

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Roland T. Mittermeir (Ed.)

Informatics Education – The Bridge between Using and Understanding Computers

International Conference in Informatics in Secondary Schools –
Evolution and Perspectives, ISSEP 2006
Vilnius, Lithuania, November 7-11, 2006
Proceedings

Volume Editor

Roland T. Mittermeir
Institut für Informatik-Systeme
Universität Klagenfurt
9020 Klagenfurt, Austria
E-mail: roland@isys.uni-klu.ac.at

Library of Congress Control Number: 2006935052

CR Subject Classification (1998): K.3, K.4, J.1, K.8

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-48218-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-48218-5 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com

© Springer-Verlag Berlin Heidelberg 2006
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11915355 06/3142 5 4 3 2 1 0

Preface

Although the school system is subject to specific national regulations, didactical issues warrant discussion on an international level. This applies specifically to informatics didactics. In contrast to most other scientific disciplines, informatics undergoes substantial technical and scientific changes and shifts of paradigms even at the basic level taught in secondary school. Moreover, informatics education is under more stringent observation from parents, potential employers, and policy makers than other disciplines. It is considered to be a modern discipline. Hence, being well-educated in informatics seemingly ensures good job perspectives. Further, policy makers pay attention to informatics education, hoping that a young population well-educated in this modern technology will contribute to the future wealth of the nation. But are such high aspirations justified? What should school aim at in order to live up to such expectations?

ISSEP 2005, the 1st International Conference on Informatics in Secondary Schools – Evolution and Perspectives already showed that informatics teachers have to bridge a wide gap [1, 2]. On one hand, they have to show the inherent properties that informatics (or computer science) can contribute to general education. On the other hand, they are to make pupils computer literate. Under the constraint of limited time available for instruction, these different educational aims come into conflict.

Computer-supported teaching or eLearning is to be considered distinct from informatics education. However, in many countries, informatics teachers still have to support the eTeaching activities of their colleagues. They might even be the only ones to support eLearning. But even in situations where teachers of other subject areas are sufficiently computer literate to use computer support in their own courses, they will expect students to arrive already technically prepared by informatics courses.

Considering this spectrum, the program of the 2nd International Conference on Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2006, was mainly structured into discussions on what and how to teach. Those aiming at educating “informatics proper” by showing the beauty of the discipline, hoping to create interest in a later professional career in computing, will give answers different from the opinion of those who want to familiarize pupils with the basics of ICT in order to achieve computer literacy for the young generation. Addressing eLearning aspects as seen from the perspective of informatics didactics are another only moderately related set of issues. This spread of topics raises the question of what is a proper examination to assess students’ performance. Furthermore, one has to see that school-informatics is still (and will remain in the foreseeable future) a subject in transition. Hence, teacher’s education was also in the focus of ISSEP 2006. Consequently, the selection of papers contained in these proceedings address the topics just mentioned. Further discussions of these and related topics are covered in “*Information Technologies at Schools*” [3], the remaining part of the proceedings of ISSEP 2006.

The 29 papers contained in this volume were selected out of a total of 204 submissions and invited contributions. The accompanying volume [3] contains 70

scientific papers. Some 50 rather school-practical contributions targeted for the “Lithuanian Teachers Session” are made available on a CD (in Lithuanian) [4]. Each scientific paper was reviewed by at least three members of the Program Committee. The reviewing process and the ensuing discussion were fully electronic.

This volume, although consisting mainly of contributed papers, is nevertheless the result of an arrangement of papers aiming in their final versions to contribute to the specific facet of the program they were accepted for. The remainder of this preface shows how they contribute to the various facets of the conference.

The core of papers contained in this volume center on the *tension between* making pupils familiar with the *fundamental ideas* upon which the discipline of informatics rests, following an aim similar to education in physics or chemistry, *and ICT or computer literacy* instruction. *Dagienė, Dzemyda, and Sapagovas* open this series of papers by reporting the development of informatics education in Lithuania. Due to the political and related social changes in this country, the differences as well as the similarities to developments in other countries are of particular interest. The following papers address the issue of familiarizing students with informatics fundamentals from very different angles. *Kalaš* describes a course where a Logo-platform supports explorative learning. Specific focus is given on (behavioral) modeling, visualizations of fractions, and biological growth. From the different examples, students can identify structure and finally develop algorithmic problem-solving skills. *Hromkovič* describes his approach of relating the beauty of informatics to students attending a course supplementary to general school education. The paper presents the rationale behind kindling pupils’ interest in informatics as a distinct science and explains related didactical aspects. Still at the “high end” of informatics education is the extra-curricular program described by *Yehezkel and Haberman*. Departing from the assumption that in general teachers lack experience and credibility as professional software developers, the authors developed a program where graduates from secondary level schools work on a real project under the mentorship of professional software developers.

In order not to lose focus, the paper by *Szlávi and Zsakó* contrasts two aspects of informatics education: the aim to teach future users of IT-systems and the aim to educate future programmers. The presentation is stratified according to educational aims attainable at particular age levels. In spite of the contrasts highlighted by this paper, *Antonitsch* shows that there are bridges between teaching applications and teaching fundamental concepts. His paper, based on a database application, can be seen as a continuation of bridging approaches reported by Voss (departing from text-processing) and by Antonitsch (departing from spreadsheet-modeling) at ISSEP 2005 [1]. Raising the student’s curiosity by showing informatics’ concepts in such varied disciplines as mathematics, biology, and art is the subject of *Sendova’s* paper. Her approach ensures a low entrance-barrier, but still leads to elementary algorithmic and programming skills.

Clark and Boyle analyze the developments in English schools. Although the British school system differs quite a bit from its continental counterpart, the trends identified by analyzing developments from 1969 onwards find their analogs in most other countries that introduced formal informatics education. Special consideration might be given to their projection into the future. Currently, we still live in a situation where most parents are not computer literate. But this deficiency will gradually vanish

during the years to come. How should school react to a situation when pupils become computer literate following their parents' or their peers' IT-related activities?

The selection of papers on fundamentals is terminated by the work of *Haberman*. It directly leads into both the section on programming and the section on ICT. Specifically, Habermann focuses on the educational milieu and on a gap in perception as to what computing (informatics) is all about. Perhaps resolving this terminological issue, as it has been resolved in distinguishing between learning basic arithmetic (calculating) and mathematics, might solve some public misunderstandings and related problems.

The papers in the initial part of the proceedings focus on the question of “*What to teach?*” To a varying extent they address this question in the context of constrained time to provide the respective qualifications to students. The succeeding set of papers addresses didactical issues of a core aspect of instruction about informatics proper, i.e., *programming and algorithms*. The key question there is: “*How to teach (programming)?*” This part of the proceedings is opened by *Hubwieser*, who explains how object-oriented programming was introduced in the context of a situation where the overall time for informatics education was restricted with respect to initial plans. While Hubwieser's approach for Bavaria foresees a focus on object-oriented software, the paper of *Weigend* addresses three basic issues related to the problem that the capability of performing a task (procedural intuition) is still insufficient for being able to formulate the individual steps necessary to conduct this task (e.g., to write a program). A Python-based system is proposed to overcome this mental barrier. But the problem of finding an appropriate algorithm has many facets. *Ginat* shows the dangers of focusing exclusively on the mainstream strategy of divide-and-conquer for solving algorithmic problems. He points to examples where a global perspective is necessary for obtaining a correct and efficient solution. One might perceive of this paper as a counterpoint to mainstream teaching. It makes teachers and students aware that problem solving needs a rich repertoire of strategies and recipes. There is no once-and-for-all solution.

Kurebayashi, Kamada, and Kanemune report on an experiment involving 14- to 15-year-old pupils in programming simple robots. The authors' approach combines playful elements with serious programming. It is interesting to see that their experiments showed the particular usefulness of this approach for pupils with learning deficiencies.

The master class in software engineering described by *Verhoeff* attaches well to the approaches followed by Hromkovič and by Yehezkel and Haberman. Pupils are invited to this extra-curricular master course which is co-operatively taught at school and at university. The approach of having students complete a small programming project in a professional manner is described in detail. Another concept of a pre-university course to foster algorithmic thinking is described by *Futschek*. He gives three specific examples that can be studied with young people transiting from school to university.

Laucius presents a socio-linguistic issue. While English is the language of computing, one cannot assume too much previous knowledge of this language with pupils if – as for most countries – English is a foreign language. In the case that the local language uses even a different character set, problems are aggravated. Hence, this issue is addressed in several papers by Lithuanian authors. The critical question,

however, might be how far one should go in localizing computer science. The “foreign” language is definitely a hurdle. However, controlled use of the foreign language allows one to clearly separate between object-language and meta-language. To close this circle, the paper by *Salanci* returns to object-oriented programming by presenting an approach for a very smooth, stepwise introduction to working with software-objects.

Papers on *ICT instruction* constitute the ensuing part of the proceedings. They can be seen as a companion to the discussion presented so far. *Micheuz* discusses the selection of topics to be covered in ICT lessons from the perspective of an increasing autonomy within a school system that is at the same time burdened by new constraints (reductions) on the number of courses it may offer. It is interesting to note that an “invisible hand” managed to ensure convergence of the topics finally covered. SeungWook Yoo et al. explain how adoption of model curricula helped to solve problems in informatics education in Korea. *Syslo and Kwiatkowska* conclude this set of papers by noting that the link between mathematics education and informatics education is essentially bi-directional. However, in most current school-books only one of these directions is made explicit. The paper presents some examples where mathematics education could benefit from adopting concepts of informatics.

The widely discussed topics of school informatics addressed so far need context. This context is to be found in the relationships between (maturity) *exams* and informatics instruction, as addressed by *Blonskis and Dagienė*. With the wealth of extra-curricular activities and competitions such as the International Olympiad in Informatics, the question of proper scoring, notably the issue of arriving at a scoring scheme that is not de-motivating to those who are not victorious, becomes of interest. *Kemkes, Vasiga, and Cormack* propose a weighting scheme for automatic test assessments. Their results are generally applicable in situations where many programs are to be graded in a standardized manner and assessments are strictly functionality-based.

Teachers’ education and school development is a different contextual aspect. *Markauskaite, Goodwin, Reid, and Reimann* address the challenges of providing good ICT courses for pre-service teachers. The phenomenon of different pre-knowledge is a well-known didactical problem when familiarizing pupils with ICT concepts. This problem is aggravated in educating future teachers. Some of them will be recent graduates – possibly even with moderate motivation to learn (and use) ICT – while others might look back on a non-educational professional career that may have involved already substantial contact with computing. Special recommendations of how to cope with this problem are given. The focus of *Butler, Strohecker, and Martin* is, in contrast, on teachers that are already experienced in their profession but follow a rather traditional style of teaching. By entering a collaborative project with their pupils, constructivist teaching principles can be brought into practice. Moreover, changes in the teacher’s and students’ roles become noticeable. The ensuing open style of learning is appreciated by all parties of the school system and the approach spreads quite well throughout Ireland.

The proceedings conclude with contributions related to *eLearning*. *Kahn, Noss, Hoyles, and Jones* report on their environment supporting layered learning. This environment allows pupils to construct games where the outcome depends on proper application of physical principles by the student-players. Enriching the model, one

can increase the depth concerning physics instruction. But the layered approach also allows one to manipulate games in such a way that finally (fragments of) programs can be written by the students.

ePortfolios currently attract a lot of attention in didactical circles. *Hartnell-Young's* paper is a worthwhile contribution to this debate, as it presents results from four schools and a special cluster, each with different aims targeted specifically for the student population to be supported. In any case, scope and aspirations were limited but results were encouraging. The paper might well serve as a warning for those who believe a particular ePortfolio can satisfy all those goodies portfolios can support in principle. Remaining at the level of meta-cognition, *Giuseppe Chiazese et al.* present a tool that makes students aware of the activities (partly subconsciously) performed while surfing the Web. Pursuing these ideas further, a transition from computer literacy to Web literacy might be finally achieved at school.

The proceedings conclude with two papers referring to aspects of internationalizing and localizing instructional software. *Targamadzė and Cibulskis* describe the development of the Lithuanian Distance Education Network, a project pursued on the European international level. *Jevsikova* provides a detailed list of issues to be observed when one prepares courseware intended for use on an international level.

A conference like this is not possible without many hands and brains working for it and without the financial support of graceful donors. Hence, I would like to thank particular in the General Chair and the members of the Program Committee, notably those who were keen to review late arrivals as well as those colleagues who provided additional reviews. Special thanks are due to the Organizing Committee led by Roma Žakaitienė and Gintautas Dzemyda. Karin Hodnigg deserves credit for operating the electronic support of the submission and reviewing process, Annette Lippitsch for editorial support for these proceedings.

