task may also have a similar resolution. As it follows, a negative meaning of the resolution shows that worse students gained more points for that question than the better ones, and this is a feature of a very poor question. According to the statistical test theory, proper tasks are the ones with resolution of 40-50% and very good tasks have resolution of $>= 60\%$.

The results of the first practical tasks analysis given in the previous exams are illustrated in Table 3.

Table 3. The difficulty and resolution of the first practical task. Students' abilities to modify algorithms according to the particular data structures.

Year	**Students**	**Difficulty**,%	**Resolution**,%
2006	1164	48,32	91,34
2007	873	49,24	68,65
Pilot 2007	119	27	69
2008	832	42.3	67.2
2009	812	45.3	96

5 Evaluation

Programming is a creative process and therefore it is impossible to formalize the requirements in a very precise and detailed way. The programs submitted for evaluation are very different. For example, the first task of the pilot exam does not include requirements to keep data in the array; in the given programs, the arrays were used for keeping not only the data, but also the results; several programs even employed record arrays. Obviously, that is unreasonable

In 2004, preparation for the national exam was started. At this time there was a concept of programming tasks and a test part. The national examination centre collected data about possible participants in the exam. The prognosis was for 1500 students. At that time there was no experience in evaluating such a larger number of programs. A similar experience was only in International Olympiads in Informatics, but even there, there was a smaller number of participants. In Lithuania the National Olympiad in Informatics with about 400 participants is held too. However, the style of evaluation is different in the contest and in the exam. In contests only the best programmers take part and even in this case, there is sometimes a very low score.

The main difference in the concept of Olympiad and that of exam is an idea of 'fixing' small errors in the program. The problem with this "fixing" concept is that it is difficult to determine whether it is a small error or large, and how many patches we can provide, etc. On the other hand, it is clear that after patching we must retest the program with all data sets, which is unusual for Olympiad, and afterwards think of how many points a student has lost due to this error.

Obviously, the Contest System from Olympiads can be useful, but it cannot be used without significant changes. The national examination centre has made a decision to create a totally new automatic evaluation system with all the requirements met. In the autumn of 2004, the work on system design was started and in February 2005, the system framework was already functional.

Then another phase started – development of different modules responsible for the evaluation on different aspects like evaluation of the programming style. The development still continues, as the main rules of the exam change step-by-step and new ideas arise for better evaluation (Table 4). One of the latest ideas is to integrate a multiple choice and open question answer testing in the same system, by adding C++ language as a possibility for the programming part.

Table 4. Evaluation of the program development

Parts or program evaluation	**% of Points**
Testing. Automatic evaluation.	80
Data structures, data reading, actions of calculation, printing of the results. *Evaluated only if* the results of at least one test are incorrect.	80
Obligatory requirements to the program (procedures and functions for single actions are indicated), programming technology, and style.	20

Application of the evaluation operates with packages of solutions. Each solution must be processed as follows: it must be compiled, and then it must be run with several data sets. The answers provided for all these data sets must be compared with the correct. ones

In some tasks several different outputs can possibly be evaluated as correct. For example, the task is to find the way how to give some amount of money, if you have some set of coins. In this task, it can possibly be found, as usual, several solutions. In this case, the evaluation program must check the sum of selected coins. It is clear that the checking result of the solution program can be rather different from the comparison of two files. This yields an idea to write a separate result correctness checker for each task. As a result, application of evaluation is not one but several programs. The correctness checker comes with a package of testing data and correct

answers. It is also possible to have some specific libraries in the package.

As students are not professional programmers, it is usual to get different simple errors in output format. An example of such a style error can be forgotten spaces between the numbers, all output in one line, etc. The decision was made to split the correctness checker into several programs: result format checker (which is rather a typical scanner as used in translators) and result evaluator. Both of them are prepared before examination by the task authors or engineers. To ease the creation of the format checker specific library is written.

The evaluator team is trying to evaluate the solutions positively. It means that students get points for their effort. For example, correct input / output routines can be assessed by several points. Also, some points can be gained for dividing the program to subroutines, for using complex data structures like the array or record, for writing nice comments, for a good programming style, etc. These criteria can be easily evaluated by a person, while computer evaluation is not so obvious. This is the reason for manual evaluation of solutions.

The practice has showed that evaluators need some interactive evaluating application, as some solutions have only some small syntax problems, like semicolon missing. The evaluators have a possibility to fix an error and to retest the solution. If the test after fixing goes smoothly, only some points are removed from all the points.

However, this manual work is time-consuming and another problem is that some points of programming style are subjective. One way for more precise results is the evaluation of the solution by two different evaluators and comparison of their points. The third evaluator is needed if a difference in the evaluation is observed. However, this is a time-consuming process. After some discussions an idea has arisen that a better similarity of evaluation the programming style can be reached after some training courses. This prompted an idea to create some programming style which can generate the reference points for evaluators.

In the Lithuanian national examination we have a fixed programming language Free Pascal, which has several styles of programming, but they are not very different. However, a good programming style is still debated in programming languages. As stated in [3],"research on measurable programming style definitions was very active in the 1980's". The main problems in this area are standards for a good programming style and choice of measure. P. W. Oman and C. R Cook [9] have proposed taxonomy for the programming style. There is rather a long list of different rules and requirements to the code. However, it is not clear, which requirements are compulsory and which are only suggestions.

6 Conclusions

The maturity exam in information technologies and programming has been prepared according to the advanced module of programming. Obviously, while preparing the exam, the most important part is developing of the appropriate tasks that would examine the students' abilities and express the module content.

Students must have freedom for creativity, even though that is unhandy for the weak ones, since they create so complex and long programs that they lack time to finish them. That is why it is reasonable to limit the freedom of actions by, for example, forbidding using two-dimensional arrays, record data types, etc.... A requirement to create at least one procedure could be made.

In additional, when selecting a solution method to a practical task, students do not consider which of the possible variants will be the simplest one. Weaker students write a program for given data in the task using the number of simple variables such that is needed to save that data (very often by printing using the keyboard or assignment statements to give initial meanings). They do this without realizing that there is a lot of data and the example in the task is only one of them.

It has been noticed that the complexity and size of the programs are mainly determined by the complexity of selected data types. However, students rarely consider this when choosing them.

On the whole, it is relevant to select practical tasks so that the results did not depend on the chosen solution method.

References

1. Anderson, J., Weert, T.: Information and Communication Technology in Education. A Curriculum for Schools and Programme of Teacher Development. Division of Higher Education, UNESCO (2002)
2. Blonskis, J., Dagienė, V.: Evolution of informatics maturity exams and challenge for learning programming. In: R. T. Mittermeir (ed.). Informatics Education – The Bridge between Using and Understanding Computers. LNCS, vol. 4226, pp. 220--229. (2006)
3. Blonskis, J.; Dagienė, V.. Analysis of students' developed programs at the maturity exams in information technologies. In: R.T.Mittermeir, M.M. Syslo (eds.). Informatics Education – Supporting Computational Thinking. LNCS, vol. 5090, Springer, pp. 204--215 (2008)
4. Dagienė, V. Alternation of concepts of Informatics matura exam. Informacijos mokslai, Vilnius, Vol. 16, pp. 39--47 [in Lithuanian] (2001)
5. General Curriculum for General Education School in Lithuania and General Education Standards for Grades XI-XII. Ministry of Education and Science of the Republic of Lithuania, Vilnius. (2002)
6. Curriculum for informatics maturity exams. National examination centre of the Republic of Lithuania, URL: http://www.nec.lt [in Lithuanian] (2007)
7. Tasks of national maturity exams. National examination centre of the Republic of Lithuania, URL: http://www.nec.lt [in Lithuanian] (2007)
8. Project „Development of maturity exams quality system". 2006-2008 SFMIS Nr. BPD2004-ESF-2.4.0-03-05/0107, http://www.egzaminai.lt/49/
9. Major Categories in the Taxonomy of Educational Objectives (Bloom 1956), URL: http://www.krummefamily.org/guides/bloom.html

Game Maker Workshop

Nataša Grgurina[1], Lars Tijsma[2]

[1]University Center for Learning and Teaching, University of Groningen, Landleven 1, 9747 AD Groningen, the Netherlands

n.grgurina@rug.nl

[2]Information and Communication Academy, HAN University of Applied Sciences, Ruitenberglaan 26, 6826 CC Arnhem, the Netherlands

l.tijsma@gmail.com

Abstract. Game Maker is a game-design tool that uses a drag-and-drop action system, along with built-in GML language, to program the events and actions of a game. It has found a place in education where it is used to create games, simulations and other sorts of applications from elementary school on. It can be used to introduce the OO paradigm in an introductory computer science course, and to train a wide range of skills across the whole of the curriculum.

Keywords: Game Maker, education, OO paradigm, introductory CS course.

1 Game Maker

Game Maker [1] is a software application written in the Delphi programming language by Mark Overmars. It was primarily developed to create games, but it is also suitable for developing applications to be used in a range of subjects, for example math or science classes [4]. The feature that makes it interesting for novice programmers is an interface that uses a drag-and-drop system. However, the built-in interpreted Game Maker Language (GML) extends the possibilities for customizing programs and expanding features. Game Maker comes with a set of standard action libraries

that allow for easy implementation of movement, drawings, sound and control structures, and it supports easy import of additional resources. Its architecture supports such things as event detection, level design, and object configuration [2], [3], [6]. Game Maker Lite edition can be obtained free of charge. A more demanding programmer wishing to make 3D games or to use extended graphic options may want to purchase the Game Maker Pro version.

All these characteristics contribute to widespread use of Game Maker in schools [5], [7], [8]. In a CS course, many programming concepts, such as OOP, object vs. instances, inheritance, code structuring, state machines, etc., can be introduced without having to write any code. All students using Game Maker can learn and practice a wide range of skills such as designing a game or other type of application, using other applications to create their own resources, writing project documentation, collaborating in a team, planning, problem-solving, decision-making, and evaluation. Finally, students love to work with it and are motivated to put extra effort into their Game Maker project. A teacher said that:

> *... the biggest problem is that students spend too much time on their games and forget other classes. While using Game Maker you learn a great many things, from simple things like English to more advanced things like creating design documents and writing neural networks. Game Maker lets people learn without them recognising it as homework but rather recognising it as a fun challenge. That's the real power of Game Maker and that's why many schools have been using it and will be using it in the future."* [7]

2 Workshop

During the workshop we will demonstrate the main features of Game Maker. We will then go on to discuss its possible uses in school, both specifically within a CS class as well as in other classes. And we will show examples of its use in schools in the Netherlands. We intend to

conclude the workshop with a hands-on session where the participants will get to work with Game Maker themselves[1].

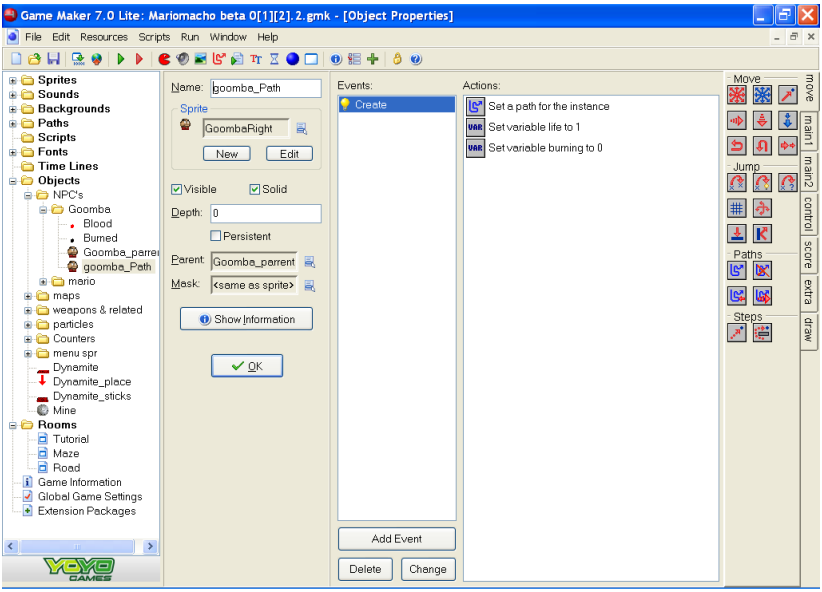


Fig. 1. Game Maker screenshot

1 Ideally, each participant should have a computer with Game Maker installed.

3 Authors

Nataša Grgurina is a computer science teacher educator at the University of Groningen, as well as a CS teacher in a high school. She has been looking for an attractive programming environment that is simple enough for novices to start programming meaningful applications right away, while providing an opportunity to teach complex programming concepts at the same time. She believes to have found this in Game Maker.

She conducted a version of this workshop with CS student teachers at Vilnius University where it received very positive feedback.

Lars Tijsma is a CS teacher and a theater devotee. He teaches CS (previously at a high school and nowadays at the HAN University of Applied Sciences in Arnhem), writes CS teaching materials and advocates the use of games in education.

References

1. Game Maker website, www.yoyogames.com
2. Game Maker, Wikipedia, http://en.wikipedia.org/wiki/Game_Maker
3. Game Maker Language, Wikipedia, http://en.wikipedia.org/wiki/Game_Maker_Language
4. Nexus Research Groep, http://www.nexusresearchgroup.com/info_systems/games.htm
5. Game Maker Website Australia, http://www.users.on.net/~billkerr/g/ad.htm
6. Overmars, M., Habgood, J.: The Game Makers Apprentice: Game Development for Beginners, Springer-Verlag, New York (2006)
7. Gamble, P.: Game Maker in Schools, In: Game Maker Technology Magazine, Issue 16, June 2009, http://gamemakertech.info/
8. Make-A-Game, http://www.make-a-game.nl/

Personalisation of Learning Objects and Environments for Informatics Science Education in Lithuania

Eugenijus Kurilovas, Silvija Serikoviene

Institute of Mathematics and Informatics, Akademijos str. 4, LT-08663 Vilnius, Lithuania

Eugenijus.kurilovas@itc.smm.lt; silvija.serikoviene@gmail.com

Abstract. Learning content and software personalisation issues are very significant for the enhancement of quality of Informatics science (or Information Technologies – IT) education. The paper is aimed to analyse the problems of personalisation of learning content and software in Informatics science education, as well as their technological quality evaluation and optimisation. The results of INSPIRE project in Lithuania are analysed in more detail. Several scientific methods and principles are used in the paper to provide some engineering solutions for personalisation of IT teaching and learning.

Keywords: learning objects, virtual learning environments, personalisation, informatics science education

1 Introduction

The aim of the paper is to analyse some personalisation problems of learning content (Learning Objects – LOs) and software (such as LO Repositories – LORs, and Virtual Learning Environments – VLEs) for Informatics (or IT) subject education based on the results of INSPIRE project [10] and previous authors' research. Learning content and software personalisation issues are very significant for the enhancement of quality of IT subject education as well as IT integration into the other subjects. The vision of the Strategy on Information and Communication Technologies (ICT) implementation

into Lithuanian education is achievement of learning personalisation with the help of ICT application.

Informatics (or IT) is taught as separate subject in Lithuanian comprehensive schools since the 5th grade. Students of 5-6th grades are trained in information processing, text documents, printing, searching on the Internet. Students of 7-8th grades concentrate on the use of IT skills and focus on integration with other curriculum subjects, aims to encourage students to apply IT for learning other things, enabling them to reach the general level of computer literacy. Competence in the use of IT is developed in teaching and learning of other subjects – languages, mathematics, natural sciences, social sciences, technology. IT course in of 9-10th grades aims to summarize the available knowledge, to teach pupils to purposefully adapt existing skills. IT course becomes more specific.

The basic notions, principles and methods applied in the paper are as follows.

LO is referred to as any digital resource that can be reused to support learning [23]. LORs are considered here as properly constituted systems (i.e., organised LOs collections) consisting of LOs, their metadata and tools / services to manage them [14]. Metadata is referred to as structured data about data [4]. VLEs are considered here as specific information systems which provide the possibility to create and use different learning scenarios and methods [9]. Quality evaluation is defined as the systematic examination of the extent to which an entity (part, product, service or organisation) is capable of meeting specified requirements [11].

Different scientific methods are used in software engineering to customise LOs according to the particular users' needs. The majority of them deal with the implementation of flexible LOs metadata standards' application profiles (APs) and search engines in the LORs based on these approaches.

Different scientific methods are also used for quality evaluation of learning software packages (such LORs and VLEs) and their optimisation for the particular learners' needs. Multiple criteria evaluation method used by the authors is referred to as the experts'

additive utility function presented further in Section 4 including the alternatives' evaluation criteria, their values and weights.

Expert evaluation is referred to as the multiple criteria evaluation of the learning software aimed at the selection of the best alternative based on score-ranking results. According to [5], if the set of decision alternatives is assumed to be predefined, fixed and finite, then the decision problem is to choose the optimal alternative or, maybe, to rank them. But usually the experts (decision makers) have to deal with the problem of optimal decision in the multiple criteria situation where the objectives are often conflicting. In this case, according to [5], an optimal decision is the one that maximises the decision maker's utility.

The authors apply the software engineering principle which claims that one should evaluate the software using two different groups of evaluation criteria – 'internal quality' and 'quality in use' criteria. 'Internal quality' is a descriptive characteristic that describes the quality of software independently from any particular context of its use, and 'quality in use' is evaluative characteristic of software obtained by making a judgment based on criteria that determine the worthiness of software for a particular project or user / group . It is impossible to evaluate quality in use without knowing characteristics of internal quality [7].

The rest of the paper is organised as follows. Section 2 presents INSPIRE project results in Lithuania, Section 3 – one of the methods to personalise LOs, Section 4 – evaluation, optimization and personalisation of VLEs. Conclusion and results are provided in Section 5.

2 INSPIRE Project Results in Lithuania

INSPIRE project has been proposed on the following reasons. Europe's future competitiveness in the global economy will depend to a great extent on its supply of scientific specialists and on ensuring that they are put to good use. Mathematics, Science and Technology (MST), including computer science, environmental science and

engineering are vital for the development of the knowledge-based and increasingly digital economy.

The INSPIRE project has proposed to set up a limited validation observatory where 60 schools in Europe have been be proposed to use, test, analyse the use of new LOs from European Learning Resource Exchange (LRE) [18] portal in the field of MST. Through this experimentation, special attention has been given and reported on as regards: (1) the impact of the new LOs and teaching methods at the level of pupils and their motivation, (2) the analysis of the pre-requisites to be defined for enabling the teachers to integrate these new techniques in their pedagogy, and (3) the critical success factors to be mastered at the level of the teacher and the school for the generalisation of such practices.

The authors while being INSPIRE coordinators in Lithuania have performed the questionnaires-based survey of the MST teachers in 10 Lithuanian comprehensive schools. 10 IT teachers from 10 schools have participated in the survey. 12 LOs on IT subject from LRE have been proposed to the teachers to evaluate during the experiment in real pedagogical contexts in their schools.

Some results of this survey are presented further. Tables $1 - 3$ present the results of the survey aimed to analyse the IT subject's learners' pre-requisites, enhanced competences (both general and subject), learning and assessment methods and digital environments (VLEs) used by IT teachers during the experiment, as well as the teachers' conclusion of LOs usability in future.

Table 1. General information

Names	Values	Ratings
Learner profile information	High knowledge / skills level	0
	Average knowledge / skills level	9
	Low knowledge/skills level	0
	Gifted	0
	Motivated	5
	Needs personalisation	0

Learning aims / General competences	Communication in mother tongue	6
	Communication in foreign language	4
	Competence in MST	2
	Digital competence	6
	Learning to learn	3
	Social competencies	1
	Enterprising and Creativity	3
	Personal and Cultural understanding	4
Subject competences	Fit the curriculum	10
	Do not fit the curriculum	0
Digital environment used in the experiment	Moodle	3
	LeMill	1
	Other	5
	Not used	0
Conclusion on LOs usability	To localise and use	3
	To use without localisation	7
	Not to use	0

Table 2. Learning methods used during the experiment

Learning methods	Description	Ratings
By information source	Word-based methods	3
	Visual-based methods	7
By theory and practice ratio	Theoretical methods	0
	Practice-based methods	10
By teacher and students activity relationship	Active learning methods	0
	Passive learning methods	7
By authoritarianism and humanity relationships	Programme-oriented methods	4
	Student-oriented methods	9
	Authoritarian methods	1
	Humane methods	1
By the students activity creativity level	Reproductive methods	5
	Creative methods	3
By the students reasoning operations relationships with the	Analysis	5
	Synthesis	6
	Abstraction and generalisation	3

logical forms and shapes	Deduction and induction	1
	Analogy	2
	Hypothesis	2
	Experiment	5

Table 3. Assessment methods used during the experiment

Description	Ratings
Test	3
Credit	0
Practical assignment	8
Creative assignment	3
Self-assessment	2
E-Portfolio	0
Project work	1

Learning methods taxonomy in Table 2 has been developed according to [21].

Tables $1 - 3$ show that all general competences were addresses by the proposed LOs, and different pedagogically sound proactive learning and assessment methods have been used during the experiment. In IT teachers' opinion, VLE Moodle is the most suitable digital environment to implement these learning and assessment methods while working with personalised and decontextualised LOs. They also think that the majority of LOs are suitable to use without localisation, and the others require localisation before implementation in school practice.

We can personalise LOs and VLEs according to the learners' profiles and preferences concerning teaching / learning methods, speed, etc.

Now let us analyse LOs and VLEs personalisation issues addressed in INSPIRE using several scientific principles and methods known in software engineering and described in the Introduction.

3 Personalisation of Learning Objects

3.1 Learning Objects Reusability

In the authors' point of view, one of the main criteria for achieving high LOs effectiveness and personalisation level is LOs reusability [2]. The need for reusability of LOs has at least three elements: (1) Interoperability: LO is interoperable and can be used in different platforms; (2) Flexibility in terms of pedagogic situations: LO can fit into a variety of pedagogic situations; and (3) Modifiability to suit a particular teacher's or student's needs: LO can be made more appropriate to a pedagogic situation by modifying it to suit a particular teacher's or student's needs [19]. There are two main conditions for LOs reusability elsewhere: (1) LOs have to fit different countries national curricula; (2) Different countries' IEEE Learning Object Metadata (LOM) standard's APs have to be oriented towards quick and convenient search of reusable LOs [13]. The principle of ultimate increase of reusability of LOs is considered by the authors as one of the main factors of e-learning systems flexibility [2] [3]. It was analysed that the flexible approach to the e-learning systems' creation and development should be based on the idea of LOs' partition to two main separate parts, i.e., LOM compliant small pedagogically decontextualised Learning Assets (LAs) as well as LOM and IMS Learning Design compliant Units of Learning – UoLs [3] [16].

European LRE system's validation in Lithuania performed by the authors while implementing FP6 CALIBRATE project [1] has shown that the teachers prefer LOs from national repositories which have the potential to 'travel well' and can be used in different national contexts. These reusable LOs preferred by the teachers are mainly the small decontextualised LAs. Therefore in order to maximise LOs reusability in Europe LRE should consist mainly of the decontextualised LAs [16]. The results of the teachers-experts survey performed by the authors in CALIBRATE show that the teachers would mostly like to find pedagogically decontextualised reusable LOs and therefore to have a service for quick and convenient search

of such LOs. These results are similar to INSPIRE results on LOs implementation in educational practice. While searching for LOs in CALIBRATE / LRE portal the experts have used browsing by subject and advance search services. These advance search services have not contained any services to ease the search of reusable LOs. The LOs in the portal are described according to the partners' LOM APs, and these APs have not contained any services to simplify the search of reusable LOs. Therefore it took very much time for the experts to find and choose suitable LOs for their lesson plans.

According to [13], the analysis of the existing and emerging interoperability standards and specifications shows that: (1) The majority of standards and specifications are not adopted and do not conform to the educational practice; (2) There exists a problem of complex solutions for the application of standards and specifications in education; (3) Standards and specifications often do not cooperate. First of all, in order to make it easier for educators to discover and use LOs that addresses the needs of their students, to maximise reuse of LOs and minimise the costs associated with their repurposing, the good solutions are lacking for the specific application profiles of IEEE LOM [13].

3.2 Customisation of Learning Objects Metadata

According to [4], the purpose of an AP is to adapt or combine existing schemas into a package that is tailored to the functional requirements of a particular application, while retaining interoperability with the original base schemas. There are several principles described in [4] providing a guiding framework for the development of practical solutions for semantic and machine interoperability in any domain using any set of metadata standards: modularity, extensibility, refinement and multilingualism. One of the mechanisms for APs to achieve modularity is the elements' cardinality enforcement. Cardinality refers to constraints on the appearance of an element. Is it mandatory or recommended or optional? According to [4], the status of some data elements can be made more stringent in a given context. For instance, an optional

data element can be made recommended, and a recommended data can be made mandatory in a particular AP. On the other hand, as an AP must operate within the interoperability constraints defined by the standard, it cannot relax the status of data elements [4].

The authors have applied this cardinality enforcement principle in their research. It was analysed that the main LOM elements which vocabulary values could reflect the LOs ultimate reusability deal with structure of LO, its functional granularity (aggregation) level, educational type as well as the kind of relation of this LO with the others [16]. The results of the authors' analysis of the European LRE Metadata AP v3.0 have shown that it would be purposeful to improve it in order to provide more quick and convenient search possibilities for those searching ultimately reusable LOs (i.e., LAs) by the means of changing (i.e., advancing / enforcing cardinality) the status of a number of LRE AP elements.

These proposals deal with changing the status of the following LOM AP elements from 'optional' to 'recommended' as well as from 'optional' and 'recommended' – to 'mandatory':

1) 1.7 General. Structure;
2) 1.8 General. Aggregation Level;
3) 5.2 Educational. Learning Resource Type; and
4) 7.1 Relation. Kind (see Figure 1).

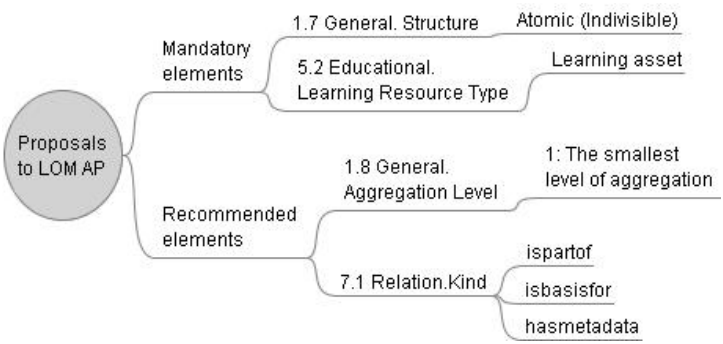


Fig. 1. Proposals on customisable metadata schema [13]

These elements should be included in the advanced search engine for those looking for reusable LOs to use them as 'building blocks' in their own lesson plans, modules or courses. The authors believe that the development of advanced search engine reflecting LOs reusability level based on this research would considerably reduce the time for the users to find and choose suitable LOs in the repositories.

There are more methods of customisation / personalisation of LOs metadata. They could be, e.g., based on the customisation of controlled vocabularies, implementation of the learners' profiles or users' tags to search for preferred LOs in the repositories. The extended search and management of controlled vocabularies by desirable elements are also often implemented in the LOs repositories to enhance the customisation of LOs for the personal users needs.

4 Optimisation and Personalisation of Virtual Learning Environments

4.1 Virtual Learning Environments Quality Evaluation Criteria

In order to choose the VLE suitable for personalised learners needs, one should apply well-developed scientific methods for evaluation of VLEs. One can divide evaluation methods of VLE quality to pedagogical, organisational and technological methods. The aim of this chapter is to analyse VLEs technological evaluation method – the expert's additive utility function containing the alternatives' criteria values and their weights is proposed further for this aim. Other criteria are out of scope of the paper.

The authors' analysis [15] of existing well-known VLEs evaluation tools and methods shows that the analysed VLE technological evaluation methods [22] [8] have a number of limitations:

5) The method developed in [22] practically does not examine VLEs adaptation capabilities criteria.
6) The method proposed by [8] insufficiently examines general technological quality criteria of VLEs.

In Methodology of Technical Evaluation of Learning Management Systems (or VLEs) [22] the evaluation criteria expand on a subset of the criteria, focusing on general technological aspects of VLEs:

7) Overall architecture and implementation: Scalability of the system; System modularity and extensibility; Possibility of multiple installations on a single platform; Reasonable performance optimisations; Look and feel is configurable; Security; Modular authentication; Robustness and stability; Installation, dependencies and portability.
8) Interoperability: Integration is straightforward; VLE standards support.
9) Internationalisation and: Localisable user interface; Localisation to relevant languages; Unicode text editing and storage; Time zones and date localisation; Alternative language support.
10) Accessibility: Text-only navigation support; Scalable fonts and graphics.

Conversely to [22], in [8] the main attention is paid to the adaptation set of criteria. These criteria are:

11) Adaptability – includes all facilities to customise the platform / VLE for the educational institution needs (e.g., the language or the design).
12) Personalisation aspects – indicate the facilities of each individual user to customise his / her own view of the platform.
13) Extensibility – is, in principle, possible for all open source products. Nevertheless, there can be big differences. For example, a good programming style or the availability of a documented application programming interfaces are helpful.
14) Adaptivity – indicates all kinds of automatic adaptation to the individual user's needs (e.g., personal annotations of LOs or automatically adapted content).

Therefore, in the authors' opinion, a more comprehensive tool / set of criteria for VLE technological evaluation is needed. It should include General technological evaluation criteria and Adaptation capabilities criteria [15]. On the other hand the comprehensive VLEs

quality evaluation tool should include both general VLEs 'internal quality' criteria and 'quality in use' criteria [7] (see Table 4).

Table 4. VLE technological evaluation criteria [10]

Criteria type	Criteria	Sub-criteria
General criteria	1) Overall architecture and implementation	Scalability
		Modularity of the architecture
		Possibility of multiple installations on a single platform
		Reasonable performance optimisations
		Look and feel is configurable
		Security
		Modular authentication
		Robustness and stability
		Installation, dependencies and portability
	2) Interoperability	Integration is straightforward
		VLE standard support (IMS, SCORM, etc.)
	3) Internationalisation and localisation	Localisable user interface
		Localisation to relevant languages
		Unicode text editing and storage
		Time zones and date localisation
		Alternative language support
	4) Accessibility	Text only navigation support
		Scalable fonts and graphics
Adaptation criteria	5) Adaptability	Language
		Design
	6) Personalisation aspects	
	7) Extensibility	Good programming style
		Availability of a documented API
	8) Adaptivity	Personal annotations of LOs
		Automatically adapted content

The tool is suitable for the expert evaluation of both VLEs 'internal quality' criteria 1–4 and 'quality in use' criteria 5–8. It provides the experts the clear instrumentality who (i.e., what kind of experts) should analyse what kind of VLEs quality criteria in order to select the best VLE software package suitable for their particular needs.