The conference was made possible due to the support of several sponsors whose help is gratefully acknowledged. Printing and wide distribution of its two volumes of proceedings were made possible due to a substantial contribution by the Government of the Republic of Lithuania, and the Ministry of Education and Science of Lithuania. Finally, hosting of the conference by Seimas, the Lithuanian Parliament, is gratefully acknowledged.

November 2006

Roland Mittermeir

1. Mittermeir R.: From Computer Literacy to Informatics Fundamentals, Proc. ISSEP 2005 (part 1), LNCS 3422, Springer Verlag, Berlin, Heidelberg, 2005.
2. Micheuz P., Antonitsch P., Mittermeir R.: Informatics in Secondary Schools – Evolution and Perspectives: Innovative Concepts for Teaching Informatics, Proc ISSEP 2005 (part 2), Ueberreuter Verlag, Wien, March 2005.
3. Dagienė V., Mittermeir R.: Information Technologies at School; Publ: TEV, Vilnius, October, 2006
4. Dagienė V., Jasutienė E., Rimkus M.: Informacinės technologijos mokykloje. (in Lithuanian), CD, available from the conference webpage <http://ims.mii.lt/imrp>.

Organization

ISSEP 2006 was organized by the Institute of Mathematics and Informatics, Lithuania.

ISSEP 2006 Program Committee

Valentina Dagienė (Chair)	Institute of Mathematics and Informatics, Lithuania
Roland Mittermeir (Co-chair)	Universität Klagenfurt, Austria
Andor Abonyi-Tóth	Eötvös Loránd University, Hungary
Iman Al-Mallah	Arab Academy for Science & Technology and Maritime Transport, Egypt
Juris Borzovs	University of Latvia, Latvia
Laszlo Böszörményi	Universität Klagenfurt, Austria
Roger Boyle	University of Leeds, UK
Norbert Breier	Universität Hamburg, Germany
Giorgio Casadei	University of Bologna, Italy
David Cavallo	Massachusetts Institute of Technology, USA
Mike Chiles	Western Cape Education Department, South Africa
Martyn Clark	University of Leeds, UK
Bernard Cornu	CNED-EIFAD (Open and Distance Learning Institute), France
Zide Du	China Computer Federation, China
Steffen Friedrich	Technische Universität Dresden, Germany
Karl Fuchs	Universität Salzburg, Austria
Patrick Fullick	University of Southampton, UK
Gerald Futschek	Technische Universität Wien, Austria
David Ginat	Tel-Aviv University, Israel
Juraj Hromkovič	Swiss Federal Institute of Technology Zürich, Switzerland
Peter Hubwieser	Technische Universität München, Germany
Feliksas Ivanauskas	Vilnius University, Lithuania
Ivan Kalaš	Comenius University, Slovakia
Susumu Kanemune	Hitotsubashi University, Japan
Ala Kravtsova	Moscow Pedagogical State University, Russia
Nancy Law	The University of Hong Kong, Hong Kong
Lauri Malmi	Helsinki University of Technology, Finland
Krassimir Manev	Sofia University, Bulgaria
Peter Micheuz	Universität Klagenfurt and Gymnasium Völkermarkt, Austria

Paul Nicholson	Deakin University, Australia
Nguyen Xuan My	Hanoi National University, Vietnam
Richard Noss	London Knowledge Lab, Institute of Education, University of London, UK
Sindre Røsvik	Giske Municipality, Norway
Ana Isabel Sacristan	Center for Research and Advanced Studies (Cinvestav), Mexico
Tapio Salakoski	Turku University, Finland
Sigrid Schubert	Universität Siegen, Germany
Aleksej Semionov	Moscow Institute of Open Education, Russia
Carol Sperry	Millersville University, USA
Oleg Spirin	Zhytomyr Ivan Franko University, Ukraine
Maciej M. Sysło	University of Wrocław, Poland
Aleksandras Targamadzė	Kaunas University of Technology, Lithuania
Laimutis Telksnys	Institute of Mathematics and Informatics, Lithuania
Armando Jose Valente	State University of Campinas, Brazil
Tom Verhoeff	Eindhoven University of Technology, Netherlands
Aleksander Vesel	University of Maribor, Slovenia
Anne VILLEMS	University of Tartu, Estonia

Additional Reviewers

Peter Antonitsch	Universität Klagenfurt and HTL Mössingerstr., Klagenfurt, Austria
Mats Daniels	Uppsala University, Sweden
Karin Hodnigg	Universität Klagenfurt, Austria
Kees Huizing	Eindhoven University of Technology, Netherlands
Toshiyuki Kamada	Aichi University of Education, Japan
Shuji Kurebayashi	Shizuoka University, Japan
Ville Leppänen	University of Turku, Finland
Don Piele	University of Wisconsin, USA

Organizing Committee

Roma Žakaitienė (Chair)	Ministry of Education and Science of the Republic of Lithuania
Gintautas Dzemyda (Co-chair)	Institute of Mathematics and Informatics, Lithuania
Vainas Brazdeikis	Centre of Information Technologies of Education, Lithuania

Ramūnas Čepaitis	Chancellery of the Seimas of the Republic of Lithuania
Karin Hodnigg	Universität Klagenfurt, Austria
Annette Lippitsch	Universität Klagenfurt, Austria
Jonas Milerius	Chancellery of the Seimas of the Republic of Lithuania
Algirdas Monkevicius	Seimas of the Republic of Lithuania
Gediminas Pulokas	Institute of Mathematics and Informatics, Lithuania
Alfonsas Ramonas	Chancellery of the Seimas of the Republic of Lithuania
Modestas Rimkus	Institute of Mathematics and Informatics, Lithuania
Danguolė Rutkauskienė	Kaunas Technology University, Lithuania
Elmundas Žalys	Publishing House TEV, Lithuania
Edmundas Žvirblis	Information Society Development Committee under the Government of the Republic of Lithuania

Main Sponsors

ISSEP 2006 and the publication of its proceedings were supported by the Government of the Republic of Lithuania, and Ministry of Education and Science of the Republic of Lithuania.

Table of Contents

The Spectrum of Informatics Education

Evolution of the Cultural-Based Paradigm for Informatics Education in Secondary Schools – Two Decades of Lithuanian Experience	1
<i>Valentina Dagienė, Gintautas Dzemyda, Mifodijus Sapagovas</i>	
Discovering Informatics Fundamentals Through Interactive Interfaces for Learning	13
<i>Ivan Kalas</i>	
Contributing to General Education by Teaching Informatics	25
<i>Juraj Hromkovič</i>	
Bridging the Gap Between School Computing and the “Real World”	38
<i>Cecile Yehezkel, Bruria Haberman</i>	
Programming Versus Application	48
<i>Péter Szlávi, László Zsakó</i>	
Databases as a Tool of General Education	59
<i>Peter K. Antonitsch</i>	
Handling the Diversity of Learners’ Interests by Putting Informatics Content in Various Contexts	71
<i>Evgenia Sendova</i>	
Computer Science in English High Schools: We Lost the S, Now the C Is Going	83
<i>Martyn A.C. Clark, Roger D. Boyle</i>	
Teaching Computing in Secondary Schools in a Dynamic World: Challenges and Directions	94
<i>Bruria Haberman</i>	

Teaching Algorithmics and Programming

Functions, Objects and States: Teaching Informatics in Secondary Schools	104
<i>Peter Hubwieser</i>	

From Intuition to Programme	117
<i>Michael Weigend</i>	
On Novices' Local Views of Algorithmic Characteristics	127
<i>David Ginat</i>	
Learning Computer Programming with Autonomous Robots	138
<i>Shuji Kurebayashi, Toshiyuki Kamada, Susumu Kanemune</i>	
A Master Class Software Engineering for Secondary Education	150
<i>Tom Verhoeff</i>	
Algorithmic Thinking: The Key for Understanding Computer Science	159
<i>Gerald Futschek</i>	
Issues of Selecting a Programming Environment for a Programming Curriculum in General Education	169
<i>Rimgaudas Laucius</i>	
Object-Oriented Programming at Upper Secondary School for Advanced Students	179
<i>Lubomir Salanci</i>	

The Role of ICT-Education

Informatics Education at Austria's Lower Secondary Schools Between Autonomy and Standards	189
<i>Peter Micheuz</i>	
Development of an Integrated Informatics Curriculum for K-12 in Korea	199
<i>SeungWook Yoo, YongChul Yeum, Yong Kim, SeungEun Cha, JongHye Kim, HyeSun Jang, SookKyong Choi, HwanCheol Lee, DaiYoung Kwon, HeeSeop Han, EunMi Shin, JaeShin Song, JongEun Park, WonGyu Lee</i>	
Contribution of Informatics Education to Mathematics Education in Schools	209
<i>Maciej M. Syslo, Anna Beata Kwiatkowska</i>	

Exams and Competitions

Evolution of Informatics Maturity Exams and Challenge for Learning Programming	220
<i>Jonas Blonskis, Valentina Dagienė</i>	

Objective Scoring for Computing Competition Tasks	230
<i>Graeme Kemkes, Troy Vasiga, Gordon Cormack</i>	

Teacher Education and School Development

Modelling and Evaluating ICT Courses for Pre-service Teachers: What Works and How It Works?	242
<i>Lina Markauskaite, Neville Goodwin, David Reid, Peter Reimann</i>	

Sustaining Local Identity, Control and Ownership While Integrating Technology into School Learning	255
<i>Deirdre Butler, Carol Strohecker, Fred Martin</i>	

eLearning

Designing Digital Technologies for Layered Learning	267
<i>Ken Kahn, Richard Noss, Celia Hoyles, Duncan Jones</i>	

ePortfolios in Australian Schools: Supporting Learners' Self-esteem, Multiliteracies and Reflection on Learning	279
<i>Elizabeth Hartnell-Young</i>	

Metacognition in Web-Based Learning Activities	290
<i>Giuseppe Chiazzese, Simona Ottaviano, Gianluca Merlo, Antonella Chifari, Mario Allegra, Luciano Seta, Giovanni Todaro</i>	

Development of Modern e-Learning Services for Lithuanian Distance Education Network LieDM	299
<i>Aleksandras Targamadzė, Gytis Cibulskis</i>	

Localization and Internationalization of Web-Based Learning Environment	310
<i>Tatjana Jevsikova</i>	

Author Index	319
---------------------------	-----

Evolution of the Cultural-Based Paradigm for Informatics Education in Secondary Schools – Two Decades of Lithuanian Experience

Valentina Dagienė, Gintautas Dzemyda, and Mifodijus Sapagovas

Institute of Mathematics and Informatics

Akademijos str. 4, LT-08663 Vilnius, Lithuania

dagiene@ktl.mii.lt, dzemyda@ktl.mii.lt, sapagovas@ktl.mii.lt

Abstract. The history of computers in the secondary schools all over the world goes back to the late seventies and early eighties of the past century. In Lithuania, the official beginning of informatics as a subject in comprehensive schools can be dated back to 1986 when our Ministry of Education and Science have declared the enactment that all schools should start teaching the course named “Basics of Informatics and Computing”.

All things considered, the way in which Lithuanian schools have introduced computers and information technology has been a success story. We have to remember that the present situation has not emerged from nothing but must be seen as a result of a comparatively short time but nevertheless it is extremely full of intensive exploration and work. The leading role was played by a group of researchers from the Institute of Mathematics and Informatics. After intense studies the cultural-based paradigm of introducing computers in schools (information culture, in a broad sense) has been developed, goals and abilities to deal with information considering the economical, cultural, and sociological aspects have been established.

The paper describes the development of Informatics in schools of Lithuania. The main attention is paid to the permanent changes and improvements in designing curricula as well as to didactical approaches.

1 Introductory Remarks

The education achievements that were reached in the eighties and the nineties of the 20th century might be explained by the implementation of computers and information technologies (IT) in schools and by forming of their impact to general education. In Europe and world wide, countries were tackling the problem in different ways: economically more capable countries were buying computers on a mass scale and supplying their schools with educational software. They were also arranging training courses for teachers, in other words: they were using computers wherever they could. The majority of other countries, on the other hand, were trying to develop theoretical well-grounded models of IT education, compose curricula, syllabi, tutorials, textbooks, and arrange trainings, i.e. to provide at least minimal knowledge and skills in informatics and computer implementation to all students with moderate investment in equipment.