4.2 Experimental Evaluation of Virtual Learning Environments

Multiple criteria evaluation method is referred to as the experts' additive utility function presented further in the section including the alternatives' evaluation criteria, their values and weights. The weight of the evaluation criterion reflects the experts' opinion on the criterion's importance level in comparison with the other criteria for the individual learner / user.

The expert's additive utility function needs the methods for measurement of the values and weights of VLEs evaluation criteria presented in Table 4.

The measurement criteria of the decision attributes' quality are mainly qualitative and subjective. Decisions in this context are often expressed in natural language, and evaluators are unable to assign exact numerical values to the different criteria. Assessment can be often performed by linguistic variables: 'bad', 'poor', 'fair', 'good' and 'excellent'. These values, e.g., used in [22] are imprecise and uncertain: they are commonly called fuzzy values. Integrating these different judgments to obtain a final evaluation is not evident.

Therefore, [20] propose to use fuzzy group decision making theory to obtain final assessment measures. First, linguistic variable values are mapped into triangular fuzzy numbers (l, m, u) (see Table 5).

Table 5. Linguistic variables conversion into triangular fuzzy numbers (TFNs)

Linguistic variables	TFN
Excellent	(0.700, 0.850, 1.000)
Good	(0.525, 0.675, 0.825)
Fair	(0.350, 0.500, 0.650)
Poor	(0.175, 0.325, 0.475)
Bad	(0.000, 0.150, 0.300)

After the defuzzification procedure which converts the global fuzzy evaluation results, expressed by a TFN (l, m, u), to a non-fuzzy value E, the following equation has been adopted by [20]:

$$E = [\,(u - l\,) + (\,m - l\,)\,]\,/\,3 + l. \qquad (\mathbf{1})$$

These non-fuzzy values E are suitable to be applied to measure the ratings of the evaluation criteria of learning software packages such as VLEs and LORs.

Table 6. Linguistic variables conversion into non-fuzzy values E according to (1)

Linguistic variables	Non-fuzzy value E
Excellent	0.850
Good	0.675
Fair	0.500
Poor	0.325
Bad	0.150

If we want to evaluate (or optimise) the technological quality of VLEs for the particular learner needs (i.e., to personalise his / her learning process in the best way according to their prerequisites, preferred learning speed and methods, etc.), we should use the experts' additive utility function together with the weights of evaluation criteria. Expert evaluation is referred here as the multiple

criteria evaluation of software aimed at the selection of the best alternative based on score-ranking results.

For example, for the most simple (general) case, when all VLE evaluation criteria are of equal importance, the experts should consider the equal normalised weights $a_i = 0.125$ according to the normalisation requirement

$$\sum_{i=1}^{m} a_i = 1 \ , \ a_i > 0 \ . \qquad (2)$$

for the VLEs quality evaluation criteria $i = \{1, \ldots, 8\}$ (see Table 4).

A possible decision could be to transform multi-criteria task into one-criterion task obtained by adding all criteria together with their weights. It is valid from the point of view of the optimisation theory, and a special theorem exists for this case.

Therefore one could formulate the experts' additive utility function as follows:

$$f(X) = \sum_{i=1}^{m} a_i f_i(X) \ , \ \sum_{i=1}^{m} a_i = 1 \ , \ a_i > 0 \ . \qquad (3)$$

The major is the meaning of the utility function (3) the better VLE meets the particular learner needs.

The application of this method for evaluation of LORs quality has been presented by the authors while implementing EdReNe [6] project during the Workshop in Sestri Levante (Italy) in September 2008.. EdReNe brings together web-based repositories of LOs with content owners and other stakeholders within education in order to share, develop and document strategies, experiences, practices, solutions, advice, procedures etc. on the organisation, structuring and functionality of repositories [6]. The LORs quality assurance strategies have been ranked the highest priority by the EdReNe experts during the project Strategic seminar in Lisbon in June 2008.

VLE experimental evaluation results for general case, when all criteria are of equal importance are presented in Table 7. The non-

fuzzy values E are calculated according to the equation (1), and all VLE evaluation criteria here are of equal importance $a_i = 0.125$.

Table 7. VLEs technological evaluation summary (all criteria, equal weights)

Evaluation criteria	ATutor	Ilias	Moodle
	General criteria		
Architecture and implementation	0.500	0.325	0.850
Interoperability	0.675	0.675	0.500
Internationalisation and localisation	0.325	0.500	0.675
Accessibility	0.850	0.325	0.500
Interim rating	*2.350*	*1.825*	*2.525*
	Adaptation criteria		
Adaptability	0.325	0.500	0.675
Personalisation	0.675	0.675	0.500
Extensibility	0.675	0.850	0.850
Adaptivity	0.325	0.150	0.325
Interim rating	*2.000*	*2.175*	*2.350*
Total evaluation rating	*4.350*	*4.000*	*4.875*
f(X) (weights = 0.125)	*0.5437*	*0.5000*	*0.6093*

These results mean that VLE Moodle meets 60.93% quality in comparison with the ideal (less than 'good'), ATutor – 54.37% (more than 'fair'), and Ilias – 50.00% ('fair'). According to this experimental evaluation results, VLE Moodle is the best alternative (among the evaluated) from technological point of view in general case. This alternative has shown the highest ratings of both 'internal quality' evaluation (see General criteria ratings) and 'quality in use' evaluation (see Adaptation criteria ratings).

In more specific cases, e.g., if the experts (decision makers) would like to select the most suitable VLE for the students with special education needs / disabilities, they should choose higher weights for the particular criteria: Accessibility (e.g., measuring weight $a_4 = 0.2$)

and Personalisation (e.g., measuring weight $a_6 = 0.2$). All the other criteria weights according to the normalisation formula (2) should be measured $a_i = 0.1$.

In this particular case the experts should find that, differently from the simple general case (see Table 7), both ATutor and Moodle are the optimal VLEs for the learners with special needs (see Table 8):

Table 8. VLEs technological evaluation summary (all criteria, different weights)

Evaluation criteria	ATutor	Ilias	Moodle
General criteria			
Architecture and implementation $a_1 = 0.1$	0.0500	0.0325	0.0850
Interoperability $a_2 = 0.1$	0.0675	0.0675	0.0500
Internationalisation and localisation $a_3 = 0.1$	0.0325	0.0500	0.0675
Accessibility $a_4 = 0.2$	0.1700	0.0650	0.1000
Interim rating	*0.3200*	*0.2150*	*0.3025*
Adaptation criteria			
Adaptability $a_5 = 0.1$	0.0325	0.0500	0.0675
Personalisation $a_6 = 0.2$	0.1350	0.1350	0.1000
Extensibility $a_7 = 0.1$	0.0675	0.0850	0.0850
Adaptivity $a_8 = 0.1$	0.0325	0.0150	0.0325
Interim rating	*0.2675*	*0.2850*	*0.2850*
Total evaluation rating f(X)	*0.5875*	*0.5000*	*0.5875*

These results mean that both VLEs ATutor and Moodle meet 58.75% quality in comparison with the ideal for special needs students (something between 'fair' and 'good"), and Ilias – 50.00% ('fair').

4.3 Minimisation of the Experts Subjectivity

Another very complicated problem for such multiple criteria evaluation and optimisation tasks is minimisation of the experts'

(decision makers') subjectivity. The experts' subjectivity can influence the quality criteria ratings (values) and their weights.

There are some scientific approaches concerning this item. One of them is formulated in [12]. In general, the experts influence importance is different, and therefore this importance should be assessed using the appropriate methodology. It is important to form the experts group purely by their competence. Furthermore, in conformity with [12], we should eliminate the extreme experts' assessments of the ratings and weights. In order to pursue the compatibility of the experts' assessments we should calculate so-called concordance rates W and distributions λ^2:

$$W = \frac{12\,S}{r^2(m^3 - m)}\,. \tag{4}$$

where r – the number of experts; m – the number of the parameters under evaluation; S – the square sum of evaluated importance rates' values deviations from the experts' aggregate average. In its turn,

$$\lambda^2 = Wr(m-1) = \frac{12\,S}{rm(m+1)}\,. \tag{5}$$

The compatibility of the experts' assessments is considered sufficient if the value of concordance rate W is 0.6–0.7 [12].

5 Conclusion and Results

Learning content and software personalisation issues have been found very significant for the enhancement of quality of Informatics education while implementation of INSPIRE project.

Personalisation of learning content and software could be enhanced by the presented LOs metadata customisation method and the multiple criteria evaluation method suitable for evaluation of quality of learning software such as LORs and VLEs.

The proposed VLEs multiple criteria evaluation method represented by the experts' additive utility function (3) is based one the transformation of the multiple criteria task into the one-criterion task obtained by adding all criteria values together with their weights.

This multiple criteria evaluation method is suitable to apply for the VLEs practical expert evaluation to meet the particular learner needs. Therefore, it is of practical importance for public and private sectors' experts (decision makers), software engineers, programmers and users.

References

1. CALIBRATE: FP6 IST CALIBRATE (Calibrating eLearning in Schools) project web site. http://calibrate.eun.org
2. Dagienė, V., Kurilovas, E.: Information Technologies in Education: Experience and Analysis. Monograph. – Vilnius: Institute of Mathematics and Informatics, 2008 – 216 p. (in Lithuanian) (2008)
3. Dagiene, V., Kurilovas, E.: Design of Lithuanian Digital Library of Educational Resources and Services: the Problem of Interoperability. Information Technologies and Control. Kaunas: Technologija. Vol. 36 (4), pp. 402--411 (2007)
4. Duval, E., Hodgins, W., Sutton, S. and Weibel, S. L.: Metadata Principles and Practicalities. D-Lib Magazine, vol. 8, No 4, 2002. http://www.dlib.org/dlib/april02/weibel/04weibel.html
5. Dzemyda, G., Saltenis, V.: Multiple Criteria Decision Support System: Methods, User's Interface and Applications. Informatica. Vol. 5, No 1--2, pp. 31--42 (1994)
6. EdReNe: EU eContentplus programme's Educational Repositories Network project web site, http://edrene.org/
7. Gasperovic, J., Caplinskas, A.: Methodology to evaluate the functionality of specification languages. Informatica. Vol. 17, No 3, pp. 325--346 (2006)
8. Graf, S.; List, B.: An Evaluation of Open Source E-Learning Platforms Stressing Adaptation Issues. Presented at ICALT (2005)
9. Institute of Mathematics and Informatics. Research on Digital Learning Tools and Virtual Learning Environments Implementation in Vocational

Education. Scientific research report, 2005, p. 80. http://www.emokykla.lt/lt.php/tyrimai/194 (in Lithuanian)
10. INSPIRE: EU LLP INSPIRE (Innovative Science Pedagogy in Research and Education) project web site, http://inspire.eun.org/index.php/Main_Page
11. ISO/IEC 14598-1:1999. Information Technology – Software Product Evaluation – Part 1: General Overview. First edition,1999-04-15
12. Kendall, M.: Rank correlation methods. Griffin and Co, London, p. 456 (1979)
13. Kurilovas, E.: Interoperability, Standards and Metadata for e-Learning. In: G.A. Papadopoulos and C. Badica (Eds.): Intelligent Distributed Computing III, SCI 237, pp. 121--130. Springer-Verlag Berlin Heidelberg (2009)
14. Kurilovas, E.: Evaluation and Optimisation of e-Learning Software Packages: Learning Object Repositories. In: Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA 2009). Porto, Portugal, September 20--25, 2009
15. Kurilovas, E.; Dagiene, V.: Learning Objects and Virtual Learning Environments Technical Evaluation Criteria. Electronic Journal of e-Learning. Vol. 7, Issue 2, pp. 127--136. Available online at www.ejel.org (2009)
16. Kurilovas, E., Kubilinskienė, S.: Interoperability Framework for Components of Digital Library of Educational Resources and Services. Informacijos mokslai. Vilnius, Vol. 44, pp. 88--97 (2008)
17. LOM Repository: Lithuanian public repository for LOs metadata, http://lom.emokykla.lt/public/
18. LRE: European Learning Resource Exchange service for schools web site, http://lreforschools.eun.org/LRE-Portal/Index.iface
19. McCormick, R., Scrimshaw, P., Li, N., and Clifford, C.: CELEBRATE Evaluation report. http://celebrate.eun.org/eun.org2/eun/Include_to_content/celebrate/file/Deliverable7_2EvaluationReport02Dec04.pdf (2004)
20. Ounaies, H.Z., Jamoussi, Y., Ben Ghezala, H.H. : Evaluation framework based on fuzzy measured method in adaptive learning system. Themes in Science and Technology Education. Vol. 1, Nr. 1, 2009, pp. 49--58 (2009)

21. Šiaučiukėnienė, L.; Visockienė, O.; Talijūnienė, P.: Šiuolaikinės didaktikos pagrindai. Vadovėlis. Kaunas: Technologija (in Lithuanian) (2006)
22. Technical Evaluation of selected Learning Management Systems (2004). https://eduforge.org/docman/view.php/7/18/LMS%20Technical%20Evaluation%20-%20May04.pdf
23. Wiley. D. A.: Connecting Learning Objects to Instructional design Theory: a definition, a Metaphor, and a Taxonomy. Utah State University. http://www.reusability.org/read/ (2000)

Appendix

The work presented in this paper is partially supported by the European Commission under the eContent*plus* programme – as part of the EdReNe project, Project Number ECP-2006-EDU-42002. The author is solely responsible for the content of this paper. It does not represent the opinion of the European Commission, and the European Commission is not responsible for any use that might be made of data appearing therein.

Reflections on Software Tools in Informatics Teaching

Peter Micheuz[1]

[1] Alpen-Adria Universität Klagenfurt
Institut für Informatiksysteme

peter.micheuz@uni-klu.ac.at

Abstract. Undoubtedly, software tools play an outstanding and dominant role in Informatics teaching. Currently we experience a plethora of these tools in all areas of Informatics education. The vast variety of still proliferating tools, together with their immanent interdependency with underlying concepts and purposes, issues a didactic challenge to all Informatics teachers. Starting with related results of an empirical study in Austrian upper secondary education, software tools are reflected from different perspectives.

1 Introduction

Since its beginning, the history of Informatics education (not only in Austrian schools) is not least the history of software and its use in Informatics teaching. Any use of computers in the wide field of Informatics education is inherently interwoven with using software in its diversity and complexity. A comprehensive understanding of Informatics education at schools encompasses three major highly dependent fields as depicted in Fig. 1.

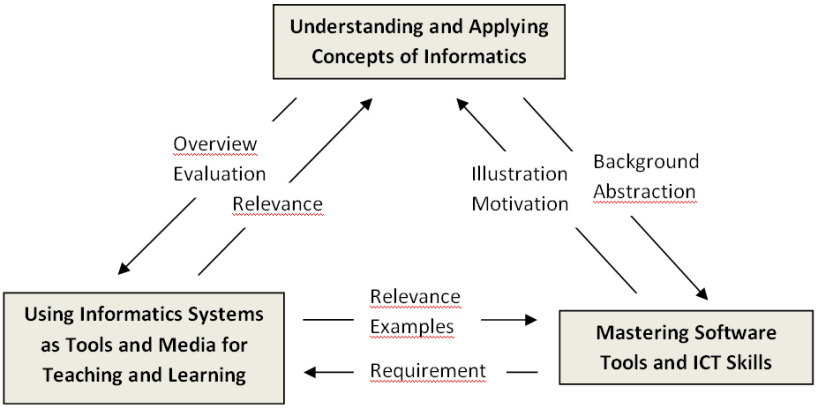


Fig. 1. Synthesis of Informatics Education [1]

Formal Informatics education at schools[1] is characterized and fundamentally influenced by computers and software. Not surprisingly, within the subject Informatics the computer emerges as

- an abstract machine (as subject-matter and for theoretical reflection),
- a concrete tool (for executing specific tasks and solving problems practically),
- a versatile medium (for supporting teaching and learning Informatics).

Software tools are ubiquitous. As immanent dynamic parts of Informatics systems[2] they play an exceptional role in all three manifestations, mapping consistently to the pillars of Informatics education as depicted in Fig. 1.

Software tools are also constitutive for informatics systems as media. For some years already, in everyday life computers are perceived rather as media than as tools. Perhaps Alan Kay gets to

1 Informatics is implemented in most countries as a separate subject in various forms and in different extensions. Where this is not the case yet, it is claimed by many stakeholders in form of resolutions.

2 An "Informatics system" is defined as the combination of hardware and software (in a network environment) for solving application problems.

point of it by his intuitive definition: "The Computer is a medium. I always thought it as a tool, a much weaker concept."

2 An Austrian Case Study

2.1 The Particular Situation in Secondary Academic Schools

This type of schools, also denoted as "Gymnasium" (grammar school), comprises lower/upper secondary education and is attended by approximately 200.000 out of 1.200.000 Austrian pupils and students, aged from 10 to 18 years. The role of ICT/Informatics in these schools has been described already in [2,3,4]. According to the title of this paper, in this chapter additional empirical findings on applied software (tools) will be given.

In a holistic view, Informatics education in Austrian secondary academic schools can be described – euphemistically - as diverse, if not somehow "anarchistic". Due to lack of strict regulations and standards, schools and teachers can act autonomously to a wide extent. This applies in particular to the free choice of software tools.

Due to autonomy of schools, formal Informatics instruction in lower secondary education is offered by each school in different ways and extensions [6]. The use of standard software (MS Office) and product training in dedicated Informatics lessons at ECDL-level[3] are the norm, whereas other software tools (e.g. webdesign and programming tools) are rather exceptions. E-Learning develops in the age group 10-14 years fairly well, with the learning platform Moodle as the prevalent backbone serving as content delivery and communication tool.

In this chapter I draw on an online-survey which I conducted in 2007. It focuses on findings about software tool issues in upper secondary level.

3 European Computer Driving License

2.2 Informatics in the 9th grade

In contrast to lower secondary level, Informatics in the 9th grade – the first year in upper secondary education, where students are 15 years old - is obligatory and mainly (input)controlled by a compact and open curriculum [7]. As a consequence the range of software used in these lessons is very wide. The results of the survey reveal a clear picture about the setting of priorities in this age group, reflecting the main subject matters in this discipline.

In Fig. 4 the eighteen most frequently used software tools are listed. The diagram shows the prevalence of Microsoft® (MS) name-branded software products. Open source software as Phase, a proprietary German free HTML editor, Open Office and the image processing software Gimp play (still) a minor role.

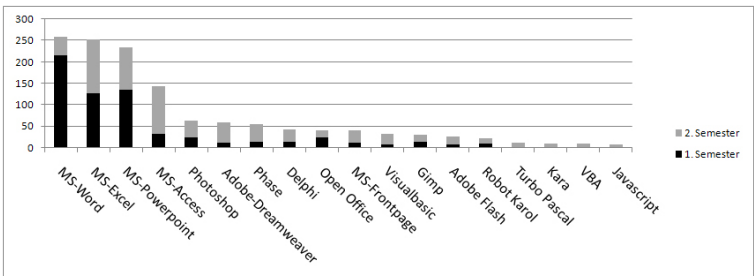


Fig. 2. Concrete software products used in the 9th grade (n = 270).

Fig. 3. Occasionally used tools with max. three nominations from 270 teachers.

Obviously, the first semester is dominated by branded standard software tools, whereas in the second semester database software,

image processing and programming languages as Delphi, Visual Basic, Robot Karol, VBA and even Javascript gain some currency.

Austrian teachers seem to be very creative in harnessing even seemingly exotic tools in their Informatics lessons. The free webeditors Bluefish and Topstyle, as enumerated in the tag-cloud shown in Fig. 3., are examples for that assertion.

2.3 Informatics in the 10^{th} – 12^{th} grades

The content-related part of the central curriculum for the elective subject Informatics which is chosen by about 20% of the students in the 10^{th} – 12^{th} grades (16 - 18 years) consists of a random list of topics, such as

- principles of information processing,
- concepts of operating systems, networks and of programming languages,
- extensions of essentials of Informatics, algorithms and data structures,
- artificial intelligence and social/legal aspects.

This comprising curricular input raises the question of utilized software tools. The answer is given by about 25% of the responding teachers for these age groups. As a result, database software, webdesign tools, programming languages and client-server tools come into play in the course of higher grades at the expense of standard software which still matters especially in the 10^{th} grade.

Viewing at the impressive remaining "software tool jungle", the old Roman proverb "quot capita, tot sententiae" can be replaced by "quot capita, tot instrumenta.".

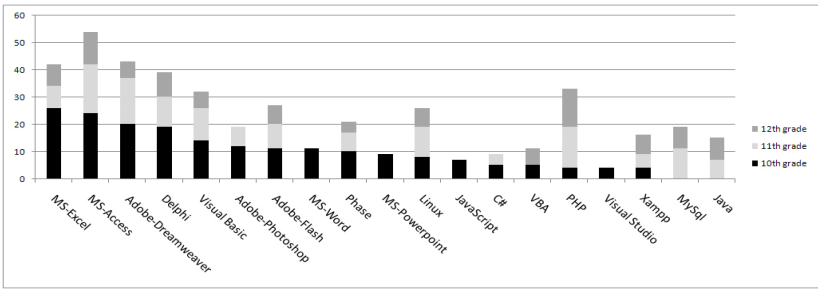


Fig. 4. Concrete software products used in the $10^{th} - 12^{th}$ grades (n ~ 100)

Fig. 5. Other tools used (max. three nominations by 100 teachers)

2.4 About Software Products and Tools

In the same online-survey Informatics teachers have been asked if they are interested in Informatics-related in-service training, and if so, they should propose their favorite topics. 190 out of all responding 400 teachers nominated 470 proposals.

Fig. 6. Aggregated proposals for in-service training

A further qualitative analysis has lead to a first attempt to categorize and sharpen the view on software tools in terms of application and development software.

Table 1. In-service training: Allocation of teachers' proposals

	Concrete products	Specific activities and general tools	
Application software	99	64	35%
Development software	103	55	34%
Tool independent concepts and other general topics	149		31%

It can be inferred that Informatics teachers often think in terms of concrete software products. This finding correlates considerably with experiences related to in-service courses for teachers[4]. Courses where concrete "products" are offered are much more accepted than "abstract" general topics.

4 In-service training and further education are for Austrian teachers at secondary academic schools not obligatory. In the past and dependent from an attractive offer, especially Informatics teachers showed much interest in in-service training. There is evidence that this interest is decreasing.

3 Teaching Tool-based Skills and Knowledge

The ways in which teachers cope themselves with software tools, and what is more important, how they deal with them in different classroom settings can be well mapped by Kolb's learning style model [7] as depicted in Fig. 7. When thinking of teaching basic IT-skills in the context of standard software, many didacts complain about mere product training and teaching "pushing the buttons", and thereby disregarding the underlying concepts of the tools. In [8,9,10] different methodical approaches in teaching text processing are described. The authors address the task of imparting practical skills combined with theoretical underpinnings and thus traversing Kolb's model at least to "abstract conceptualisation".

A French study about spreadsheet skills and knowledge [10] revealed considerable deficits among junior high school students, due to minimal training at one stage, whereby students do not master even basic principles of software interface, not to mention basic concepts as variables, data types and functions. The main finding of that study - "occasional use of software is not sufficient" - is remarkably redolent of the old but proved saying "practice makes perfect".

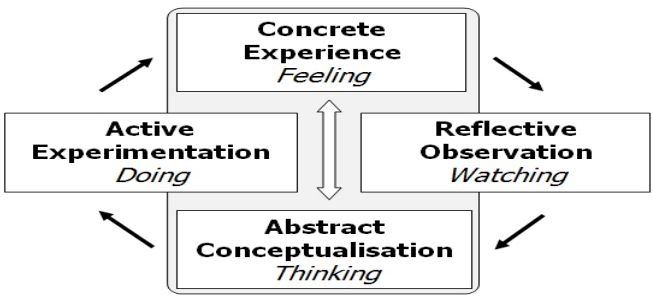


Fig. 7. Kolb's learning styles

However, as Mittermeir [11] insinuates in the analogy calculating: mathematics = ICT : Informatics, teaching software tools should not be overloaded by formal and abstract concepts especially for early age groups. In primary schools nobody would think of introducing simple counting by referring to Peano axioms. We have to consider this especially in case of teaching and training ICT-skills with regard to standard software, which usually takes place in lower secondary education. Imparting abstract and formal concepts in viewing at standard software through an object oriented lens [cmp. 8] is definitely a viable option in upper secondary education. However, at lower secondary level it is a subject for debate.

The following assertion, found at [12], deserves a deeper reflection: "*A child does not discover the world by learning abstract rules. Instead it learns by looking at concrete examples. An example contains the rules as well. In contrast to rules, the recognition of examples can be based on tangible reality. The knowledge extracted from an example serves as a pattern that is used to remember facts and to construct new solutions. When grown-ups are about to learn something or have to apply unknown tools, they are put into a child's position again. They will favor concrete examples over abstract rules. The rules will happily be generated automatically, for this is how the brain works.*"

At first sight, this contradicts with Kolb's model. But thinking about this model from the learner's and not from the teacher's perspective, concept building by concrete examples has to be regarded as a promising option.

Using software tools in combination with understanding their underlying concepts must be seen in a wider context of the practice-theory issue. Having Bloom's taxonomy [13] in mind, practical activity and basic knowledge about a particular software tool build the basis of this model. A more comprehensive theoretical body of knowledge about understanding and applying software tools address higher cognitive levels in Bloom's pyramid.

Hartmann [14] presents a further model addressing the practice-theory issue. Accordingly, in classroom settings a clear distinction

between theory and practice is proposed. This model can be applied even for skills training with concrete standard software. For example, with regard to word processing, the advanced subject matters "format-templates" or "form letters" should be taught separately from practicing and exercising with the concrete tool. This leads to the versatile model of a matrix, which can be applied not only to software tools, but also to real-world environments.

Table 2. General model of tools

	Concepts	Practical realization
Tools	What is product independent with regard to tools? Typical tasks and procedures	How are tool concepts realized with a concrete product?
Objects	What is product independent with regard to the corresponding objects? Attributes, categories	How are object-concepts realized in concrete object types?

Further, teachers should not only consider the difference between concrete practicing and abstract concepts, but also be aware of the strict distinction between particular software tools and their associated objects.

In [18, pp. 147-149] convincing examples (Picture editing, E-Mail, operating systems, algorithms and data structures) illustrate the power of this model. Another concrete example, publishing websites on the internet, is given in the table below.

Table 3. Special software tool model: Publishing websites

	Concepts	Practical realization
Software tools	(Web)editors, client-server-support (FTP), CMS, Webserver	MS-Word, Dreamweaver CMSimple, Typo 3, Joomla Apache-Server
Objects	protocols, textfiles, documents	TCP/IP, HTTP HTML, CSS

Being aware of inherent shortcomings of all models as miniature representations of reality, they are helpful in reducing complexity in didactics issues though. A deep understanding of these models by the learner means that he/she has to be a reflective practitioner [15] and competent user. This advanced state of proficiency can be best reached by extending teaching methods. Therefore an appropriate blend of problem-oriented, task-oriented, menu-oriented, function oriented, concept oriented and abstract-oriented approaches, based on contextual and situational teachers' didactic skills, is necessary [16].

4 Classification and Criteria of Software Tools

At first sight and illustrated in chapter 2, we face a plethora of software tools in Informatics teaching in Austrian higher secondary education. Due to the complexity and versatility of some software tools, their distinct assignment into classification schemes is a demanding task. Below, a first approach to classify the wide range of used and taught software is attempted.

Table 4. Classification scheme for software tools (used in Informatics lessons)

Dimension	Domain	Examples and remarks
Scope	Is it an application, development, simulation, creativity, gaming or authoring tool?	The "classics" standard software and programming languages are currently extended by large frameworks as Java, .Net technologies, APIs on the one hand and small educational tools on the other
Didactics	Which Informatics concepts does the tool cover?	Invisible to the user, this addresses, "fundamental ideas" as algorithms, file and data types, client-server principle, object orientation and functional modeling.
Appropriateness	In which age group should the tool be used?	As toys, books and computer games in general, also software tools are suitable for various age groups.
Functionality	How many features are supported?	Visible to the user, but often not recognized. Many tools suffer from "featuritis".
Complexity	How many application areas are covered?	Standardsoftware, as the trademarks Excel and Flash, are application and programming tools. Scratch, for instance, is a painting-, creative- and programming tool.
Ergonomics	Is the tool "easy-to-use" and neatly arranged?	Luckily, we observe a trend to better "usability" and standardization. This issue relates to a high degree to habituation.
Legality	Commercial software or Open source?	This important aspect addresses the pirate copies. This issue must still be considered as a legal gray zone.
Costs	Are there special license and pricing conditions?	For educational purposes (in order to avoid legal problems) open source could be the choice in the future.

Dimension	Domain	Examples and remarks
History	For how long has the tool been used? Is it a "day fly"?	The more than twenty years old computer language "Turbo Pascal" is still used in some Informatics lessons …
Technicality	Which OS does the tool support? Are there stable releases?	Windows or Linux is not only a "question of faith". Exotic software could make trouble.
Standards	Does the tool serve basic standard(s) and established formats?	The "browser war" between Internet Explorer or Mozilla Firefox and complying with standards is permanently a matter of concern. Are open standards supported?
Singularity	Are there equivalent products available?	Especially many web 2.0 tools are interchangeable at will. A market adjustment is needed urgently.
Coverage	Where/how is the tool used?	Is the software proprietary? Is it used in many countries? Is it used also commercially?
Locality	Is it a stand alone -, network or web 2.0 application?	"Cloud computing" is a current neologism and trend in computing with high expectations.

In view of the ongoing proliferation of software tools especially of that available on the web (in the cloud), this classification scheme may serve as useful orientation for categorizing appropriate tools for Informatics teaching.

Two of these criteria, costs and locality, will affect the organizational setting of Informatics education rather than its quality. Obviously, currently we face the shift to increased educational use of open source software, and moreover, there is a remarkable dynamics in web 2.0 applications. The transition from WWW to the WWC (world wide computer), as Nicholas Carr predicts, can be expressed by the neologism "cloud computing" and is currently associated with high expectations.