In the late seventies and early eighties, the so called learning about computers (or later called learning about information and communication technology, ICT) trend became prominent [28]. The majority of Eastern European countries (at that time academic, theory-based teaching at their schools was dominating), although in a different range, had followed the trend.

Lithuania at that time was part of the Soviet Union. One of the early Soviet pioneers in the field of theoretical and systems programming, a founder of the Siberian School of Computer Science A. Ershov has declared a slogan: Programming is the second literacy [17]. It has become a popular metaphor, which is now widely used without any reference to the author. That weren't empty words – in order to reach the objectives the Soviet Union invested fairly much resources and efforts.

In 1985 the full-scaled teaching of informatics and fundamentals of computing was declared and this meant that such subject was introduced nearly in all comprehensive schools of the Soviet Union.

Lithuania rejected to start teaching the subject in all schools motivating it by the fact that all textbooks were just in Russian and time to translate them to Lithuanian was required. During those years several scholars of the Institute of Mathematics and Informatics started to work intensively in the field of informatics' education: there was need to schedule what and how should be taught in schools in order that students could obtain computer skills and be able to use them in the future. The textbook was translated and – after certain coordination with Soviet Union educational officials – supplemented with educational material more appropriate to our country [2]. Such step was a fairly great achievement, having in mind the conditions of that time, and, on the other hand, it witnessed the achievements and appreciation of scholars of our country.

An original Lithuanian textbook was under intensive preparation. However, it took several years of research to prepare it. The textbook was published just after Lithuania has regained independence [11]. In parallel, in 1991, the first curriculum for teaching informatics in secondary schools was developed [3]. Later it was revised, improved and supplemented several times.

Due to intense research, the direction of informatics in Lithuanian schools quite soon turned into the direction of “learning from a computer” and even sooner, overtaking some other countries, started to actualize the paradigm of “learning with a computer” (later called learning with ICT) [23, 26].

Teaching of information skills integrates goals and abilities to deal with information and employ ICT in a rational way. We even started using the term “information culture” to describe the main goals of ICT integration into comprehensive education of Lithuania. The definition of information culture reflects the current turn of the world towards ICT's use in education, which, in essence, focuses on information skills. The development of information culture (i.e. information skills) was chosen and, until now, is set as the general goal of comprehensive education in the national curricula of informatics and IT [3, 6, 7].

The paradigm not just integrated computer applications during the lessons as it is in Scandinavian countries. The Lithuanian situation, i.e. such aspects as economical, social, cultural, and psychological factors as well as teaching and learning cultural traditions, had to be taken into account. Thus, the cultivation of the cultural-based learning with the computer paradigm was undertaken.

At that time the content of informatics' teaching on a world-wide level consisted of four main concepts: 1) basics of informatics, i.e. the concepts of information and informatics; 2) practical informatics – work with computers; 3) theoretical basis of computing; 4) algorithms and programming. Lithuania has chosen the synthetic and optimal enough relation of these directions and during 20 years has created the original model of informatics' teaching in the comprehensive schools.

In 1986–1997 the duration of the compulsory course in informatics experienced only minor changes – the course took 70 hours overall. The schools that had computers at their disposal could expand the course with additional or elective hours. The course was lectured for two years in school-leaving grades (11 and 12) of secondary schools composing it in different ways. Since 1997 the teaching of informatics essentially changed: the compulsory course for the 9th and 10th grades was introduced (it remained compulsory for the 11th and 12th grades as well) and the possibility to take advanced elective modules was provided. Since 2005 the basics of informatics (IT) as a separate subject has been introduced to the 5th and 6th grades.

2 Role and Influence of Multidisciplinary Sciences

Discussing twenty years of Informatics in Austrian secondary schools, Peter Micheuz mentioned: “even if the visible changes in hardware, software and curricula are remarkable enough it should be pointed out that this short history was a history of people behind these developments, enthusiastic teachers as well as responsible policy makers in that field” [24]. When tracing evolution of informatics in Lithuanian schools, we have noticed a special case: the scope of informatics was formed nearly just by scientists, who sometimes were advised by some advanced teachers.

The Institute of Mathematics and Informatics is one of the state's scientific institutions in Lithuania with the main task to pursue research work in various fields of mathematics and informatics. More than 25 years ago considerations of introducing the science of informatics and in particular of programming to pupils started.

The Institute was established half a century ago as a scientific theoretical institution. Important investigations were performed in the field of differential equations, computational mathematics, and mathematical logic. The studies in probability theory conducted at the institute are well known all over the world. The purely theoretical research of the institute's scholars have not dissociated them from activities in education and particularly from various kind of research in applying information technologies in education. The activities have especially increased during the last two decades when computers and informatics took a very important place in the general education.

The importance of a scientific institute could be assessed according to two features. Firstly, what role do its theoretical results play in the context on a world wide scale? Secondly, how useful it is for its country, its region, and its citizens?

In this section we will review experimental and more practical investigations that are necessary and important for Lithuania in the field of informatics and IT education in schools.

The Institute of Mathematics and Informatics is a research institution that gathers scholars who have an opportunity to choose freely directions and topics and perform

their research. Such openness and confidence in scholarly attitude had a particular value and importance when Lithuania was under Soviet Unions regime.

Just after Russia started forming the direction of informatics' teaching in comprehensive schools and when the slogan about programming as the second literacy was declared, the group of Institute scholars has undertaken the exploratory investigations. The field was already quite cultivated.

Already in 1978–1979 the students' education in programming by using postal services was drafted. After accomplishment of certain experiments, the Young Programmer's School by Correspondence was established in 1981 [9]. The school, which celebrated the 25th anniversary this year, continues to function nowadays; it is led by the Institute scholars who engage students from higher educational institutions to work there as well. The activity of the Young Programmer's School in distance learning was one of the first examples concerning informatics and had a strong impact on many phenomena related with informatics' teaching. For example, in 1992–1993 the project "Distance learning of informatics (programming)" initiated by UNESCO was accomplished [10, 13], the Olympiads in Informatics were introduced. At the beginning just among students of the Young Programmer's School and since 1990 students of the whole country were admitted [16].

The most important experimental research in the field of information technologies at the Institute could be divided into several groups: 1) fostering the cultural heritage and e-democracy (informatics for the humanities), 2) didactics and methodology of informatics, 3) localization of software, 4) information technologies applying in Lithuanian philology [4].

The new ideas declaring that the information technologies are important not only for computer scientists but for the specialists working in the humanities as well emerged at the Institute as early as 1980. In 1994 the UNESCO Chair in Informatics for the Humanities was established at the Institute. This was a rather innovative step: there were only thirteen UNESCO Chairs in ICT in the whole world at that time.

One of the most important issues in the interaction of IT and culture is to supply users with localized computer software. Last year at the Institute we celebrated the ten-year anniversary of the first software localized to Lithuanian [14]. It should be noticed that localization work has been done systematically. The corresponding research has been completed, the term dictionaries have been compiled, the dialogue and the digital vocabulary have been translated, etc.

In such an environment the scope of Informatics education for secondary schools was being formed. Cultural-based paradigm of introducing Informatics in Secondary schools as well as relations with culture and language were not casual, but resulted from the common activity of the Institute's researchers.

During that time a lot of results have been achieved. We can list the most important of them: 1) continual suggestions to policy makers on the teaching of informatics and integration of ICT, 2) accomplished analysis in informatics didactics, 3) development of curricula for informatics and IT in primary, basic, and secondary schools, 4) proposals on the educational standards of informatics and IT, 5) preparation of textbooks and tutorials, 6) delivering courses for teachers in informatics and IT, 7) running the Young Programmer's School and National Olympiads in Informatics as well, 8) localization of well-known educational computer programs (OpenOffice.org, Comenius Logo, Geometer's Sketchpad, etc.), 9) development of dictionaries to

computer terms, 10) publishing of the educational journals: the methodological one called “Informatika” (1986-2001) and the international one “Informatics in Education” (since 2002; http://www.vtex.lt/informatics_in_education/).

3 From Computer Hardware to Fundamentals of Information Handling and Algorithms

At the beginning of introducing informatics in schools, the contents of the subject was rather narrow: it included learning about the components of hardware, as well as programming based mostly on teaching commands.

The course in informatics becomes a compulsory course in all Lithuanian comprehensive schools in autumn 1986. Only a few secondary schools had computer labs then. So, at that time, the theoretical nature of teaching informatics in Lithuania was unavoidable. The textbooks certified by the Education Ministry of Russia had to be used; it provided just one possibility – to translate them to Lithuanian. Those were quite intense years of research of education in informatics for the Lithuanian group of scholars. The worldwide trends and the appropriate tactics had to be evaluated. When translating the Informatics textbook [2] several essential points were seriously doubted and subject for debate.

One of them was the choice of programming languages. The textbook that was recommended for all republics under the Soviet Union embraced three programming languages: an algorithmic language created specially for teaching, Basic and Rapyra. Lithuanian scholars substantiated the necessity of teaching the Pascal programming language and got the permission to supplement the translation with this language. Thus, the Lithuanian version of the textbook was supplemented with original material (that wasn't an easy political task at that time).

However, the translated textbook didn't satisfy the Lithuanian educators of informatics. An original methodology for informatics teaching was under development. In 1991, the first national curriculum of informatics was developed and the first Lithuanian textbook of informatics was published (Fig. 1) [11, 12].

The newly elaborated compulsory curriculum of informatics covered three main areas: 1) information theory: the concept of information, principles of its transmission, storage and processing; 2) logic: logical principles of computers in particular as well as other discrete devices; 3) algorithms: basic control structures of programming and simple data types.

The textbook covered only these three areas. In addition, teachers used other books as well as the materials developed by themselves to further students' practical skills using computers.

In 1995, there were discussions initiated concerning the age of school students that is appropriate to start attending a course of informatics. The discussion had to take into account the country's economic level and the general educational context. Three options were analyzed: grade 6 or 7 (age 12-13), grade 9 or 10 (age 15-16), and grade 11 or 12 (age 17-18).

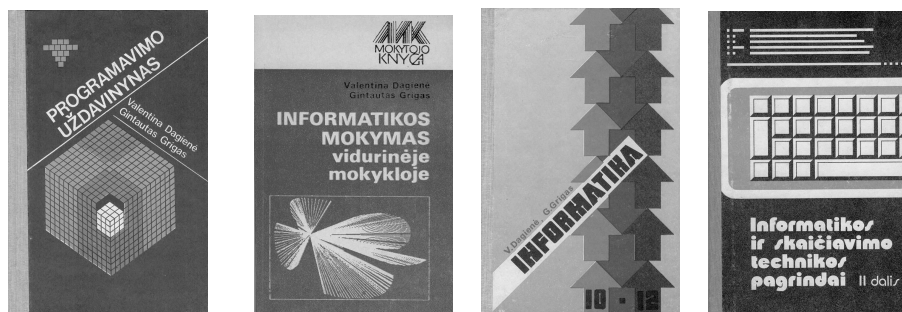


Fig. 1. The first Informatics textbooks used in Lithuania: a) translated from Russian, b) original, c) explanatory book for teachers in teaching Informatics, d) programming training book

It was ascertained that the best time to start teaching informatics is grade 5 or 6. The students should acquire the necessary skills of information technology at the beginning of the basic school and later on could apply their knowledge when learning other subjects. However, the course of informatics for younger students should be based on teaching how to use computers. Due to the lack of hardware at schools at that time, practicing on computers couldn't be introduced as a daily task. The decision was made to start teaching the compulsory course of informatics from the 9th grade. Thus, since the school year 1999/2000 informatics began to be taught as a compulsory course in the final years of the basic school (9th and 10th grades).

4 Goals of Learning and Teaching of Informatics

Jointly with policy makers, Lithuanian scientists have developed the informatics curricula relied on the conception of informatics presented by IFIP under the auspices of UNESCO: "Informatics is a science dealing with the design, realization, evaluation, use, and maintenance of information processing systems; including hardware, software, organizational and human aspects, and the industrial, commercial, governmental and political implication" [22].

Goals, aims, means, and scope of learning Informatics have been discussed in many workshops, conferences, and papers [20, 21]. After long reflections it was concluded to associate developing information skills with traditional and modern culture.

The development of students' information culture was chosen as the main goal of teaching informatics in Lithuanian comprehensive schools. That is a deep-going, comprehensive attitude. In the course of time the conception of information culture may change. Technology may influence learning by changing fundamentally both the content of the subject and the way in which that content can be taught and learned.