On closer examination, however, locally installed standard software[5] is still dominating Informatics lessons, especially at lower

5 In recent years, all federal schools in Austria were centrally equipped with Microsoft Windows and Office Software. However, the future about using mainly

secondary and the beginning of upper secondary level. Regarding these complex tools, Pareto's principle, also known as the 20-80 rule, can be applied in two respects. First, just a few software tools and products cover a wide range of Informatics teaching, and many individual and proprietary tools are applied for the rest. Second, the functionality of such software is used only partially and, in typical classroom settings, it is far from being exhausted. However, even with a restricted set of features a wide range of tasks can be accomplished. For example, MS Excel with the embedded language VBA must be considered as an application and development tool, although many teachers are not aware of that. This raises a fundamental didactical issue and methodological question which has to be decided individually by the teacher: Is it more appropriate to exploit the full conceptual potential of a software tool or should the students get acquainted with different special tools instead? This is an interesting topic for future research.

5 Concluding remarks

Compressing the quasi infinite spatial of software tools related to Informatics teaching into a finite paper might appear as an outsized challenge. Accordingly, every attempt to accomplish this task can not raise the claim of completeness. However, in view of the enormous influence which software tools exert in everyday Informatics lessons, it is a worthwhile undertaking.

Beginning with a glimpse on software tool usage in Austrian schools, the focus in this paper changed to didactic issues. All tools have particular purposes and never should be an end in itself. The question of teaching tool skills and competences, together with the combination of profound tool knowledge including its underlying concepts, has been discussed. This should be still a matter of concern for future research. Finally, a classification scheme was proposed in order to provide orientation and guidance for the plethora of current software tools.

---

commercial software at schools is uncertain.

Antoine de Saint-Exupery is said to have remarked, "Technology develops from the primitive to the complicated to the simple." In case of so many different software tools, it is evident that we will stay in the state of complicatedness still for a while.

"Men have become the tools of their tools." This quote from Henry Thoreau who lived in the 19th century should cause more worry…

References

1. Hubwieser, P.: Didaktik der Informatik. Springer Verlag, Berlin, 2003
2. Micheuz, P.: 20 Years of Computers and Informatics in Austrian's Secondary Academic Schools. In From Computer Literacy to Informatics Fundamentals, edited by R. Mittermeir, Springer, Berlin, 2005, pp 20 – 31
3. Micheuz P.: Some Findings on Informatics Education in Austrian Academic Secondary Schools, in: Informatics in Education, Journal, Lithuanian Academy of Sciences, Vilnius, 2008
4. Micheuz, P.: Informatics Education at Austria's Lower Secondary Schools between Autonomy and Standards. In The Bridge between Using and Understandig Computers, edited by R. Mittermeir, Springer, Berlin, 2006, pp 189 – 198
5. Haider G.: Schule und Computer, Österreichischer Studienverlag, 1994
6. Micheuz P.: Zahlen, Daten, Fakten zum Informatikunterricht an den Gymnasien Östereichs. In: Proceedings of 13.INFOS: Fachtagung „Informatik und Schule INFOS 2009.
7. Kolb D.A.: Experemental Learning: Experience as the source of learning and development. Englewood Cliffs. New York, Prentice Hall, 1984
8. Voss S.: Informatics Models in Vocational Training for Teaching Standard Software. In Mittermeir R. (ed.) From Computer Literacy to Informatics Fundamentals, Springer, Berlin, 2005, pp 145 –155
9. Schulte C.: Duality Reconstruction – Teaching Digital Artefacts from a Sociotechnical Perspective. In: Mittermeir R. (ed.) Informatics Education – Supporting Computational Thinking, edited by R., Springer, Berlin, 2008, pp 110 –121
10. Tort F.:, Spreadsheet Knowledge and Skills of French Secondary School Students. In: Mittermeir R. (ed.) Informatics Education – Supporting Computational Thinking, edited by R., Springer, Berlin, 2008, pp 110 – 121

11. Mittermeir R.: Was ist Schulinformatik? Rechnen:Mathematik = ??:Informatik in Donhauser D., Reiter A.(eds): ME 2001, ÖVE Schriftenreihe Nr. 26, pp. 3-13.
12. Kühne T.:, A Functional Pattern System for Object-Oriented Design (1999), http://www.mm.informatik.tu-darmstadt.de/~kuehne/tfps/fps-sans-escher.pdf (accessed 2009-10-20)
13. Krathwohl, D. R., Bloom, B. S., & Masia, B. B. (1973). Taxonomy of Educational Objectives, the Classification of Educational Goals. Handbook II: Affective Domain. New York: David McKay Co., Inc.
14. Hartmann W., Näf M.: Reichert R., Informatikunterricht planen und durchführen, Springer, Berlin, 2006
15. Schön D. A.: The Reflective Practitioner, Temple Smith, London, 1983.
16. Csiki N., Zsako L. ICT Teaching Methods – Applications. In: Mittermeir R., Syslo M. (eds.) Informatics Education contributing across the curriculum, Faculty of Mathematics Computer Science, Nicolaus Copernicu University, Torun, 2008, pp. 47-53

A Reflective Practitioner's Perspective on Computer Science Teacher Preparation

Noa Ragonis[1,2], Orit Hazzan[1]

[1] Department of Education in Technology and Science,
Technion – Israel Institute of Technology, Haifa, 32000, Israel

{noarag, oritha}@technion.ac.il

[2] School of Education, Beit Berl College, Doar Beit Berl, 44905, Israel

noarag@beitberl.ac.il

Our question then is not so much whether to reflect as what kind of reflection is most likely to help us get unstuck. (Schön, 1983, p. 280)

Abstract. The paper presents a research aimed at examining reflective processes carried out by prospective computer science (CS) teachers. The reflections were facilitated during a Method of Teaching CS course and during a tutoring process that was integrated into the course. In the paper, we present the research layout and its findings. Data analysis revealed that these reflective processes encourage the prospective CS teachers to function as reflective practitioners (Schön, 1983, 1987). Specifically, the prospective teachers exhibited eight viewpoints when reflecting, first, as *learners* in the Method of Teaching CS course, and second, as *teachers* while practicing teaching in the tutoring process. In light of the research findings, we discuss the importance of including reflective processes in CS teacher preparation programs. We suggest that reflective processes are especially important in CS education due to their potential contribution in promoting and improving problem-solving processes, which are central elements in CS.

Keywords: Computer science education, computer science teacher preparation, reflection, reflective practitioner, tutoring model, reflection in teaching, Method of Teaching CS course.

1 Introduction

In this paper we examine the reflective practitioner (RP) perspective [10], [11], in the context of computer science (CS) teacher preparation. The RP perspective guides professional people (architects, managers, musicians, educators, and others) to rethink and examine their professional creations during and after the accomplishment of the creation process. The working assumption is that such reflection enhances the proficiency and performance within such professions. In this spirit, we suggest that adopting the RP perspective may improve the CS prospective teachers' skills of learning and teaching problem solving, which are rooted in the core of the CS discipline, and are known to be both hard to learn and hard to teach [1], [4], [14].

The findings presented in this paper were revealed in a research whose objectives were to investigate the pedagogical contribution of integrating a tutoring model into the Method of Teaching CS course. Reflective processes emerged as one of the most central elements in promoting the prospective teachers' teaching skills and their awareness to pedagogical tools that can promote their pupils' problem-solving skills. Indeed, our research findings indicate that reflective processes improved the prospective CS teachers' teaching skills, and further, they used reflective processes to enhance their tutees' problem-solving skills. Specifically, we present eight viewpoints on reflection that the prospective CS teachers elicited, on the one hand, as *learners* in the Method of Teaching CS course, and on the other hand, as *teachers* while practicing teaching in the tutoring process.

The paper begins with a description of the concept of reflection and the RP perspective, followed by a description of the research framework. Then, we broadly present the research findings according to the eight above mentioned viewpoints on reflection. Finally, we

summarize and discuss further implications of applying an RP perspective in CS education.

2 The Reflective Practitioner Perspective

The two main books that present the reflective practitioner (RP) perspective are Schön's *The Reflective Practitioner* [10] and *Educating the Reflective Practitioner* [11]. The first book presents professions in which reflective thinking is inherent, such as architecture and management; the second book focuses on how *to educate* students of such professions to be RPs. In these books, Schön analyses the added advantages that can be obtained from continuously examining one's practice and one's thinking about that practice.

According to our literature review, one of the significant ways to acquire pedagogical-disciplinary knowledge, which also increases teachers' motivation, involves activities performed in actual teaching situations [5] and provide opportunities that guide the teacher towards reflective processes that address coping with learners' thinking [5], [16]. More recently, Khisty and Khisty [3] and Stroulia and Goel [15] discussed how to use reflection to teach problem-solving processes and Hazzan [2] applied this perspective to software engineering education. In a similar way, we hope to contribute by addressing the RP perspective with respect to CS teacher preparation, as is described in what follows.

In our research, the prospective teachers' reflective processes included reflection on learning activities performed in the Method of Teaching CS course and reflection on the tutoring activity, in which each student in the Method of Teaching CS course tutored a student in an introductory CS course, with a focus on problem-solving processes. The tutors' reflection included a teacher's perspective (their own perspective) and a learner's perspective (their tutees'), and reflection on feedbacks they received from the tutoring coordinator as well as from their fellow tutors. Most of the reflection, as is illustrated in the Findings section, led to the modification and

refinement of the tutors' actions both in the course and in the tutoring process. Furthermore, according to the prospective CS teachers' standpoints, the reflection itself, as well as the awareness to reflective processes, improved and developed their teaching skills. The prospective CS teachers stated that they intend to use reflection in their future professional work as CS teachers, first, by reflecting on their own teaching, and second, by leading their pupils to reflect during problem-solving processes in order to improve their solutions.

3 Research Framework

3.1 The Methods of Teaching Computer Science Course

Teacher preparation programs include a component that focuses on pedagogical content knowledge (PCK) which is what a teacher is required to know in order to teach a certain subject matter [12], [13]. In the context of the research described in this paper, this knowledge is acquired in the Methods of Teaching Computer Science course taught at the Technion's Department of Education in Technology and Science. The course aims at broadening the prospective CS teachers' PCK and sets the basis for the in-school practical training that takes place after it. The course syllabus where RP perspective was applied is presented in [9].

Course structure and population. The course consists of 112 hours of classes and training, divided into two semesters, each of which is devoted to different high school curriculum units. The course participants are prospective CS teachers who usually take the course during their third year of study (out of four).

Course objectives. The course's main objective is to construct a varied toolbox for the prospective CS teachers to use during their practicum and in their future work as CS teachers. The following objectives are related to reflective processes:

1) Expose the prospective CS teachers to difficulties encountered by learners when learning different topics from the CS curriculum;

2) Enable the prospective CS teachers to master pedagogical skills for teaching CS considering different kinds of learners;
3) Enable prospective CS teachers to master pedagogical tools for teaching CS, including the creation of a supportive and cooperative inquiry-based learning environment;
4) Expose the prospective CS teachers to a variety of CS teaching methods;
5) Expose the prospective CS teachers to the research conducted in CS education and to its application in teaching processes.

Teaching methods used in the Methods of Teaching CS course. The course illustrates how to actively apply a variety of teaching principles and methods in CS teaching and includes lectures, workshops for developing different teaching materials, hands-on experience with various software programs, practice of teaching in the course plenum, and many discussions.

Reflection as expressed in the Methods of Teaching CS course. Two main kinds of reflective processes were integrated in the course: (1) reflection that takes place during the accomplishment of the course assignments – as learners, and (2) reflection that accompanies the tutoring activity – as teachers. The two perspectives enable the prospective teachers to increase their awareness of the potential advantages of reflective processes. The following activities reflect the wide and deep attention given in the course in order to educate the prospective CS teachers to become RPs:

1) Exposure of the concept of reflection, including reflection before an action takes place, during its performance and after it has been completed;
2) Reflection on personal experience of each prospective CS teacher in the course. This includes, for example, self-reflection after presenting a teaching material prepared by the student to his or her peers in the course plenum, and reflection on feedback they receives from their peers and from the course instructor.

3) Scenario illustrations of how reflective process can enhance CS learners' problem-solving skills in general and in the context of high school CS classes in particular;
4) An ongoing reflection process takes place as part of the tutoring model;
5) At the end of each semester, each prospective CS teacher submits a written reflection on his or her own reflection processes during the course.

3.2 The Disciplinary Focus Tutoring (DFT) Model

One of the main activities carried out in the Methods of teaching CS course is the tutoring activity. The objective of the tutoring is to promote the prospective CS teachers' skills in guiding learners through problem-solving processes in CS. Since the tutoring focused on learning the discipline, it is referred to as Disciplinary Focus Tutoring (DFT). The innovation of the DFT method is that it focuses on the *tutor* rather than on the tutees, as do many other tutoring programs. The tutors are prospective CS teachers enrolled in the Methods of Teaching CS course, and the tutees are college or high school students enrolled in an introductory CS course. DFT is based on two developing levels: active imparting to the prospective CS teachers of pedagogical-disciplinary knowledge during the Methods of teaching CS course, and experiencing and applying this knowledge in actual teaching situations with their tutees as part of the tutoring process, as is elaborated upon below. Tutoring takes place in tutor-tutee pairs that meet for five sessions. Each tutor participates in two cycles of tutoring, one in each semester, with a different tutee in each cycle. During the sessions, the tutees raise difficulties they encountered while developing solutions to given problems, and the tutor guides the tutee through the problem-solving process. Tutoring is based on the tutor's identification of the tutee's difficulties, and the subsequent application of different teaching strategies to overcome such difficulties. The serial nature of the sessions enables the tutor to receive feedback on the knowledge the tutee acquired in previous sessions, thus providing the tutor with an

opportunity to reflect on his or her own teaching. Tutoring a different tutee in each cycle enables the tutor to compare and draw conclusions from the first cycle and apply them to the second cycle. A coordinator of the tutoring activity provides the tutors with ongoing support, within a coaching framework.

Tutor obligations. For each of the two tutoring cycles the tutors are required to:

1) Complete a feedback sheet for each tutoring session and submit it to the tutoring coordinator (see Table 1). The aim of the feedback sheet is to encourage and foster reflection by the tutors when functioning as teachers.
2) Hold individual meetings with the tutoring coordinator; one following the first tutoring session and one after completing each cycle of five tutoring sessions.
3) Present the Methods of Teaching CS course plenum with one episode from the tutoring process.
4) Complete a final summarizing feedback questionnaire.
5) Optional: Request the tutees to fill out a feedback sheet after each tutoring session. The idea of creating the tutee feedback sheet came from one of the tutors, and was complied jointly by the entire group of tutors. The decision whether or not to ask the tutee to fill it out was left to the tutors. The mere idea of constructing such a feedback form, however, can be seen as assimilation of the idea of reflection.

Table 1. Tutor feedback worksheet

A. General

1. Describe the subject of the session:
2. Describe the problem discussed:
3. Describe the course of the session:

B. Tutor Feedback

1. What concept/s do you think constituted a difficulty for the tutee?
2. Describe the difficulty/misconception you mentioned in your answer to Question 1.
3. What teaching tools did you use to help the student overcome the difficulty/misconception?
4. Did you use knowledge acquired in the Methods of Teaching Computer Science course or knowledge you acquired in another course?
5. What more would have helped you provide the necessary assistance? (Additional disciplinary knowledge, additional teaching knowledge, what kind of knowledge? which tools?)
6. If you could repeat this tutoring session, what would you do differently?
7. What is your personal feedback at this stage of the tutoring? (The nature of the communication between your tutee and yourself, the quality of support, your advancement of the tutee, your benefit from the tutoring, any difficulties, etc.)

Feedback from the course coordinator. Each submitted tutor feedback sheet was responded with a written feedback by the course coordinator. Such feedback included encouragement to pursue teaching and pedagogical processes applied in the session, calling of attention to specific processes that could be improved, guidance to alternative teaching tools to help the tutees overcome their obstacles, and recommendations for additional problems that could be presented to the tutees. In addition, the course coordinator employed an open door (and open email) policy, so that tutors could seek advice on any issue or concern, at any time.

Additional details about the DFT model are presented in [6], [7], [8].

3.3 Research Description

Research objectives. The main research objective was to investigate the contributions of the DFT model to the prospective CS teachers and to examine its practicability. In this paper we examine the following question in depth: How do prospective teachers become reflective practitioners?

Research population. The research population consisted of students who were enrolled in the Methods of Teaching Computer Science course described in Section 3.1. The data was collected during two different academic years: 2006-2007 and 2008-2009, with ten students each year.

Research methodology and tools. The research employed both qualitative and quantitative tools. Although we recognize that quantitative data are not significant in small groups, they are used in this exploration to support the qualitative findings. To validate the findings, the following research tools were used:

1) Interviews were held with the tutors following their first tutoring session with their respective tutees and again after the final session.

2) Tutoring Session Feedback Worksheets: Each tutor completed a Session Feedback Worksheet after each of the five tutoring sessions held each semester.
3) An overall evaluation questionnaire was completed by the tutors at the end of each semester, consisting of 36 position questions and 16 open questions. Several questions addressed the RP perspective employed in the course.
4) An overall evaluation questionnaire was completed by the tutees at the end of each semester, consisting of 13 position questions and 5 open questions.
5) A summarizing interview was held with 6 of the tutees.
6) Various homework assignments.
7) Researcher's diary.

4. Research Findings: Prospective CS Teachers as Reflective Practitioners

The results presented in this section emerged from a comprehensive data analysis of the data gathered by the various research tools. Specifically, we identified eight viewpoints (VP) on reflection that the prospective CS teachers exhibited. In what follows, we explain the essence of each viewpoint together with one or two illustrative excerpts taken from the different data gathering tools. To maintain students' privacy, they are identified by number, for example [St. 2].

VP1. Reflection on learning in previous CS courses

The content of the Methods of Teaching CS course, the variety of learning activities, and the tutoring process, all caused the prospective CS teachers to reflect on their past learning of CS concepts. For example:

- *I was never taught how to begin a problem-solving process, what the stages are.* [St. 15]
- *You learn while you teach... I am happy that I will be teaching recursion because I know that I will further my understanding of the concept.* [St. 3]

VP2. Reflection on learning in the Method of Teaching CS course

The prospective CS teachers reflected on the learning activities facilitated in the Methods of Teaching CS course, on their own performance while working on the activities, and on the teaching skills they acquired during the "learning to be a teacher" process. For example:

- *The skills you taught us of looking at things from the learner's perspective were very helpful indeed.* [St. 12]
- *Reading articles and writing summaries helped me learn how to express myself clearly and in comprehensible way.* [St. 10]

VP3. Reflection on teaching in the tutoring process

*The prospective CS teachers referred to their perfo*rmance as teachers in the tutoring process. This included pre-session reflection, in-session reflection, post-session reflection, and reflection on the overall tutoring activity. For example:

- Pre-session reflection - while preparing the tutoring session: *It contributes to my awareness and understanding of what teaching is. It highlights many new topics that should be paid attention to while teaching. In other words, taking the learners' condition into consideration and trying to predict the problems that will emerge during the teaching process, etc.* [St. 8]
- In-session reflection - during the tutoring session: *Indeed, during the lesson I noticed that I had digressed to a topic they had not yet learned. Therefore, it is really important to distinguish between topics and to consider previous knowledge.* [St. 4]
- Post-session reflection – on previous sessions – with criticism or satisfaction: *If I could repeat the tutoring session, I would give her additional time to think about the second part of the question and I would not give her such an obvious clue. I should have given her only a partial clue.* [St. 7]; *I do not regret that we began the solution in a specific way and then switched to another way. I think that it exposed him* [the tutee] *to different thinking processes and to* [the importance of] *examining ideas.* [St. 19]

- Reflection on the overall tutoring activity: *In summary, I recommend doing it – yes, yes, yes – otherwise we may not meet anyone who needs to have something explained to him or her.* [St. 9].

VP4. Reflection on the tutee's learning processes

During the tutoring sessions and after them, while writing the feedback worksheet, the tutors addressed their diagnosis of the way their tutee acquired the said knowledge. For example:

- Becoming familiar with the tutee's way of thinking and the tools that may help him or her overcome cognitive obstacles: *I was constantly reflecting. I would say something and think what she could be thinking, and I would hear what I was saying, and I would think whether I could explain it differently.* [St. 7]
- Becoming aware to the fact that what they had considered to be understood by the tutee, was not: *Even though we had talked about that topic and the tutee claimed that he understood it, I realize that since we did not practice it, the knowledge I had imparted to the tutee did not become his own knowledge.* [St. 17]

VP5. Reflection on the encouragement of tutees to reflect during problem-solving processes

The tutors' expressed that their understanding of the advantages of reflection in problem-solving process, led them to encourage their tutees to reflect as well, as an additional tool to support and improve their problem-solving processes. Specifically, they encouraged the tutees to reflect on their understanding and on their performance during problem-solving processes in the tutoring session itself. For example,

- *If she gives an incorrect solution, and I point that out to her and she does not see it, I tell her to go over her solution and explain what she did till she sees and understands herself, where her mistakes were, and corrects them herself.* [St. 1]

- *In this session I felt, several times, that I have not achieved my goals and I am not sure exactly how to do that* [achieve my goals] *.... At the end of the session, I felt I needed feedback from the tutee. He said that the session helped him understand the material. My feeling was not so good: I am only at the beginning of my way to effective teaching and there is still a lot of room for improvement.* [St. 4]

VP6. Reflection as a tool to envision the tutors' future as CS teachers

Based on their reflective processes, the tutors also envisioned their expectations as future school teachers. For example:

- *Reflection is very important to my future work as a teacher. Reflection enables to think critically about things I did and to better apply the same task the next time around.* [St. 19]
- *When we finished working on the topic of functions, I really felt that I could teach it well.* [St. 7]

VP7. Reflection on the future pupils' understanding

The tutors addressed the toolbox they intend to use to understand their future pupils' difficulties and how they will tailor different approaches to different pupils' ways of thinking. For example: *The tutoring in general is something else - it is important. Now I will enter the classroom and I will think that maybe the pupils think that it* [a specific CS topic] *is* [understood] *like this... and that* [another CS topic] *is* [understood] *like that...* [St. 7]

VP8. Meta reflection: Reflection on reflection processes

The prospective CS teachers also exhibited meta-cognitive skills while reflecting on their own reflection. This can be seen as the first rung of what Schön calls ladder of reflection (1987, p. 114). For example:

- *I love to reflect after any activity of any kind, it really helps me understand.* [St. 10]

- *After an unsuccessful attempt at explaining something, I took a moment and thought about what it was that I was trying to explain. I tried to recall previous explanations that I had given. I actually used previous reflections, and chose an example that was based on my previous experience.* [St. 16]

The prospective CS teachers were also asked to address the reflective processes they underwent during the course in the overall evaluation questionnaire. Table 2 presents their responses with respect the RP perspective they employed in the course (on a 1-7 scale: 1 - low, 7 - high). As can be seen, results numerically support the appreciation the perspective CS teachers expressed towards reflection processes as presented above.

Table 2. Contribution of reflection processes to DFT, Tutors (N=16*)

Question	Average (SD)
Dealing with the different aspects of reflective thinking contributed to my learning.	5.88 (1.00)
I think that dealing with different aspects of reflective thinking will contribute to my work in the future.	6.26 (0.95)
It is important to complete a reflective report after each tutoring session.	6.39 (1.00)

* At the time of writing this paper, only six of the ten tutors had finished their tutoring activity for the 2008-2009 academic year.

5 Summary

The working assumption of the research described in this paper is that reflective processes are important as part of CS problem-solving processes. Therefore, it is important to apply it in pre-service CS training and to provide prospective CS teachers with tools to encourage their future pupils to do so. This perspective is derived from Schön's reflective practitioner framework for professional work

[10], [11]. Clearly, the RP perspective is suitable for teaching processes in general; in our work, we applied it to CS teacher preparation programs.

We presented the context in which the RP perspective was introduced, both in the Methods of Teaching CS course and in the tutoring model applied in the course. We showed advantages gained from combining the theoretical aspect of RP with its significant practicing in the course and in the tutoring activity. Our data analysis indicates that the prospective CS teachers adopted the RP perspective as part of their behavior, their learning, and their tutoring. Moreover, the prospective teachers assimilated the RP point of view into their professional development as future CS teachers. Specifically, the prospective CS teachers highlighted eight viewpoints of RP that address their own learning, their own teaching processes and their learners' ways of thinking (tutees at this stage and pupils in their future work at school).

As can be seen, in most cases the prospective CS teachers used reflection as a tool to improve their teaching skills that in turn supports their professional development as CS teachers. Indeed, this is, in fact, the essence of the RP perspective. In this spirit, we highlight our opinion that an RP perspective in CS education is especially important due to its potential contribution to promoting and improving problem-solving processes which are central elements in CS.

References

1. Ericson, B., Armoni, M., Gal-Ezer, J., Seehorn, D., Stephenson, C., Trees, F., CSTA Teacher Certification Task Force: Ensuring Exemplary Teaching in an Essential Discipline: Addressing the Crisis in Computer Science Teacher Certification. New-York: ACM Press (2008)
2. Hazzan, O.: The reflective practitioner perspective in software engineering education, The Journal of Systems and Software 63(3), pp. 161--171 (2002)

3. Khisty, C.J., Khisty, L.L.: Reflection in Problem Solving and Design. Journal of Professional Issues in Engineering Education and Practice, 118(3), pp. 234--239 (1992)
4. Kolczyk, E.: Algorithm – Fundamental Concept in Preparing Informatics Teachers. In R.T. Mittermeir & M.M. Syslo (eds.) Informatics Education - Supporting Computational Thinking, Lecture Notes in Computer Science 5090, ISSEP 2008 (pp. 265-271). Germany, Berlin/Heidelberg: Springer (2008)
5. Putnam, R.T., Borko, H.: What do new views of knowledge and thinking have to say about research on teacher learning? Educational Researcher, 29(1), 4--15 (2000)
6. Ragonis, N., Hazzan, O.: Integrating a Tutoring Model into the Training of Prospective Computer Science Teachers. Journal of Computers in Mathematics and Science Teaching. 28(3), pp. 309--339 (2009)
7. Ragonis, N., Hazzan, O.: A tutoring model for promoting the pedagogical-disciplinary skills of prospective teachers. Mentoring & Tutoring: Partnership in Learning, 17(1), pp. 50--65 (2009)
8. Ragonis, N., Hazzan, O.: Tutoring model for promoting teaching skills of computer science prospective teachers. ACM SIGCSE 40(3), 276--280 (2008)
9. Ragonis, N., Hazzan, O.: Disciplinary-pedagogical teacher preparation for pre-service computer science teachers: Rationale and implementation. In R.T. Mittermeir & M.M. Syslo (eds.) Informatics Education - Supporting Computational Thinking, Lecture Notes in Computer Science 5090, ISSEP 2008, Germany, Berlin/Heidelberg: Springer, 253--264 (2008)
10. Schön, D.A.: The Reflective Practitioner. BasicBooks (1983)
11. Schön, D.A.: Educating the Reflective Practitioner: Towards a New Design for Teaching and Learning in The Profession. San Francisco: Jossey-Bass (1987)
12. Shulman, L.S.: Those who understand: knowledge growth in teaching. J. Educational Teacher, 15(2), pp. 4--14 (1986)
13. Shulman, L.S.: Reconnecting foundations to the substance of teacher education. Teachers College Record, 91(3), pp. 300--310 (1990)
14. Stephenson, C., Gal-Ezer, J., Haberman, B., Verno, A.: The new educational imperative: Improving high school computer science education. Final report of the CSTA Curriculum Improvement Task Force (2005) http://csta.acm.org/Publications/White_Paper07_06.pdf [2009, July]

15. Stroulia, E., Goel, A.K.: Learning problem-solving concepts by reflecting on problem solving. In: Proceedings of the European Conference on Machine Learning on Machine Learning (Catania, Italy). F. Bergadano and L. De Raedt (eds.), 287--306. Springer-Verlag New York, Secaucus, NJ. (1994)
16. Wilson, S.M., Berne, J.: Teacher learning and the acquisition of professional knowledge. Review of Research in Education, 24, 173--209 (1999)

The Development of a Regional CS Competition

Ralf Romeike, Andreas Schwill

Didaktik der Informatik
Universität Potsdam
August-Bebel-Str. 89
14482 Potsdam – Germany

{romeike,schwill}@cs.uni-potsdam.de

Abstract. Since 1998 our working group organizes the annual regional computer science contest for students of secondary schools in the Federal State of Brandenburg in Germany. Several times the competition has been realigned and conceptually changed. In this paper we report about reflections on design and arrangement of the contest and give an account of the respective experiences with different models.

1 Introduction

Promoting interest in CS and trying to attract pupils to choose CS as a subject of study in university is a long-term goal of CS educators. Competitions in CS are reported to be a good way to reach interested students by allowing them to experience the "world of CS" outside what they learn in school concurrently (cp. [1, 2]).

Since 1998 the didactics of computer science group at the University of Potsdam organizes the annual regional computer science contest for students of secondary schools (grades 9-13) in the Federal State of Brandenburg in Germany[1]. Started in 1997 as an open contest without any thematic specifications the contest in 1998, in 2003 and again in 2006 has been realigned and conceptually

1 The contest is co-organized by BLIS, a small non-profit society that manages regional contests in science on behalf of the ministry of education in Brandenburg.

changed and is now to a large extent based on closed problems. In this paper we report about our reflections on design and arrangement of a regional CS contest in general and give an account of the respective experiences with different models of the Brandenburg contest.

In part 2 we analyze factors that need to be taken into consideration when establishing a contest in respect to existing contests in the field in Germany. In chapters 3 to 5 experiences, problems, actions and reflections of the three stages in the development of the Brandenburg CS contest are described. In Chapter 6 the overall experiences are summarized and discussed in regard to the initially stated intentions.

2 Objectives for Establishing a Contest

When establishing a computer science contest one has to take into account the objectives and the intended group of participants. Generally there are two partially coupled parameters (Fig. 1):

The first parameter concerns the intended target group. One may attempt a *broad effect* via the contest, i.e. motivate an as large as possible number of students to pursue computer science. The goal is to make the concerns of computer science accessible to a larger public, and at the same time raise the general level from juveniles with respect to computer science problems and their solutions. On the other hand one may emphasize *excellence* by posing problems which lie far beyond the abilities of an average computer science student and in particular juveniles without access to regular computer science lessons, and thus acquiring and selecting gifted students.

The second parameter deals with the type of problems and tasks used in the contest. Either one organizes an *open* contest with only very general or even without guidelines concerning the subject of contributions or one poses *closed* problems which are to be solved by the participants within a certain framework and limited time.