This goal was striven for both basic and secondary education. However, later the conception became deeper and more comprehensive. In our opinion, information culture is a broad concept. It is considerably broader than information skills or abilities to work with computer [18, 19].

When preparing the general curricula of informatics, it was attempted to define and concretize the concept of information culture in different ways. The main concepts

included into the information culture were the following: 1) knowledge of the essential systems of informatics and the ability to apply the knowledge in cognition and creation, 2) apprehension of informatics terms and their right use, 3) comprehension of the influence of informatics means on the general culture of mankind, 4) skills of using ICT, 5) ability of logical and creative as well as algorithmical thinking, 6) readiness for constant improvement of one's information style of activities [15].

The goal of information culture is understood as an ideal towards which all the information education in schools should be directed, including compulsory and optional courses meant for informatics. The objective stated is crucial for the basic teaching tasks, related with students' knowledge, skills, practice, abilities, and attitude to values.

The concept of information culture was developed successfully enough. It embraced technological and human factors, and with regard to language and cultural aspects of IT implementation it was constantly going deeper. In 2005 the first habilitation in education of informatics named "Information culture in comprehensive schools: modeling of curricula and learning process" was accomplished [7].

The scope of the compulsory course first of all is expressed by the main goal of growing up in informatics. Then students are encouraged to colligate the obtained knowledge in informatics and the specialization in informatics is being prompted. The first aim is easily understandable in the context of the graduation from comprehensive school: a student has to get a summarized picture of knowledge obtained both at home and at school. The second aim could be related with the early profiling which was intended to help students gifted in informatics to "discover themselves".

5 Obligatory Course of Informatics in Basic Schools

The course of informatics for basic schools had to reach many goals. It had to reveal the main conception of using computers and information technology, to acquaint students with the principal terminology and notions, to show the links of the subject with other school subjects, and to pose the main tasks and problems.

The course of teaching informatics in grades 9–10 is split into four parts: 1) computers (a small introduction to hardware and software), 2) basic principles of handling information, 3) text processing, and 4) introduction to elements of algorithms.

Each part is presented in the curriculum in detail. The topical themes, notions, tasks are indicated. The consistency and depth of their teaching is the matter of a school. It results from the general regulations of the curricula: they are rather of an orientating nature than a directive one.

The first part (computers) is closely connected with the last one (the introduction to algorithms). The part of algorithms seems to be the most problematic one in the course of informatics. The discussion on how they should be taught never ends. In essence, efforts are made to teach students by this course to understand the meaning and significance of a formal notion and to be able to use it in describing the operations performed. The understandings of basic constructors of the formal

language as well as the fundamentals of a computer's functioning are also conveyed. To implement algorithms Pascal or Logo could be chosen [8].

Handling of information is an infinitely broad subject. Students should be familiar with many concepts related to internet, electronic mail, hypertext, and multimedia. They also need to develop skills for information searching, retrieving, handling, and transferring.

Text processing and document preparation is a very important part. A lot of students have known keyboarding and writing a text by means of a computer from their younger age. The greatest attention is paid to national peculiarities (for example, in Lithuania we use different quotation-marks than those used in English) and cultural presentation of information, text, and other data.

At the moment we undergo certain changes – information technology has begun to be taught in grades 5–6. The whole teaching of informatics in comprehensive school is under the reform.

The curricula for 5th – 10th grades of secondary schools include a compulsory course in IT [5]. Students are being prepared to the further life as citizens of the information and knowledge society that are able to use modern technologies. They are prepared to adapt themselves in a changing world and are ready to develop their professional skills constantly.

In order to find out whether the content of informatics taught in secondary school is adequate and whether the topics chosen correspond with expectations of teachers and students a survey was performed. The population chosen for the research consisted of all secondary schools (511) and gymnasia (92) in Lithuania. A random sample of 23 gymnasia and 31 comprehensive schools was selected from all schools chosen for the survey. The result clearly demonstrated that the components of the content were properly selected, they were all appreciated by teachers: more than 50 percents of the pedagogues welcome them (Fig. 2).

It is highly important to introduce the computer not only as a technological device but also as a tool that helps to develop a person's working, creative and daily activity

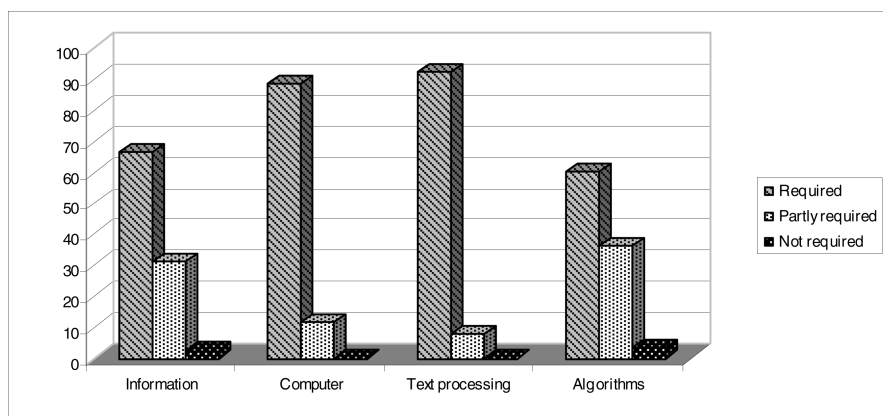


Fig. 2. The necessity of the components of the content (general curriculum) for grades 9-10, according to the data obtained by presentable survey

[25]. In teaching IT, the human being should be the central axis. It is said, the old computing is about what computers can do; the new computing is about what people can do.

In Lithuanian informatics' course, constructive pedagogy is linked to introduce students of younger grades to computing activities. It introduces the main concepts in informatics in a simple indirect way. It is important to propose a practical course, especially problem solving, to develop creativity and to cultivate systematic and consistent thinking. The IT course for secondary schools in fact, was prepared on the basis of constructive methods and cognitive theory.

6 Obligatory and Advanced Courses of Informatics in Secondary Schools

In the years 2000–2001, the whole secondary education has been profiled: two profiles, the humanities and sciences were established. Informatics like the other subjects has been taught on two levels: the core and advanced one. To lecture the general course of informatics in grades 11–12, 70 hours are allotted (in a two-year course). The same amount of hours is allotted additionally to the advanced course.

The core course renders fundamentals of informatics as key subject. It provides minimal proficiency necessary for a student to continue the studies and live in the information society. Its purpose is to give an opportunity for students to get ready to meet the practical needs of living under the conditions of the information society, to acquire the general informational prudence. The core course covers widely used notions of informatics, the main skills of handling information, the usage of elementary means of information technology, and ability of activities.

According to the set of practical proposals elaborated by UNESCO [1], a modular approach was chosen for developing informatics curricula. At the moment three modules are established: introduction to data bases, multimedia, and basics of programming.

The elements of the programming module are quite usual with respect to goals and contents. They have already acquired deep traditions in Lithuanian schools. The scope of this module is problematic in the sense that some theory is required and a lot of time should be spent solving practical problems.

The core curriculum of informatics consists of five main parts: 1) working with more complex texts and basic principles of layout, 2) surfing the Web and communicating, 3) making presentation, 4) working with spreadsheets, and 5) cognitive, social, and ethical issues.

Working with texts and the Web follow-up the teaching of informatics in basic schools. Thus, now the most important part is to develop projects involving work on real-modeling. It is desirable that teaching of presentation should be an integrated part. Working with a spreadsheet as well as teaching social and ethical issues are the essentials of the core course.

In accordance with the data of the survey for the grades 11–12 we also can state that the content of informatics (general curriculum) was properly prepared and teachers welcome all the topics included (Fig. 3). There is just one problematic part of it: the social aspects. This may be explained by several reasons: 1) the social topics

are more theoretically-based, 2) computer tools intended for the topic may be hardly found, 3) teachers of informatics didn't have to chose (or didn't chose) the subject in higher educational institutions.

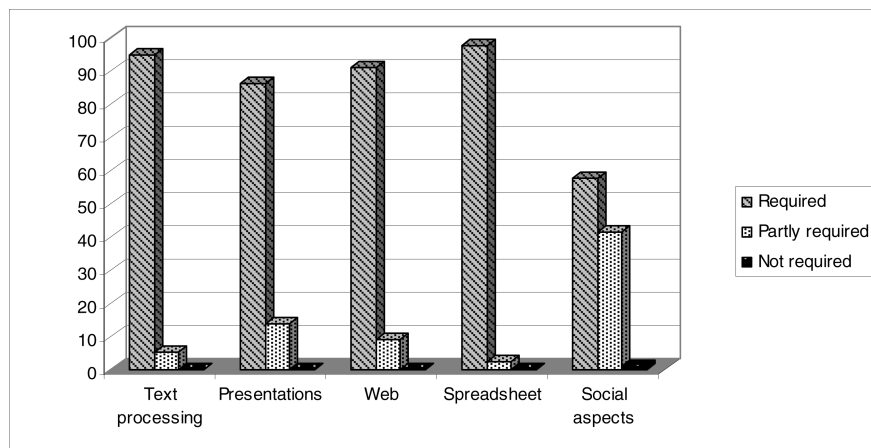


Fig. 3. The necessity of the components of the content (general curriculum) for grades 11-12, according to the data obtained by presentable survey

The advanced course is meant for penetrating deeper into the theoretical aspects of various selected topics of informatics. It is designed to teach one to operate with the knowledge and methods of informatics when solving theoretical and practical problems. It should also train abilities and attitude of studying the subject independently. The main attention is paid to critical thinking, problem perception and solution, the right use of terminology, accurate and grounded selection of technological means and methods and their relation with the general national culture and self-consciousness. Thus, the advanced course ought to serve for an all-round development of professional abilities and proficiency and a better preparation of the studies in a higher school.

Before graduating from the 12th grade, students can choose IT in their school-leaving examination. Starting with this spring, it is possible to choose programming in the State school-leaving examination. The topics of the school examination fully correspond to the curriculum and standards of the compulsory IT course, while the State examination additionally embraces the programming module of the advanced course. Both examinations include test questions on the theoretical part (mainly) and practical tasks which rather reveal practical skills of problem solving.

7 Conclusions and Challenges

In Lithuanian comprehensive schools informatics has become versatile: it includes work by computer, deeper aspects of information handling, and fundamental knowledge indispensable in the information society. That is positive.

Informatics' teaching goals that were formed during the period of two decades, have developed and the informatics curricula, educational standards, and textbooks were constantly revised. Informatics in primary and secondary schools of Lithuania becomes a significant subject, making links between the computer as a tool and culture. The cultural-based paradigm for integrating computers in schools developed by the scientists of the Institute of Mathematics and Informatics was carried on.

A Strategy for introducing ICT in the Lithuanian education system has been developed [27]. According to the strategy, one of the main goals for the period of 2005–2007 is to attain a breakthrough in teaching and student learning by utilizing modern information technologies.

The successful introduction of ICT in schools and learning Informatics is determined not only by money invested in, but also by a change of mental habit and promoting co-operation among researchers in educational technology, policy makers, and teachers. When reflecting on the design and use of technology for education we need to consider the whole teaching and learning situation. The social context is very important, too.

Promoting curricular change, shifting the focus from knowledge as a set of content and especially from technical knowledge to knowledge as integration of process and skills, including culture, language, etc. is a very difficult task. The changing global context due to the impact of ICT is redefining the type of literacy and skills that are needed. Such skills are not only technical but also cognitive, relational and they involve high-order thinking. The importance of new skills has started to receive considerable political interest throughout Europe. These are new challenges for researchers to concentrate their attention to this field.

References

1. Anderson, J, Weert, T. Information and Communication Technology in Education. A Curriculum for Schools and Programme of Teacher Development. Unesco, 2002.
2. Basics of Informatics and Computing (Infolrmatikos ir skaičiavimo technikos pagrindai). Textbook for Secondary Schools. Eds. A. Ershov, V. Monachov, Kaunas: Šviesa, 1987 (in Lithuanian)
3. Curricula for Comprehensive Secondary Schools: Informatics for grades 10–12. (Bendrojo lavinimo mokyklos bendrosios programos) Kaunas: Šviesa, 1991, 10 p. (in Lithuanian)
4. Černiauskas, V., Dagienė, V., Kligienė, N., Sapagovas, M. Some Aspects of Integration of Information Technologies into Education. Informatics in Education, Vol. 1, 2002, pp. 43–60.
5. Dagiene V. Teaching Information Technology in General Education: Challenges and Perspectives. Lecture Notes in Computer Science. R. T. Mittermeir (Ed.). Springer-Verlag, Berlin, Heidelberg, vol. 3422, 2005, 53–64.
6. Dagiene, V. The Model of Teaching Informatics in Lithuanian Comprehensive Schools. Journal of Research on Computing in Education, Vol. 35, No 2 (2002–2003), pp. 176–185.
7. Dagienė, V. Information culture in comprehensive schools: modeling of curricula and learning process. Hab. Doctoral thesis. Kaunas: Vytautas Magnus University, 2005 (in Lithuanian)
8. Dagiene V. Problems of Teaching Logo as a Part of Informatics. EuroLogo '97: Sixth European Logo Conference. Budapest, 20–23 August, 1997, 225–233.

9. Dagienė, V., Dagys, V., Grigas, G. Young Programmer's School – 20 years anniversary (Jaunujų programuotojų mokyklai – 20). Alfa + Omega, 2001, No 1 (11), pp. 93-101 (in Lithuanian)
10. Dagienė, V., Grigas, G. Development of problem solving skills and creativity through distance teaching of programming. Eds. Davies G. and Samways B. Teleteaching. IFIP Transactions (A-29), Elsevier, Sc. Pub., 1993, pp. 179–182.
11. Dagienė, V., Grigas, G. Informatics (Informatika). Textbook for grades 10–11. Kaunas: Šviesa, 1991, 200 p. (in Lithuanian)
12. Dagienė, V., Grigas, G. Teaching Informatics (Informatika). Textbook for teachers. Kaunas: Šviesa, 1992, 182 p. (in Lithuanian)
13. Dagiene, V., Grigas, G. Two modes of distance teaching of informatics. British Journal of Educational Technology. Vol. 24, 1993, No. 2, pp. 144–145
14. Dagienė, V., Grigas, G., Jevsikova, T. Open Source Software Policy in Education. (Atvirųjų programų politika švietime) Vol. 34, Vilnius: VU Pub., 2005, 25–29 . (in Lithuanian)
15. Dagienė, V., Markauskaitė, L. Information Technology in the School Subject of Informatics. International Conference and Exhibition "Information Technologies and Telecommunications in the Baltic States". Riga, April 2-5, 1997, pp. 137-142.
16. Dagiene, V., Skupiene, J. Learning by competitions: Olympiads in Informatics as a tool for training high grade skills in programming. *2nd International Conference Information Technology: Research and Education*. T. Boyle, P. Oriogun, A. Pakstas (Eds.), London, 2004, pp. 79–83.
17. Ershov, A. P. Programming is a second literacy. Pocat e umela intel., 1982, 1, No 6, pp.457-471 (in Russian)
18. General Curriculum and Education Standards: Pre-school, Primary, and Basic Education (Bendrojo lavinimo mokyklos bendrosios programos ir išsilavinimo standartai). Vilnius, Ministry of Education and Science, 2003 (in Lithuanian)
19. General Curriculum for General Education School in Lithuania and General Education Standards for Grades 11-12. Vilnius, Ministry of Education and Science, 2002 (in Lithuanian)
20. Grigas, G. Distance teaching of informatics: motivations, means and alternatives. Journal of Research on Computing in Education. Vol. 27, 1994, No.1, pp. 19–28.
21. Grigas, G. Integration of informatics into school based on the goals of teaching. IFIP Open Conference "Informatics and Changes in Learning". Gmunden, Austria, June 7–11, 1993, pp. 2.
22. Informatics for Secondary Education: A Curriculum for Schools. Paris: UNESCO, 1994.
23. Jonassen, D. H. Computers in the Classroom: Mindtools for Critical Thinking. New Jersey: Prentice Hal Inc., 1996.
24. Micheuz, P. 20 Years of Computers and Informatics in Austria's Secondary Academic Schools. From Computer Literacy to Informatics Fundamentals, Lect. Notes in Computer Science, 3422, Springer, 2005, pp. 20-31.
25. OECD Schooling for Tomorrow. Learning to Change: ICT in Schools. Education and Skills. OECD publications. Paris, OECD Center for Educational Research and Innovation, 2001.
26. Pelgrum, X. J., Schipper, A. T. Indicators of Computer Integration in Education. Computers and Education. Vol. 21, No. 1/2, pp. 141-149.
27. Strategy for the introducing of information and communication technologies in the education system of Lithuania for the period of 2005-2007. 2004. <http://www.emokykla.lt/>
28. Taylor, H. G., Aiken, R. M., van Weert, T. J. Informatics Education in Secondary Schools. Guidelines for Good Practice. IFIP Working Group 3.1. Geneva, IFIP, 1993.

Discovering Informatics Fundamentals Through Interactive Interfaces for Learning

Ivan Kalas

Department of Informatics Education,
Faculty of Mathematics, Physics and Informatics, Comenius University
842 48 Bratislava, Slovakia
kalas@fmph.uniba.sk
<http://edi.fmph.uniba.sk>

Abstract. Theoretical informatics is a hard component of future teachers of informatics academic training. Relatively small space is allocated to it and thus the students' learning curve is very short and very steep. We assume this happens because we expose our students to fundamental problems before they come across them by themselves. We tend to provide students with formal analysis of unfamiliar topics using unfamiliar language.

To improve this we decided to build a new seminar, in which a series of interactive environments developed in Imagine Logo are being used. Using them, students discover fundamentals of informatics. Through such tangible experience they construct their own knowledge and discuss many key challenges with us.

We continue iterative improvement of the contents and form of the seminar. Using methods of the action research we want to understand the problem and help our students to be familiar with the language of theoretical informatics and build positive attitude towards the informatics fundamentals. As an indirect effect we have noticed their growing interest in new forms of teaching/learning methods and growing expectations for the quality of educational software.

1 Introduction

Future teachers (e.g. of informatics) in our educational system have to complete 5 years at a university; to get the complete qualification, they have to graduate from a future teachers' master program in two specializations (subjects). In our case, these are future teachers of informatics and either mathematics or physics, rarely some other natural sciences subject. In their program 20% of all credits are obtained from general courses on pedagogy, psychology, digital literacy etc., 40% from one specialization and other 40% from the other.

Within these 40% of the program, correspondingly small time and space is devoted to learning the fundamentals of informatics. For the students, this is a hard and a highly unpopular domain. When we tried to analyze the reasons for their negative attitude, we came to the following conclusion: Because of the shortage of time we expose them to key issues and problems before they manage to discover those problems by themselves. We present to students formal analysis of the fundamental

problems which aren't so far from *their* fundamental problems. Moreover, we do so using a language which is completely unfamiliar to them.

To improve this situation we have implemented a new one term seminar¹ in the second year of their university study on Discovering Fundamentals of Informatics. During 12 weeks we pass through series of interesting and complex topics, however, we treat them in a completely informal way encouraging *constructionist learning*, see e.g. [10]. To achieve this we are developing new interactive software interfaces for exploratory learning, devoted to one or another key issue of informatics.

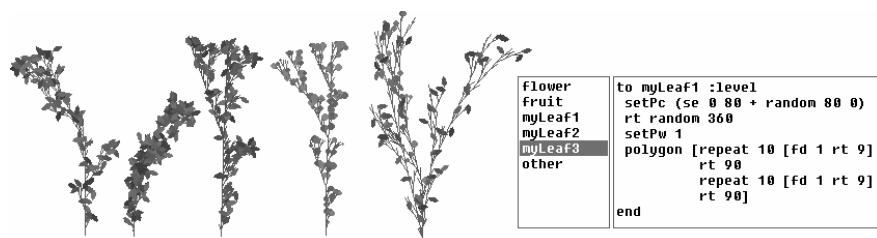


Fig. 1. When developing software applications for our seminar we provide students with *constructivist interfaces for exploratory learning*. To do so we use the Imagine Logo platform (see Sect. 7), which allows us to achieve a high degree of interactivity and openness. This figure shows an application (see more in Sect. 6), in which students construct L-systems (about L-systems see e.g. [9]), have them generate words and interpret them in a Logo turtle way. They define their own procedures as well to be run at the end of each branch.

So far we have run the seminar twice, always trying to observe the reactions of students using interviews, questionnaires and surveys. In the following sections we want to inform about the structure of the seminar, about its form and contents, about teaching/learning methods which we involved, about the Imagine Logo platform which allows us to develop powerful interactive software interfaces, about goals of the seminar, our expectations and results as evaluated by us and by the students.

2 Structure and Form of the Seminar

Intentionally we scheduled the seminar before any course on theoretical foundations. We wanted our students to obtain concrete experience within several key issues like constructing logic circuits, modifying them and exploring corresponding Boolean functions – before they start formal treatment of problems with minimization of DNF expressions etc. We wanted them to *play* with regular expressions before they start proving theorems about them, before they start talking about them using yet another formal language.

This intention has an obvious didactical goal – to build concrete tangible foundations, to increase motivation and preconditions for a more successful encounter with elements of theoretical informatics. Besides this, however, our main goal (and its

¹ Together with D. Pardubska and M. Winczer.

implementation) is supplemented by several secondary goals. We will present them in detail in Sect. 8, let us briefly mention only two of them: (a) we want our students to experience highly alternative arrangements of the teaching/learning process. While it is *possible* to implement this change at the university level, it is *completely inevitable* in primary and secondary education. Thus, future teachers of primary and secondary schools must experience such *constructionist learning* during their pre-service education at the latest. (b) We also want to attract our students' interest in using small educational software applications – *interfaces for learning* or *microworlds* – as far as soon in their program they will study how to design and develop such applications. We want to cultivate their taste, expectations and demands for modern educational software as tools of modern constructionist exploratory learning.

Out of many fundamental topics we decided to treat the following²:

- T1: Informatics and its captivating theory
- T2: Boolean functions and logic circuits (function machines 1)
- T3: Visual fractions (function machines 2)
- T4: Computability and the match box computer
- T5: Ciphers, codes and code breakers
- T6: Propositions and logical riddles
- T7: Regular expressions and their visual representation
- T8: Recursive curves and L-systems
- T9: Language, rules and growth as computation
- T10: Concluding discussion on informatics fundamentals

We spend 1 to 2 weeks pursuing each topic. We thoroughly prepare a worksheet for each session, with initial motivations, short characteristics of the key concepts, problems to be explored (often open, divergent, without single proper answer), assignments to be solved and further topics to be discussed. We also suggest some related topics as proposals for *voluntary reports*. Students work in a PC lab, following the scheme of a worksheet – either individually or in small groups, and always with 2 or 3 lecturers. Some topics are supported by interactive domain-specific software applications, see below. The atmosphere is sociable, exploratory and constructivist.

3 Interactive Interfaces for Learning

When studying fundamentals of informatics, students should encounter formal discourses only after they gain tangible or semi-tangible experience with some key concepts, relations and fundamental problems. Such experience can productively be provided by means of thoroughly designed, domain specific, open and highly interactive software interfaces for learning, usually referred to as **microworlds**, see e.g. [3]. Through these interfaces we want to give students opportunities to build their own knowledge of fundamental informatics concepts. We want them to encounter these concepts through **active engagement** and **exploratory learning**.

² Obviously, beside other reasons we considered also how appropriate those topics would be for supporting them with visual interactive exploratory learning.

Although there are many software applications which run simulations or visualize basic concepts of informatics, like algorithms or data structures, microworlds can be distinguished from other kinds of such learning environments *by their focus on immersive learning and their sensitive tuning to a person's cognitive and motivational states*, see [11]. According to [3], a microworld consists of the following:

- A set of computational objects that model the mathematical or physical properties of the microworld's domain.
- Links to multiple representations of the underlying properties of the model.
- The ability to combine objects or operations in complex ways, similar to the idea of combining words and sentences in a language.
- A set of activities or challenges that are inherent or pre-programmed in the microworld; the student is challenged to solve problems, reach goals etc.

Three examples below illustrate how we have engaged the cited design attributes to implement our approach. We haven't succeeded so far in supporting every topic by its own domain specific microworld, although it is our intention to do so in the future.