Along with a recognizable and memorable name brand a profile of the contest will develop that is defined for future participants by a choice of subjects, problems and attractive prizes. General awareness of the competition (publicity) will occur through competition organization, award ceremony, and the presentation of contributions.

Fig. 1. Estimated market positions of major relevant contests in Germany

A clever tuning of these three elements, *target group*, *type of problems*, *name brand* and their relationship towards creating a memorable profile will enable a long term positioning of the contest on the "market" generating permanent demand by participants. However, one has to keep in mind that several attractive "market positions" are already occupied by well-established contests in Germany and, thus, are less suitable for the Brandenburg contest (Fig. 1). We do not include the International Olympiads here since they are not open and participation is by invitation only.

Based on these considerations in 1998 the second Brandenburg CS contest was started and reorganized in the succeeding years due to the experiences gained, which will be described below.

3 Stage 1: Introduction of the Brandenburg CS Contest: 1998-2002

A major influence on the focus of the new competition originated with the intended target group: In 1998 neither in the state of Brandenburg nor in the rest of Germany was computer science an established subject in school with respect to subject content and organization. Therefore, the contest in Brandenburg should predominantly strive to spread problems and ways of thinking in computer science in the sense of a *broad effect* and help to substantiate the subject of computer science in school. Excellence was thought to be subordinate to the point when the subject of computer science is stable enough and students can be expected to have adequately acquired the many fundamentals in school necessary for excellence.

So far no computer science contest has succeeded in acquiring a reasonable amount of female participants. This issue was treated in particular with supportive activities and by the selection of problems to be solved. 20% female participation was thought to be a great success and a feature of the contest that would guarantee national attention. The Federal Contest of Computer Science attracted an average of only 2-3% girls, slowly increasing to 5,5% in 2006, 6,5% in 2007 and to 10,5% in 2008 (cp. [3-5]).[2]

Problems: After deciding on a broad effect one is more or less committed to closed problems. The reasons for this will be explained and illustrated through experiences, the second author has gained as organizer of the Federal Contest of Computer Science and which have been reported in [6-8]. The first three contests had been announced without any subject or problem specifications (#participants=113, 221 resp. 150 in the three contests). Thereafter the contest format was changed to closed problems (#participants=700-1200 each year in the first round) for the following two reasons: First, the contest was active in a market segment that was already successfully

2 Interestingly, the German Beaver contest made it to attract tremendous 41% of girls in 2008 [9].

occupied by the renowned contest "Jugend forscht" (young researchers) and its math/CS branch. Second, the CS contest did not have a sufficiently broad effect because an open contest does not reduce – as one might expect – the access barrier but on the contrary sets a hurdle. A project is more difficult for a potential participant if he or she does not know the problem and thus has to start with a problem-finding phase. Also, it is troublesome if a participant does not know what is expected from him or her and if he or she has no idea about the quality of the competing contributions. Furthermore – this was an assertion for the first three contests that has been partially confirmed by the participants – participants submitted contributions which were already available in more or less polished form and only had to be adapted according to the contest regulations. With this advantage in development the qualitative level of the contributions increases, but new participants usually cannot catch up. The well-meaning approach to increasing the number of participants through an open subject format seemed to fail.

After these considerations we had to specify the type of tasks used for the contest. In order not to conflict with the well-established Federal Contest of CS closed problems were rejected. We decided to use one task each year that consists of a more or less open problem or everyday-life situation. These had to be treated by participants from two different perspectives (called A and B in the following) and required different competencies while avoiding the above-mentioned negative effects of open contests. With respect to Fig. 1 we did not only occupy a single "market position" but covered a larger area hoping to address more participants.

Perspective A consisted of analytic-descriptive work with possibly detailed solutions and scenarios using a computer. It could be managed without any special knowledge and in particular without programming knowledge. The core computer science part was between 30% and 50%. Perspective B required a detailed implementation of a solution scenario developed in A or of different aspects thereof. This task required detailed knowledge of computer

science, in particular programming experience, similar to other contests.

A contest contribution consisted either of a solution solely to A or to both perspectives. Contributions to just B were not possible. What did we attempt by that? On one hand we expected that A also addressed students who did not take computer science lessons but who were interested in computer science and had a certain creativity. It is known that girls belong to this group of students less interested in technical aspects (cp. [10-12]). Furthermore, computer science – often recognized as the science of computers – might now be considered as a science that comprises more than just technical expertise. On the other hand the proposed type of problems motivates pair work where ideally a student interested in CS without programming experience mainly deals with A while his/her experienced partner mainly deals with B. The teamwork, typical for computer science at large, which is potentially involved with this approach, would differentiate this contest from other approaches.

Unfortunately, most of the objectives mentioned earlier have not been achieved. Between 1998 and 2002 the contests dealt with the following subjects from which an arbitrary problem had to be addressed:

1998: Computer science and language
1999: Computer science and history / the history of computer science
2000: Computer science and traffic
2001: Computer science and criminality
2002: Computer science and arts

The title of the 2001 winners' contribution gives an idea about the kind of contributions that were submitted: *A web-based file of criminals*.

While sometimes up to 100 students participated, the number of contributions was always less or equal to 5, though some were very extensive. Furthermore, only a small number of schools and almost always the same ones were creative enough to find a problem for the given subject and to organize a project in order to contribute to the

contest. Participation was primarily based on the interest and activity of the teachers.

After some detailed discussion the contest was realigned in 2003.

4 Stage 2: Realignment of the Contest: 2003-2005

Until recently it took a teacher's initiative to form and organize a project in school in order to participate. Now students should gain the opportunity to participate directly and without a major influence from the teacher. We changed this because there was feedback from students that showed that students wished to participate when their teachers did not manage to organize a respective project or find enough group members. Accordingly we expected a larger response to our call for participation after this change. The teacher's influence was now limited to selecting up to two students from his or her school and nominating them for participation in the contest. This procedure, looking superfluous at first sight, ensures that students can approach teachers who might not follow announcements, registration deadlines etc. The contestants, as quasi "official" delegates, now can draw the attention of teachers, the school, and to the subject of CS itself.

Furthermore, we almost completely abandoned the project-oriented approach where projects had to be defined and accomplished over a longer period of time during lessons. Therefore the contest was organized on a single day at the University of Potsdam and was divided into three sections where we integrated experiences from the final round of the Federal Contest of CS. First, students had to pass a 15 minute oral examination on computer science problems and ways of thinking. Secondly, students worked on a larger relatively open problem in groups of 4-5 for about 3 hours while being observed by a judge. Here they had to show how to apply different CS techniques and work efficiently in a team. A group consisted of students of almost equal age. The contest day finished with a plenary session where all students had to present their results. The experts of the jury monitored all discussions and solution strategies of

participants and evaluated individual and group performances as well as their abilities to work in a team.

In the third year we also added a multiple-choice-test between the first and the second part. Due to the different competencies that are required in the examination procedure we felt much more confident of finding the best students.

We implemented this approach for the contest in 2003-2005 and selected the following group tasks:

2003: Robot soccer

2004: E-Commerce for services

2005: Ubiquitous computing

We received about 30% more applications than places were available (32-40 participants, equals 8-10 groups).

The students enjoyed the group work very much, which gave them the opportunity to deeply discuss CS problems with their peers; an activity possibly missing from their normal school lessons.

From feedback of the participants we knew that while the first theme was very motivating the latter two were recognized as somewhat tedious. All in all the solutions presented by the teams were considered by the jury as often superficial, less substantiated and uncreative. This was probably due to the lack of profound CS knowledge and too much time spent concretizing the open problems.

Accordingly, we modified the contest in 2006 again in order to overcome these weaknesses.

5 Stage 3: Fine Tuning of the Contest: 2006-today

As a reaction to the aforementioned issues we replaced the more general open theme for the group work by a small set of 3-5 well-defined (artificial) problems that can be solved by computer science methods but often have no obvious relation to computer science. One of the problems is usually a bit easier than the others and serves as a warm-up. All these problems help to reduce the initial phase of problem analysis and understanding which were necessary for the open problems used earlier. We select these problems from problem

sections of books and journals; for some problems we know the solution in advance, for some we do not. The danger that some students incidentally are familiar with a problem we consider as very small.

Here is an example of a problem we used in the contest 2009:

Diagnosis of an infectious disease: Among p persons an infectious disease circulates. In order to separate the sick from the healthy persons we take blood samples from all p persons and test them for viruses. Obviously, by analysis of each single blood sample we can find out whether a person is ill or healthy. This requires p lengthy analyses.

We wish to accelerate this procedure. Instead of analyzing each single sample we join parts of several samples and analyze the mix. This analysis gives the information whether there is an ill person among the tested group or whether all persons of the group are healthy. By a suitable choice of groups we want to need less than p analyses. Is that possible or not?

---

6 Experiences, Reflections and Discussion

Participants: The participants were between grades 9 (around age 14) and 13 (around age 19). The number of female participants varied between 10% and 15%, which is a relatively high share, compared to other contests.

Problems and group work: Working in a group seems very motivating and interesting for the students. They learn from each other and find out what they know and do not know. Furthermore, they are introduced to computer science subjects previously not encountered. Some participants who have begun to study computer science report that they are confronted with similar problems during their studies they have faced during the contest. Some students profit from their competition experiences, preparing for the contests in the following years by closing gaps in their knowledge, participating several times and improving to gain higher positions.

Funding: The budget of the contest is approximately 1000 EUR funded by the Brandenburg Ministry of Education and covering all expenses. The awards for the best students (around 10 of the 30-40) are between 50 EUR for the first places and 20 EUR for the third places in addition to some computer science books donated by a local publisher. Although the financial prize sometimes barely covers the traveling expenses for Potsdam, it does not appear that students are making their decisions to participate dependent on the prize money. The fascinating event, meeting people with the same interests, as well as the possible honor of receiving a prize seems incentive enough.

Long-term impact and broad effect: Since the number of participants is limited to 32-40 the direct impact of the contest is assumed to be relatively small. However, we count more on the indirect effect:

- Around 20 schools are involved in the contest by delegating their best students to the contest. At least as many teachers are involved with the contest, some even train their students beforehand.
- Winners and their schools are frequently mentioned in local newspapers thus improving their reputations[3]. The subject of computer science is enhanced and attracts more attention.
- The best students get a (hopefully) positive impression of the University of Potsdam and its computer science department, which might attract them to study here. In fact we occasionally get feedback from students studying CS, however, we do not have exact figures for how many students study CS in Potsdam or elsewhere as a result of participation in the contest.

All in all we are fine with the current version of the contest and are looking forward to the future.

References

1. Dagiene, V.: Information Technology Contests – Introduction to Computer Science in an Attractive Way. Informatics in Education, 5 (1), 37–46 (2006).

3 The motivational benefit of such "fame" is also reported by Pohl [13].

2. Pohl., W.: Computer Science Contests in Germany. Olympiads in Informatics (2007) Vol. 1, 141-148.
3. Statistics of the 25th Federal Contest of Computer Science Germany: http://bwinf.de/fileadmin/templates/bwinf/presse/Statistik_25._BWINF.pdf
4. Statistics of the 26th Federal Contest of Computer Science Germany: http://bwinf.de/fileadmin/templates/bwinf/presse/Statistik_26._BWINF.pdf
5. Statistics of the 27th Federal Contest of Computer Science Germany: http://bwinf.de/fileadmin/templates/bwinf/presse/Statistik_27._BWINF_Runde_1.pdf
6. Claus, V., Schwill, A.: Informatikkenntnisse von Jugendlichen, untersucht am Beispiel der drei Bundeswettbewerbe Informatik. Informatik-Spektrum 6 (1986) 270-279, Springer.
7. Claus, V., Schwill, A.: Die Wechselwirkungen zwischen Problemstellung, Programmier-sprache und verwendeten Informatikmethoden am Beispiel der beiden Bundeswettbewerbe in Informatik. Informatik-Fachberichte Bd. 90 (1984) 87-91, Springer-Verlag.
8. Claus, V., Schwill, A.: Evaluating and improving informatic education in secondary schools by means of programming contests. Proc. of the 4th World Conference on Computers in Education (1985) 25-30, North-Holland Publishing Company.
9. http://www.informatik-biber.de/Archiv/informatik-biber-2008
10. Roberts, E. S., Kassianidou, M. und Irani, L. (2002): Encouraging Women in Computer Science. SIGCSE Bulletin., 34, 84-88.
11. American Association of University Women (2000): Tech-Savvy: Educating Girls in the New Computer Age. American Association of University Women, Washington, DC.
12. Guzdial, M. und Soloway, E. (2002): Teaching the Nintendo generation to program. Commun.ACM, 45 (4), 17-21.
13. Pohl, W.: Computer Science Contests for Secondary School Students: Approaches to Classification. Informatics in Education (2006) Vol. 5, No. 1, 125-132.

Modern Web Development in Schools

Lothar Schäfer, Hans-Stefan Siller and Florian Strasser

University of Salzburg
Department of Mathematics and Informatics Education
Hellbrunnerstr. 34
5020 Salzburg

lothar.schaefer@sbg.ac.at
hans-stefan.siller@sbg.ac.at
florian.strasser@sbg.ac.at

http://www.uni-salzburg.at/

Abstract. Modern web development is nearly untaught in schools, even though a paradigm shift has happened over the last years which made the knowledge of web programming and programming languages unnecessary. We picked up this trend and produced an online tutorial, in which we demonstrate the development of a website with the aid of TYPOlight, a Content Management System free of charge. To ensure the use of our tutorial in class, we also have prepared a USB flash drive, containing an Ubuntu Linux live operating system and an XAMPP web server. This system is flexible and can be applied to a number of environments. The students' task is to develop a website modelled on the one in the tutorial and thereby acquire skills in the use of the TYPOlight CMS on their own. By using different forms of representations throughout the tutorial and due to TYPOlight's usability this task can be easily accomplished.

1 Web Development Meets New Ideas

As today's websites are getting more and more complex, Web Content Management Systems (CMS) are quickly becoming essential tools for personal and professional web design. They provide features for quick and easy creation, administration and maintenance of websites from remote locations using simple and intuitive user interfaces that do not

require much knowledge of web programming. In order to teach Informatics in a modern and relevant manner it is necessary to take this recent trend into account.

Of course it is not possible to follow every trend in school teaching, but in this case we talk about a paradigm shift that must not be ignored. Up to now, the creation of websites is introduced with the concept of HTML - mostly in combination with Cascading Style Sheets (CSS) - and a development environment - commonly an editor - or by commercial software, e.g. Dreamweaver. Knowledge of the syntax of HTML and CSS is necessary, but is now made less important by Content Management Systems with focus on application competence. Since TYPOlight [10] is a free Content Management System that incorporates many useful features while being very user friendly, it is well suited for teaching the basic paradigms of complex web development.

We want to introduce a flexible new method of teaching Content Management Systems that takes into account the complexity of the subject matter and computer infrastructure in schools or educational institutions. In order to achieve these goals, a tutorial on TYPOlight (Fig. 1) that was built using only TYPOlight is provided. (Note: This tutorial has only been developed in German so far. Hence the following pictures are only available with a German text!)

The main purpose of this approach is to allow easy access to the tutorial as a website and to showcase the power of TYPOlight, motivating students to learn how to use it for their own projects. The tutorial uses different forms of representation and guides students through the process of building a new website from scratch by modelling it after the tutorial itself.