4 T2: Boolean Functions and Logic Circuits (Function Machines 1)

This environment is a dynamic building set of basic logic operators, Boolean input and output values and connecting wires. When designing this microworld, we have been inspired by Feurzeig's Function Machines, see [4], which ... *supports learners in a rich variety of mathematical investigations. Its visual representations significantly aid students' understanding of function... and other computational concepts. It is especially valuable for developing mathematical models. To understand a model, students need to see the model's inner behavior working as it runs. At the same time, they need to see the model's external behavior – the outputs generated by its operation.*

In our microworld students drag instances of logic operators from a graphical menu into the working area, connect them by shapeable wires and attach logic values as inputs and outputs. They observe corresponding Boolean functions, they are encouraged to find the "smallest" possible scheme (what is the *size*?), they play with identical schemes (how do we know they are *identical*?), they remove redundant operators (how do we know they are *redundant*?), they look for different schemes of all possible binary functions (*different* in which way?), they construct the smallest scheme which generates certain Boolean function... They model well known rules for logic operations and build more complex circuits. They **give names** to them and use those compound items as extensions of the initial language of logic operators. They build layers of such abstractions and thus build more and more complex circuits like half-adder, full-adder, 4-bit adder or ripple-carry adder. While doing so we discuss several possible views on how to measure logic circuits, on completeness of a set of logic operators, on alternative views of optimization etc.

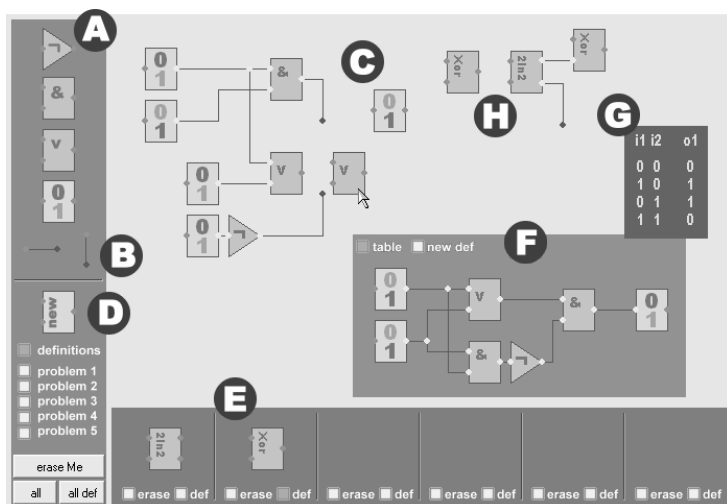


Fig. 2. This is a microworld for constructing logic circuits and studying corresponding Boolean functions. Panel **A** offers basic building “pieces”, panel **B** offers two kinds of connecting wires (extendable and shapeable). In **C** we see a student building his/her new logic circuit. Using the **new** tool from panel **D** one can then select a circuit in the working area, give it a name and store it into panel **E** as an extension of the language of the basic operators. Any definition can be explored or modified in the floating panel **F**. One can also explore corresponding Boolean function in the floating panel **G**. Moreover, newly defined circuit can be repeatedly used as an element in more complex circuits, see **H**.

5 T3: Visual Fractions (Function Machines 2)

The environment of Visual Fractions allows building a kind of Function Machines as well – namely, these are schemes or models to explore and discover the concepts of fractions and fraction relations. It emerged from research done in the CoLabs project (see <http://matchsz.inf.elte.hu/Colabs/>) aimed at the study of on-line communication and collaboration between children.

So this microworld has been developed outside our seminar. However, it addresses teachers of mathematics (and thus also future teachers of mathematics and informatics): they can either use its predefined activities or build their own learning materials (preferably together with their pupils) – as models for exploring fractions and fraction relations. This environment is *pre-didactical*, i.e. it is free of any inherent didactics but allows building any model with multiple representations of fractions and relations.

The microworld provides ten different visual representations of fractions: pie, box, decimal fraction, percentage, ratio, picture, family, number line, fraction and balloon. Fraction objects can be connected together to specify dependencies, which means that some objects represent values of some other objects. The dependencies between objects are dynamic – we can change the value of an independent object with a click and consecutively the value of all dependent objects will change (for more details see [7]).

Beside multiple representations of fractions and relations this microworld also provides **regions** – rectangular objects above the background, each one with its own **key value** and a **condition**. A region is satisfied (which can be visualized by a special animated Yes-No object) if all fractions lying within the region comply with the condition, for example, if they are all smaller than the key value or if their sum equals the key value etc. Thanks to this the power of Visual Fractions language is surprisingly high and building more complex activities is a kind of programming, see [7] and [8].

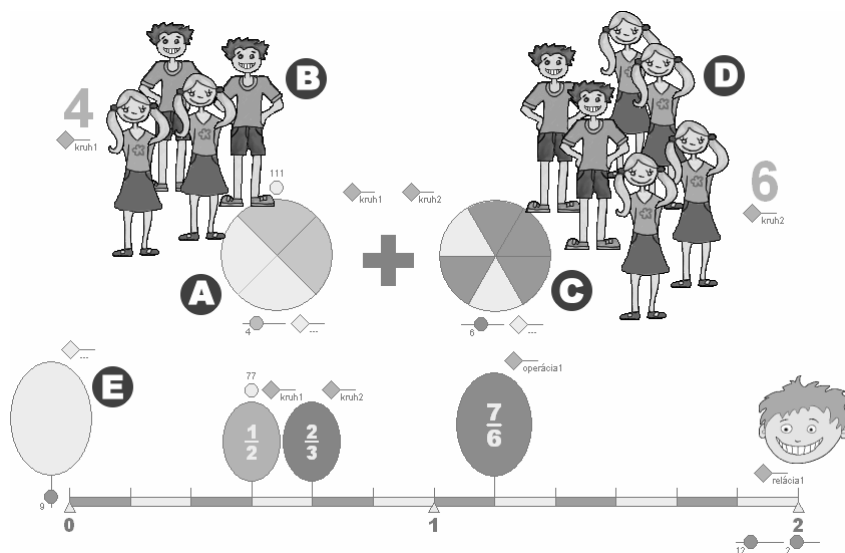


Fig. 3. A fraction with the denominator 4 is represented by a yellow pie **A**. Group **B** consists of four children and the proportion of girls in it depends on the numerator of **A**. The same fraction ($2/4 = 1/2$) is dynamically represented by the position of a yellow balloon on the number line **E**. Fraction **C** has the denominator of 6 and the number of girls in group **D** depends on it. The actual value of this fraction is also represented by the blue balloon on the number line. Third balloon's position (and value) represents the actual sum of fractions **A** and **C**. Our students are encouraged to study and build such models with chains of dependencies, overlapped regions with conjunction of conditions etc., thus discovering the *power of expression of the language of visual fraction*.

6 T9: Language, Rules and Growth as Computation (L-Systems)

In this topic students work in a microworld, in which the growth of natural plants is modeled through computations in a system of an alphabet, initial string and a set of parallel rewriting rules. If the resulting string is visually interpreted in the Logo-like way by the basic turtle commands **forward** and **right**, we may obtain pictures of well-known recursive lines. If we add some extra symbols into the alphabet and interpret them as *cell subdivision* plus if we add various sources of randomness into the process of visualization, we end up with a laboratory for exploring *algorithmic beauty of plants* based on famous and visually attractive L-systems, see Figs. 1 and 4 and [9].

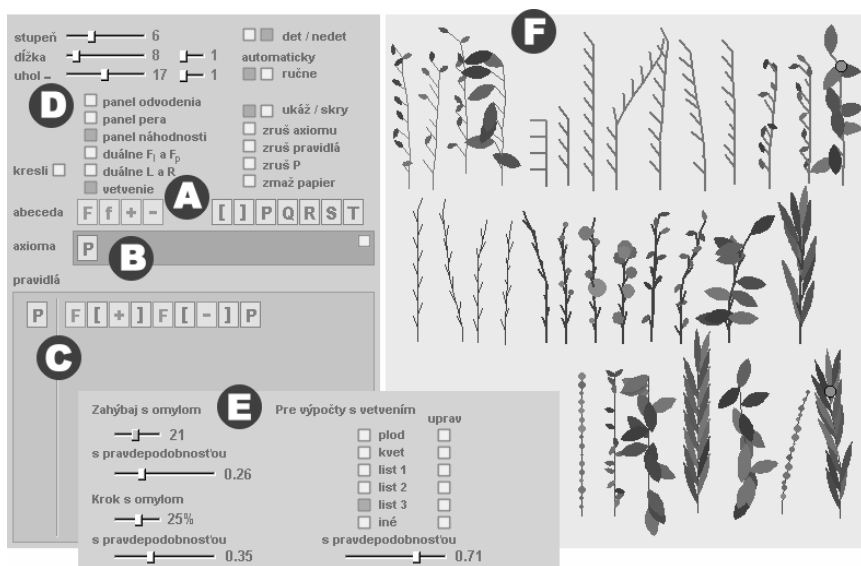


Fig. 4. Students drag symbols of the alphabet from area **A** to construct initial string (axiom) in **B** and one or several rewriting rules in **C**. In **D** they specify attributes for the way in which the resulting string should be visualized. Various sources of randomness within the visualization can be set in the floating panel **E**. Students construct the rules of growth and observe its visual representation run by one or two or... thousands of Imagine Logo turtles (as cells) in **F**.

7 Imagine Logo as a Platform

Recent developments in computer environments for learning have extended traditional solutions by scores of new tools, features and techniques, as illustrated by Imagine, a new generation of the Logo language with a powerful environment for children, students, teachers and professional developers of educational applications, see [5] and [1]. Besides traditional elements of turtle geometry it offers several new attractive features and concepts for education. In [6] we have explored whether and how these new enhancements support understanding in modern exploratory learning. In the research projects like this one we make use of Imagine Logo as a tool for quick and elegant development of interactive interfaces for learning – as illustrated above by three of such microworlds. The most important features, which allow us to do so, are:

Programmable Shapes – One of the most interesting and innovative properties of Imagine, which allows the developer to specify turtle's shapes (or any pictures, still or animated) directly by simple Logo commands. In [5] we presented a broad palette of possibilities offered by this property for educational environments.

Events and parallel processes – Represent a new approach in the development of interactive interfaces for learning. In a rather natural and powerful way they support what we develop – either with children or for children: in many iterative design loops we build actors or objects, we build their shapes, behaviours, interactions with the

environment or between each other. Parallel processes are forces, which can independently and in parallel watch over relations and dependencies, make actors move, make objects follow certain paths or curves, run various experiments, simulations, gadgets, dynamic models etc.

Multiple Turtles and object-oriented structure – An “unlimited” number of actors (that can be organized into “families” of similar objects with similar or identical behaviours) help us to construct scenes with dozens or hundreds of actors of any kind. These actors may be dynamically created, may react to outer circumstances, alter their reactions, become models for new clones, may cease to exist etc.

These are some of the features of Imagine, thanks to which we can build highly open and interactive microworlds and modify them in accordance with our observations and reactions of the students who utilize them in their exploratory learning.

8 Evaluation of the Approach

To reach better understanding of the problem, which our students face when acquainting with the fundamentals of informatics, and to find its improvement, we decided to apply methods of action research – *a powerful tool for change and improvement at the local level*, see [2]. In the iterative cycles (each one taking a school year) we realize fresh runs of our seminar. We systematically try to innovate it based on our previous experience and observations, so that it better meets our goals and students’ needs.

One term (i.e. 14 weeks with 2 hours per week) is devoted to this motivating introductory seminar. When completing it successfully, students earn 2 credits out of the total number of 120 credits for their whole Informatics education.

8.1 Our Expectations

Goals that we want to meet by this seminar can be divided into two categories. Our main and primary goal is **G1**. Beside it, we want to benefit through the contents and form of the seminar to meet our secondary goals **G2 – G7**. We want:

- **(G1)** to help students engage in and **discover fundamental issues**, concepts and problems of informatics.
- **(G2)** to accelerate students’ interest in **exploratory learning**. Based on the constructionism of Papert, see [10], we want our students to learn to discover things by themselves.
- **(G3)** students to experience modern **non-standard arrangement of the teaching/learning processes** through open informal collaborative discussions and explorations. We expect them to adopt this approach in their future teaching.
- **(G4)** to foster students’ interest in working with **innovative study materials** (for example worksheets).
- **(G5)** to develop students’ positive attitude towards modern **software interfaces for learning** – microworlds. We want to cultivate their taste, expectations and demands for modern educational software (still so rare!),

- **(G6)** to accelerate students' interest in **developing similar microworlds**,
- **(G7)** to develop their **algorithmic and problem solving skills** – although unexpected yet intrinsically present key competency in several topics of our seminar. As experienced by Lehotska, when future teachers are building complex activities in the Visual Fractions environment, algorithmic skills are both required and developed, although future teachers are rarely aware of it explicitly, see [8].

8.2 Students' Reactions

As the instruments for data collection we have been using semi-structured and unstructured questionnaires, interviews, observations and iterative experimental design of the worksheets, scenarios and microworlds. Although both groups so far (in 2004 and 2005) were rather small (up to 15 people), we consider all findings productively contributing to each further run of the seminar. We have also profited from our every week observations and informal interviews with students during the seminar in designing new worksheets and iteratively modifying the microworlds.