Using workstation computers to work with applications built mainly for server environments proves difficult because a variety of installations and configurations are required before students can start with their tasks. To eliminate this extra work, the tutorial is stored on a USB flash drive that contains its own bootable Linux operating system with all necessary server software and configurations already in place. This way all students get their own preconfigured environment and the teacher does not have to worry about problems that may arise during setting up a working web server or database on several computers. In addition to this, students are enabled to take their current work with them and use it on any computer they wish.

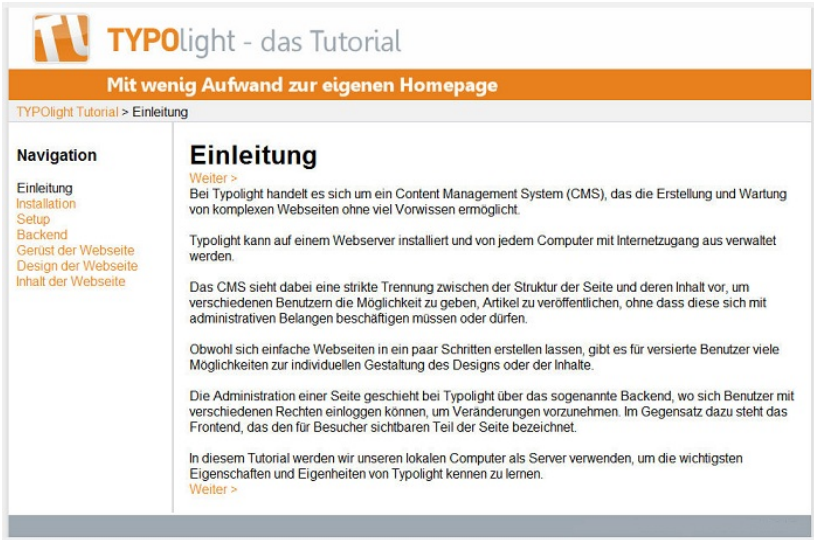


Fig. 1: Starting point of the tutorial

Of course, not all workstation computers can allow all users to boot from their own flash drive, because this poses a serious security risk regardless of any safety precautions in the host operating system. In order to make our concept work in this scenario, the flash drive also contains applications for use in Windows environments that operate in a manner similar to their Linux counterparts. This way every student should quickly end up with a working environment for viewing the tutorial and creating his or her own website.

This concept works for educating adults or teachers in much the same way as it does for schools - especially by aiming at the secondary level - because the only infrastructure required is a computer room, provided in most educational facilities nowadays. Participants can also take the flash drive with them and don't have to worry about losing their work.

The most obvious benefit of our approach here is the high level of portability of the described system. By using techniques such as computer independent operating systems and portable servers students can

also get a grasp of some possible future trends in informatics including thin clients and cloud computing. When using the provided tutorial in different locations and on different operating systems it soon becomes apparent that really the only thing required for it to work is processing power.

2 Technical Aspects

The USB flash drive contains one bootable partition with a Linux operating system that is ready for use once started. The other partition can be read by any standard Windows operating system and contains a folder with a preconfigured XAMPP web server that can be started as a service on the host machine. The downside of this method is that not every computer will have the exact same condition to start from, since some applications like web browsers may differ from machine to machine. Nevertheless, both possibilities feature equal functionality and user friendliness. The decision to use one setup or the other will mainly be influenced by the existing infrastructure. This feature set (bootable USB flash drive plus XAMPP web server) opens up many different ways of teaching TYPOlight in a wide variety of setups.

The tutorial on TYPOlight is the core component in teaching Web Content Management Systems as it tells students how to create a new website from scratch following an easy step-by-step instruction. For this purpose the running web server has two directories; one contains the tutorial itself and the other offers a fresh TYPOlight install. Since both directories are served publically by the web server, the teacher can watch the progress of any individual student from a central location and also provide help if needed. The teacher only needs to know the IP address of the respective student's server to use a web browser and take a look at the content of the students' website. With the appropriate website settings, it is even possible to develop the website in teacher - student teamwork.

2.1 Cyclic Processes

Due to TYPOlight's structure, cyclic processes may occur during the development of the website. Objects are generated and afterwards applied to already existing objects which are thereby refined. During

the preparation of the tutorial our first concern was to avoid such processes and find the most cycle free development process. Therefore, we first create for example all necessary modules before applying them onto the website's layout. This procedure is not mandatory and cannot always be implemented in practice.

What might look like a problem at first actually is the solution for dealing with a complex process like the development of a Website. The mentioned cycles make it possible for the developer to forget some aspect in his building process or just concentrate on another aspect and then finish his work afterwards with no problem at all.

The following illustrations (Fig. 2) show one of those cycles. At first a new layout is generated (1), only to be immediately edited (2). Within the layout modules are responsible for the site's content (3). Now a new module is created (4), edited (5) and inserted into the existing layout (6).

This cyclic procedure helps to steadily approach the solution and gives the user the possibility to add or change content whenever necessary. For this reason TYPOlight was our choice. First, it supports later additions or changes and second, it represents a stable and nearly tough training environment. You can make the students feel, that whatever step they take, it will in an iterative way always lead to the desired solution, i. e. the finished website. In addition to the introduction to modern web development, the teacher can also point out the strengths and advantages of iterative and recursive processes in specific examples.

2.2 Separation of Structure, Content and Layout

Another interesting aspect in the use of the TYPOlight CMS is the strict separation of structure, content and layout. Starting out by generating the websites structure, content and layout are not from importance at that time. They have to be dealt with later. So the students can easily see the advancement of separating those three fundamental aspects. If they already have some experience with HTML, this point can be further emphasized by the teacher, for instance by creating and displaying different layouts for the same webpage, comparing this with a conventional HTML approach.

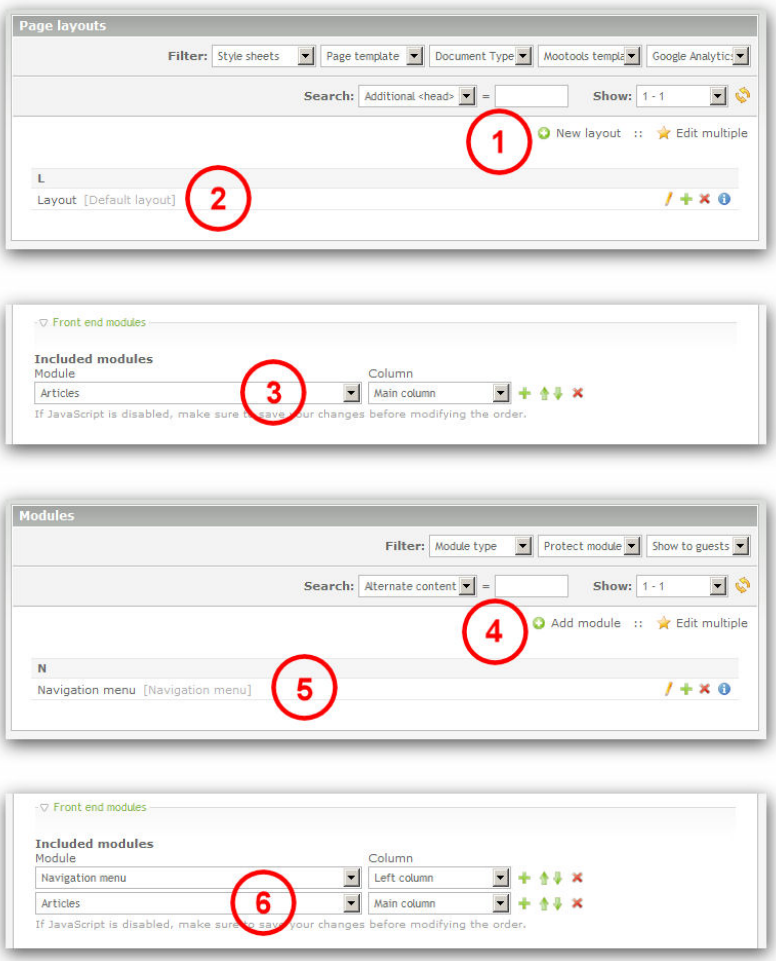


Fig. 2: Graphical illustration of cyclic process while working with the tutorial

The separation is also important if someone wants to test the possibilities of this CMS in advanced situations. Since the content is controlled with templates, it's possible to change the content without any knowledge of the actual layout, displaying the strengths of abstraction layers.

3 Using the Tutorial in Education

As already mentioned, TYPOlight is a Content Management System managed with a user interface, liberating the web developer from the well-known programming tasks. Therefore, the competencies in developing a website have shifted from programming and developing to using a single application and in this way have become accessible to a wider audience. Students using TYPOlight hence automatically take on the role of an application user. This, however, does not eliminate the need for knowledge of basic HTML or CSS syntax, as these skills are still required at some points in the course of teaching the CMS.

The idea of our teaching sequences is that the students acquire most of the new skills with the help of the tutorial while the teacher's role is that of coaching (cf. [3]). Students are construing their knowledg themselves as Papert [6] mentions it. Sutherland [8] shows the success of coaching in education through an empirical study. In our opinion, two teaching sequences result from the students' task to create a website. Each of them mainly focuses on the acquisition of responsibilities in using the system. At the same time topics that are not commonly discussed in class can be introduced in a practical way.

In the first teaching sequence the given USB flash drive can be used to boot the Ubuntu Linux operating system with a running web server. Alternatively, the XAMPP server located on the Windows partition of the USB flash drive can be started. Afterwards, the installed web browser Firefox is opened and the TYPOlight tutorial as well as the TYPOlight training environment are started by typing in the respective addresses or via preloaded bookmarks. Now the students are able to work in the familiar environment of a web browser, while in the first case the Ubuntu Linux operating system (possibly unfamiliar to the students) is running in the background. This is an excellent op-

portunity to reduce hindrances in using other operating systems than Windows.

The second teaching sequence does not begin with the use of TYPOlight itself, but with the setup of an environment for the use of TYPOlight. In this case it is possible to work with the Windows operating system as well as with Ubuntu Linux. During the setup routine the concepts of web servers and database servers can be discussed and demonstrated in class. The XAMPP server in use is exemplarily designed and deployable for that purpose. After having completed the setup in only a few steps, this teaching sequence is continued in the same manner as the first one.

At this point the teacher's role is that of a coach. The students mainly have to work with the tutorial autonomously - as mentioned in [2] - whereas the teacher only intervenes in case of problems or more detailed questions, as shown in [7]. This can be done either at the student's computer or via network access to the website's backend.

The tutorial can also be thought of as a starting point for further project-based teaching as quoted in [1], [2]. All basic skills and core competencies needed for developing an own website on any chosen topic will be acquired with the help of the tutorial.

3.1 Skills and concepts taught

Many skills, techniques and concepts can be learned by this method of teaching Web development in either direct or indirect manner, following the idea of the spiral principle. At first, students get a good feeling of abstraction layers in software as they learn to build websites without using any low level markup or programming languages. It seems as if the CMS could take care of all the hard work and leave only the creative and productive part to the designer. The impact of creative intelligence is enhanced (cf. [4], [9]).

But the students soon learn that customization beyond a certain point still requires knowledge of HTML syntax or even PHP code and database queries if they want to create new modules for TYPOlight to better fit their needs. Furthermore, the modular nature of the CMS with its strict separation of structure and content teaches some of the ideas of object orientation.

In the end, students will realize that tools such as TYPOlight can make some work unnecessary through abstraction while they still require a thorough understanding of the underlying concepts to be able to use them in a meaningful way.

3.2 Didactic Concept

The aim of maximum sustainability is reached through different approaches. At first, the entire process of developing a website is captured and split into small and consecutive sections. This enables the students to stop working at any time and to continue whenever they want to. The structure of this segmentation is available via the navigation menu. It is also possible to leaf through the tutorial like a book.

Furthermore, different forms of representation are applied throughout the entire tutorial. Nearly every section begins with a screencast including audio, explaining the next steps in the development process. Subsequently, the content of each video is displayed in shorthand text with the appropriate screenshots. This is useful if the students have forgotten some aspects or were not able to follow the screencast, because this way they can easily catch up on the content with the help of screenshots. The explanatory notes are kept short and simple on purpose in order to not disencourage the students by long text passages. Each section is concluded with questions and tasks. On the one hand they contain theoretical aspects on the other hand they provide tasks to work beyond the tutorial's extent and encourage to try out more complex functionality included in the Content Management System.

This approach not only provides a clear structure, it also follows the E-I-S principle [5] of holistic learning. The iconic level is addressed with the help of screencasts and screenshots. The explanatory notes appeal to the symbolic level, whereas the enactive level is covered by building the actual website.

Moreover, the iterative processes earlier mentioned represent a different and essential advantage. The sequence of necessary operations during the development process is often repeated, leading to a natural deepening of the acquired skills and thus strengthening the application competence.

4 Conclusion

Our tutorial gives the students an insight into modern web development. At this point it must be emphasized that this insight is strongly practice orientated and offers the students concrete and realistic job opportunities. The acquired knowledge and some motivation make it possible to create complex websites for clients thus entering the professional world.

Modern web development has overall become more complex over the years, but is at the same time easier for the user, since he needs less knowledge of web programming. If this trend continues, this knowledge will be even less required and a website will be developed and designed only by using a Content Management System.

At the same time the students get the chance of experiencing TYPOlight on their own, the teachers role becomes that of a coach, guiding the students in their learning process.

References

1. BM UKK: Grundsatzerlass zum Projektunterricht - Wiederverlautbarung der aktualisierten Fassung (engl.: Acceptilation on projects in education). Wien: Ministry of Education, 2001
2. BM UKK: Lehrplan Informatik AHS-Oberstufe. (engl: Syllabus for Computer Science for secondary schools). Wien: Ministry of Education, 2004
3. Collins, A.; Brown, J.S.; Newman, S.E.: Cognitive Apprenticeship: Teaching the crafts of reading, writing, and mathematics. In: Resnick, L.B. (Ed.): Knowing, learning, and instruction. Hillsdale, NJ: Erlbaum, 453-494
4. Funke, J.: Psychologie der Kreativitaet. In: Holm-Hadulla, R.M. (Ed.): Kreativitaet. Berlin: Springer, 2000, 283-300
5. Kautschitsch, H.: "Erfolgreiche" Bilder durch neue Medien (engl: Successfull pictures through New Media). In: Schriftenreihe der Mathematik. Band 23. Trends und Perspektiven (engl.: Scientific series for Mathematics. Vol. 23. Trends and Perspectives). Wien: Verlag Hoelder-Pichler-Tempsky, 1996, 191-197

6. Papert, S.: Mindstorms: Children, Computers and powerful ideas. All about LOGO., BasicBooks, A Division of HarperCollins Publishers, Inc.: New York, 1993

7. Siller, H.-St.; Maass, J.: Fussball EM mit Sportwetten (engl.: Football championship and Sports betting). In: Brinkmann, A.; Oldenburg, R. (Eds.): Materialien fuer einen realitaetsbezogenen Mathematikunterricht. Bd. 14 (engl.: Materials for real-life Mathematics in education. Vol. 14). Hildesheim: Franzbecker, 2009, 95-113

8. Sutherland, L.: Developing problem solving expertise: The impact of instruction in a question analysis strategy. In: Learning and Instruction., 12, 2002, 155-187

9. Wirth, J.; Klieme, E.: Computer-based assessment of problem solving competence. In: Assessment in Education., 10 (3), 2003, 329-345

10. http://www.typolight.org/ (last access: 20.08.2009)

Author Index