Before starting the seminar, each student answered a semi-structured initial questionnaire, in which we wanted to find out what was his/her conception of the fundamentals of informatics, which real problems did he/she find very hard in informatics etc. Often the answers were surprisingly superficial, sometimes even naive.

After completing the whole run of our seminar we again asked students to evaluate what they felt they learned, what they acquired about the fundamentals of informatics etc. Unfortunately, such open and unstructured questionnaires proved to produce hardly any valuable evaluation: the answers were mostly their opinions and attitudes to the way the seminar was organized, how they felt during the seminar... often these were short compliments or well-meant suggestions for small improvements.

They didn't answer our questions on what they had learned in the "vocabulary" of our expectations, plans and goals³. Therefore we decided to hand out another closing questionnaire, more structured and less open, in which we asked them three lots of questions: (a) general info on themselves, (b) their own rating of all goals *formulated by us*, and (c) which topics attracted their interest the most. Results from the first lot gave us some small statistics:

- the average age of the seminar attendee was 21 year, 50% being girls,
- 50% passed final upper secondary exam on informatics (this fact made us consider students' answers from the initial questionnaire surprisingly naive),
- 61% of students did some programming at the secondary school,
- up to 33% of them already did some form of teaching,
- 67% classified themselves as being in favour of solving open divergent problems.

In the second lot of questions we presented the complete list of our own goals⁴ (primary and secondary) to all students and asked them to rank the accomplishment of each of them. Next to our 7 goals we added an open option to specify any additional

³ Among other reason, this proves that our students aren't often reflecting on their own learning.

⁴ To briefly recall the goals: G1 – fundamentals of informatics; G2 – exploratory learning; G3 – modern teaching/learning methods; G4 – modern learning materials; G5 – using microworlds; G6 – developing microworlds; G7 – algorithmic skills.

goal which the student considered accomplished. Fig. 5 shows in its left part summative data from both groups of students:

- ranked as the most successful was the goal G2 (exploratory learning),
- goal G1 (fundamentals of informatics) was ranked only fifth (!),
- surprisingly for us, goal G6 (stimulate interest in developing microworlds) was ranked by students – future teachers of informatics – as the last but one,
- goal G7 (algorithmic skills) was also evaluated as poorly accomplished,
- nearly nobody identified any other goal then G1 – G7.

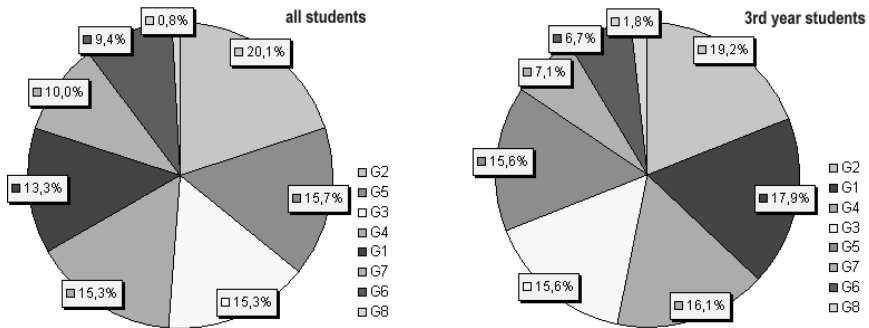


Fig. 5. How students evaluate accomplishing our goals: on the left summative data from all students, on the right only those who evaluated our goals after a lapse of 1 year. More experience gained during that year may have increased rating for G1.

In the right part of Fig. 5 we present the same data, however collected only from older students who completed the seminar a year ago and answered our second closing questionnaire with a lapse of 1 year. While the three lowest positions are identical, goals G1 to G5 ranked very close to each other and G1 ranked second (!). We were pleased to notice that with a higher lapse of time and higher age students seem to be better aware of the primary contribution of our discussions and discoveries. On the other hand, neither they realized clear and close relation between algorithmisation and expressing solutions in a language⁵ with constraints.

In one case – interesting and encouraging for us – a student specified and highly ranked an alternative goal other than ours: as rather high achievement he/she considered the fact that the seminar was often run by three or even four lecturers working in uncommon collaboration (productive for all).

In the third lot of questions we asked our students, which topics attracted their interest the most. Naturally, when interpreting their answers we must be careful – it wouldn't be correct to consider their ranks as the order of success or pertinence of the topics towards our goals. Most probably, topic T5: Ciphers, codes and code breakers ranked first, among other reasons, because it is common and popular since our early years. Nevertheless, in our action research this evaluation gives an indirect hint to us

⁵ For example, specifying rules of an L-system for the given recursive curve or for given look of a modeled plant, tree or grass.

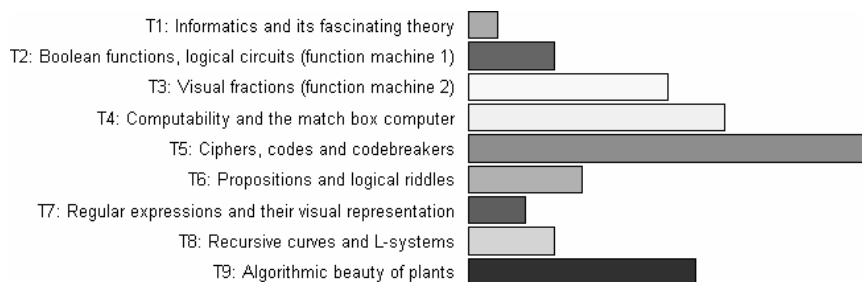


Fig. 6. How did the topics attract interest of the students

which topics must be given more attention, better materials and stronger support by an interactive software microworld.

9 Further Considerations

In the academic programs of future teachers of informatics, ICT play (or should play) double role: they are the subject of our research that we are just learning to understand, and also they are the means of our learning – more efficient and more attractive than anything else. One of the features of ICT (still exploited only marginally) is **visualization**. We mean visualization of concepts, relations, processes, dependencies, behaviours. We decided to benefit from those less exploited opportunities of ICT, connect them with the potential of Imagine Logo in the area of building interactive interfaces for learning and utilize them in favour of our students while they discover the fundamentals of informatics by themselves.

If the domain presented to students is hard, it is crucial to consider two questions:

- Can they understand the problem? Is this also *their problem*?
- Can they understand the language we are using to tackle it?

If in our program we don't have enough time to seriously build the language and make students experience the problem by themselves, we should either skip it or benefit from still undiscovered potential of ICT for developing interactive, visual and immersive applications (interfaces) for learning.

Our seminar on Discovering Fundamentals of Informatics is trying to do so, although we are still far from a satisfactory state. Our experience gained from the ongoing action research proves that this is a productive approach which meets more than one goal. Namely, it accelerates students' interest in **exploratory learning**. We plan to continue the development of new interactive microworlds and also to extend the effect of the results of our research on the form and contents of the seminar.

We can only agree with Resnick when in [10] he says: *Research has shown that many of our best learning experiences come when we are engaged in designing and creating things, especially things that are meaningful either to us or others around us... Like finger paint, blocks, and beads, computers can also be used as a "material" for making things – and not just by children, but by everybody. Indeed, the computer is the most extraordinary construction material ever invented, enabling people to*

create anything from music videos to scientific simulations to robotic creatures. Computers can be seen as a universal construction material, greatly expanding what people can create and what they can learn in the process.

Acknowledgments. This research was partly supported by the European Social Fund in Slovakia through the MISS 21 project  **Európsky sociálny fond**.

References

1. Blaho, A. and Kalas, I.: Object Metaphor Helps Create Simple Logo Projects, Proc. of EuroLogo 2001, A Turtle Odyssey, Linz (2001) 55 – 65
2. Cohen, L., Manion, L., Morrison, K.: Research Methods in Education. RoutledgeFalmer, London (2000)
3. Edwards, L.D.: Microworlds as representations. In diSessa, A.A., Hoyles, C., Noss, R. & Edwards, L.D. (Eds.), *Computers and exploratory learning*. Springer (1995) 127 – 154
4. Feurzeig, W., Richards, J.: Function Machines, CACM Vol. 39, No. 8 (1996) 88 – 90
5. Kalas, I., Blaho, A.: Imagine New Generation of Logo: Programmable Pictures, Proc. of WCC2000, Educational Uses of ICT, Beijing (2000) 427 – 430
6. Kalas, I., Blaho, A.: Exploring visible mathematics with Imagine. In: Marshall, G., Katz, Y. (eds): *Learning in School, Home and Community*. IFIP Kluwer (2003) 53 – 64
7. Lehotska, D., Kalas, I.: Proc of the 7th Int Conference on Technology in Mathematics Teaching, Vol. 1. – Bristol: University of Bristol (2005) 108-116
8. Lehotska, D.: Visual Fractions in Teacher Training. Accepted paper for the IFIP TC3 WG3.1, 3.3 and 3.5 Joint Conference in Alesund (2006)
9. Prusinkiewicz, P. and Lindenmayer, A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, New York (1996)
10. Resnick, M.: Rethinking Learning in the Digital Age. In: Kirkman, G. (ed.): *The Global Information Technology Report: Readiness for the Networked World*, Oxford University Press (2002)
11. Rieber, L.P.: Microworlds. In Jonassen, D. (Ed.): *Handbook of research for educational communications and technology* (2nd ed.). Mahwah, NJ, Lawrence Erlbaum Associates (2004) 583 – 603

Contributing to General Education by Teaching Informatics

Juraj Hromkovič

Information Technology and Education
Swiss Federal Institute of Technology
ETH Zurich, ETH Zentrum CAB F16
Zurich, Switzerland
juraj.hromkovic@inf.ethz.ch

Abstract. What is informatics? What are its main contributions, and which results of informatics are fundamental knowledge about the functioning of our world? Why should computer science be part of the school curriculum and which topics should be introduced? How does teaching informatics contribute to the pupils' way of thinking? In this article, we search for possible answers to these questions.

1 What Is Informatics?

Computer science and informatics are the most common terms used to name the science, whose fundamentals and contributions we will discuss here. Everybody studying or practicing this scientific discipline should every now and then think about how one would define computer science, and contemplate its contributions to science, education and daily life. It is important to realize that learning more and more about a scientific discipline and going deeper and deeper into the understanding of its nature always results in the development of our opinion on the role of this particular science in the context of all sciences. Hence, it is especially important for teachers and students to consistently review their perception of computer science. We have one more reason to think about contributions of informatics to science. We are required to convince society that there are good reasons to teach computer science in schools. And we cannot do this without first understanding its fundamentals in the context of all scientific disciplines. Informatics has a very serious, special problem. Too many people relate computer science to the ability to “drive” a computer: to surf the internet and to use different software like Microsoft Word for text editing. Despite the fact that softwares change from year to year, and hence the ability to use them does not provide any essential value for the general education, several western countries exchanged computer science for Information and Communication Technology (ICT) in schools. In this way we, as computer scientists, are also responsible for this big misunderstanding, and especially those of us who argue that the main reason for teaching computer science is the availability of ICT in almost all families. Many families also own cars, but yet we do not attempt to teaching the

skill of driving a car in schools. The main problem is in viewing computer science through its applications. If that is to be the case, how would physics then be defined? The development and construction of all kinds of technical equipments including computers is based on the physical laws. In this extreme view, almost everybody is a physicist because most are certainly knowledgeable in the operation of products built upon the knowledge of physics, such as TV, cars, mobile phones, etc. Am I going too far? Has the use of computers become an exception because of its complexity?

The answer is clear. We cannot define a science and judge its contributions by its commercial products. We cannot even do so by listing its main research results, because a science is not a sum of its achievements and discoveries. Scientific disciplines spawn new ones at the very fundamental level, where the notions are created and new goals are formulated. To understand our own discipline and its fundamental contributions to science, knowledge and general education, we must go to this level. A more detailed discussion on this topic is presented in [3,4]. We will not hesitate to provoke a conflict between your current opinion of computer science and the viewpoints presented here so as to initiate a discussion that could lead to the development of your understanding of informatics.

Let us first attempt to answer the question

“What is computer science?”

It is difficult to provide an exact and complete definition of a scientific discipline. A commonly accepted definition is:

Computer science is the science of algorithmic processing, representation, storage and transmission of information.

This definition presents information and algorithm as the main objects investigated in computer science. However, it neglects to properly reveal the nature and methodology of computer science. Another question regarding the substance of computer science is

“To which scientific discipline does computer science belong? Is it a meta science such as mathematics and philosophy, a natural science or an engineering discipline?”

An answer to this question serves not only to clarify the objects of the investigation, it also must be determined by the methodology and contributions of computer science. The answer is that computer science cannot be uniquely assigned to any of these disciplines. Computer science includes aspects of mathematics, and natural sciences as well as of engineering. We will briefly explain why.

Similar to philosophy and mathematics, computer science investigates general categories such as

determinism, nondeterminism, randomness, information, truth, untruth, complexity, language, proof, knowledge, communication, approximation, algorithm, simulation, etc.

and contributes to the understanding of these categories. Computer science has shed new light on and brought new meaning to many of these categories.

A natural science, in contrast to philosophy and mathematics, studies concrete natural objects and processes, determines the border between possible and impossible and investigates quantitative rules of natural processes. It models, analyzes, and confirms the credibility of hypothesized models through experiments. These aspects are similarly prevalent in computer science. The objects are information and algorithms (programs, computers) and the investigated processes are the physically existing computations. Let us try to document this by looking at the development of computer science. Historically, the first important research question in computer science was the following one with philosophical roots.

“Are there well-defined problems that cannot be automatically (by a computer, regardless of the computational powers of contemporary computers or futuristic ones) solved?”

Efforts to answer this question led to the founding of computer science as an independent science. The answer to this question is positive. We are now aware of many practical problems that we would like to solve algorithmically, but which are not algorithmically solvable. This conclusion is based on a sound mathematical proof of algorithmic nonsolvability (i.e., on a proof of the nonexistence of algorithms solving the given problem), and not on the fact that no algorithmic solution has been discovered so far.

After developing methods for classifying problems according to their algorithmic solvability, one asks the following scientific question:

“How difficult are concrete algorithmic problems?”

This difficulty is not measured in the difficulty of developing an algorithmic solution, or in the size of the designed program. Rather, this difficulty is measured in the amount of work necessary and sufficient to algorithmically compute the solution for a given problem instance. One learns of the existence of hard problems, for which computing solutions needs energy exceeding that of the entire universe. There are algorithmically solvable problems such that the execution of any program solving them would require more time than has passed since the Big Bang. Hence the mere existence of a program for a particular problem is not an indication that this problem is solvable within practical limits.

Efforts to classify problems into practically solvable (tractable) and practically insolvable led to the most fascinating scientific discoveries of theoretical computer science.

As an example, let us consider randomized algorithms. Most programs (algorithms), as we know them, are deterministic. By deterministic, we mean that the program and the input completely determine all steps of the work on the problem. At every moment, the next action of the program is unambiguously determined and depends only on the current data. Randomized programs may have several options for the next action. Which option is taken is randomly chosen. The work of a randomized algorithm may be viewed as if the algorithm

tosses a coin from time to time to determine its next action, i.e., to choose the next strategy in its search for the correct answer. Hence, a randomized program can have many different computations for an input. In contrast to deterministic programs that reliably deliver the right solution for any input, randomized programs may give erroneous results. The aim is to suppress the probability of such false computations, which under some circumstances means to decrease the proportion of false computations.

At first sight, randomized programs may seem unreliable, as opposed to their deterministic counterparts. Why then the necessity for randomized programs? There are important problems whose solution by the best known deterministic algorithm require more computer work than one can realistically execute. Such problems appear to be practically insolvable. But a miracle can happen: this miracle can be a randomized algorithm that solves the problem within minutes, with a minuscule error probability of one in a trillion. Can one ban such a program as unreliable? A deterministic program that requires a day's computer work is more unreliable than a randomized program running in a few minutes, because the probability that a hardware error occurs during this 24 hours of computation is much higher than the error probability of the fast randomized program. A concrete example of utmost practical significance is primality testing. In the ubiquitous use of cryptographic public-key protocols, huge prime numbers (approximately 500 digits long) must be generated. The first deterministic algorithms for primality testing were based on testing the divisibility of the input n . Alone, the number of primes smaller than \sqrt{n} for such huge values of n exceeds the number of protons in the universe. Hence, such deterministic algorithms are practically useless. Recently, a new deterministic algorithm for primality testing running in time $O(m^{12})$ for n of binary length m was developed. But it needs to execute more than 10^{32} computer instructions in order to test a 500-digit number and so the amount of time since the Big Bang is not sufficient to execute such a computation on the fastest computers. However, there are several randomized algorithms that test primality of such large numbers within minutes or even seconds on a regular PC.

Another spectacular example is a communication protocol for comparison of the contents of two databases, stored in two distant computers. It is mathematically provable that every deterministic communication protocol that tests the equivalence of these contents, needs to exchange as many bits as those within the databases. For a database with 10^{16} bits, this would prove to be tedious. A randomized communication protocol can test this equivalence using a message of merely 2000 bits. The error probability of this test is less than one in the number of all protons in the universe.

How is this possible? It is difficult to explain this without some basic knowledge of computer science. The search for the explanation behind the strengths of randomized algorithms is a fascinating research project, going into the deepest fundamentals of mathematics, philosophy, and natural sciences. Nature is our best teacher, and randomness plays a larger role in nature than one would imagine. Computer scientists can cite many systems where the required characteristics and

behaviors of such systems are achievable only through the concept of randomization. In such examples, every deterministic reliable system is made up of billions of subsystems, and these subsystems must interact correctly. Such a complex system, highly dependent on numerous subcomponents, is not practical. In the case that an error occurs, it would be almost impossible to detect it. Needless to say, the costs of developing such a system is also astronomical. On the other hand, one can develop small randomized systems with the required behavior. Because of their small size, such systems are inexpensive and the work of their components is easily verifiable. And the crucial point is that the probability of a wrong behavior is so minuscule that it is negligible.

The presented concept of randomized algorithms is only one among many concepts developed in informatics that influence our view on fundamentals of science at all.

Despite its above-illustrated scientific aspects, computer science is a typical problem-oriented and practical engineering discipline for many scientists. Computer science not only includes the technical aspects of engineering such as:

organization of development processes (phases, milestones, documentation), formulation of strategic goals and limits, modeling, description, specification, quality assurance, tests, integration into existing systems, reuse, and tool support,

it also encompasses the management aspects such as:

team organization and team leadership, costs estimation, planning, productivity, quality management, estimation of time plans and deadlines, product release, contractual obligations, and marketing.

A computer scientist should also be a true pragmatic practitioner. When constructing complex software (for instance writing programs of several hundred thousand instructions) or hardware systems, one must often make decisions based on one's experience, because one does not have any opportunity to model and analyze the highly complex reality. All the features of engineering are involved in the design and in the development of final products. To mention at least some of them – modularity in the design processes, testing reliability in a structured way, etc.

2 Why Teach Computer Science?

Considering our definition of computer science, one may get the impression that the study of computer science is too difficult for secondary school. One needs mathematical knowledge as well as the understanding of the way of thinking in natural sciences, and on top of that, one needs to be able to work like an engineer. This may really be a strong requirement, but it is also the greatest advantage of this education. The main drawback of current science is in its overspecialization, which leads to an independent development of small subdisciplines. Each branch has developed its own language, often incomprehensible even for researchers in a

related field. It has gone so far that the standard way of arguing in one branch is perceived as superficial and inadmissible in another branch. This slows down the development of interdisciplinary research. Computer science is interdisciplinary at heart. It is focused on the search for solutions for problems in all areas of sciences and in everyday life, wherever the use of computers is imaginable. While doing so, it employs a wide spectrum of methods, ranging from precise formal mathematical methods to experience-based “know-how” of engineering. The opportunity to concurrently learn the different languages of different areas and the different ways of thinking, all in one discipline, is the most precious gift conferred on a computer science student.

Teaching informatics in secondary or even primary schools has to start with programming. Programming is more than just a useful skill of a computer scientist. Learning programming means learning a language of communication with technical systems, learning to tell a machine what activity we would like to have from it. Since machines do not have any intelligence, our instructions must be so clearly and unambiguously formulated that no mistake can arise. In this way, the pupils learn to describe ways and methods for achieving aims that can be correctly followed by everybody without needing to provide the knowledge why they successfully achieve these goals. The development of this skill essentially contributes to the pupils’ natural language skills by motivating pupils to properly think about how to best express what they would like to communicate. After supplementing the programming courses with some elementary data structures and algorithms, we propose to switch to the fundamentals.

Why teach the fundamentals? Theoretical computer science is a fascinating scientific discipline. Through its spectacular results and high interdisciplinarity, it has made great contributions to our view of the world. However, theoretical computer science is not the favorite subject of university students, as statistics would confirm. Many students even view theoretical computer science as a hurdle that they have to overcome in order to graduate. There are several reasons for this widespread opinion. One reason is that amongst all areas of computer science, theoretical computer science is the mathematically most demanding part and hence the lectures on theoretical fundamentals belong to the hardest courses in computer science. Not to forget, many computer science students start their study with a wrong impression of computer science, and many lecturers of theoretical computer science do not present their courses in a sufficiently attractive way. Excessive pressure for precise representation of the minute technical details of mathematical proofs plus a lack of motivation, a lack of relevance, a lack of informal development of ideas within the proper framework and a lack of direct implementation and usage, can ruin the image of any fascinating field of science. Is theory really suitable at the secondary school level? Is it so important that we have to invest huge efforts to master its teaching? We try to answer both these questions affirmatively.

In our previous depiction of computer science as a science with many facets, we have already indirectly brought attention to the importance of the theoretical fundamentals. Because there are several important reasons for the indispensability

of theoretical fundamentals in the study of computer science, we would like to list them in what follows.

1. *Philosophical depth*

Theoretical computer science explores knowledge and develops new concepts and notions that influence science at its very core. Theoretical computer science gives partial or complete answers to philosophical questions such as:

- Are there problems that are not automatically (algorithmically) solvable? If so, where does the boundary lie between automatic solvability and automatic unsolvability?
- Are nondeterministic and randomized processes capable of what deterministic processes are incapable? Is nondeterminism and randomization better (more efficient) than determinism?
- How does one define the difficulty (hardness) of problems?
- Where are the limits of “practical” algorithmic solvability?
- What is a mathematical proof? Is it more difficult to find mathematical proofs algorithmically than to verify the correctness of a given proof algorithmically?
- What is a random object?

It is important to note that many of these questions cannot be properly formulated without the formal concepts of algorithm and computation. Thus, theoretical computer science has enriched the language of science through these new terms, contributing to its development. Many known basic categories of science, such as determinism, chance, and nondeterminism have gained new meanings, and through this, our general view of the world has been influenced.

2. *Applicability and spectacular results*

Theoretical computer science is relevant to practice. On one hand, it provides methodological insights that influence our first strategic decision over the processing of algorithmic problems. On the other hand, it provides particular concepts and methods that can be applied during the whole process of design and implementation. Moreover, without the knowledge and concepts of theoretical computer science many applications would be impossible. Besides the concept of randomized algorithms (as already mentioned), there are many other “miracles” that were born in theoretical computer science.

There are difficult optimization problems for which a simple relaxation of their constraints and requirements decreases the hardness of the problem so much that this decrease corresponds to a gigantic leap from an unrealistic computational demand to that of a few minutes. Often, this relaxation is so small that it is practically negligible.

Do you believe that it is possible to convince somebody of the knowledge of a secret (password), without having to reveal a single bit of this secret?

Do you believe that two persons can determine who of them is older without revealing their ages to the other party? Do you believe that one can almost with certainty check the correctness of mathematical proofs of several thousand pages, without reading it, only by looking at a few randomly chosen bits of them? All these are indeed possible. This not only shows that, thanks to theory, things are made possible though previously they were believed to be impossible, it also shows that research in theoretical computer science is exciting and full of surprises, and so one can be inspired and enthused by theoretical computer science.

3. *Lifespan of knowledge*

Through the rapid development of technology, the world of applied computer science continuously evolves. Half of the existing information about software and hardware products is obsolete after 5 years. Hence, an education that is disproportionately devoted to system information and current technologies, does not provide appropriate job prospects. Whereas the concepts and methodology in theoretical computer science have a longer average lifespan of several decades. Such knowledge will serve its owner well for a long period of time.

4. *Interdisciplinary orientation*

Theoretical computer science is interdisciplinary in its own right and can take part in many exciting frontiers of research and development – genome projects, medical diagnostics, optimization in all areas of economy and technical sciences, automatic speech recognition, and space exploration, just to name a few.

As much as computer science contributes to all other fields, it also benefits from the contributions from other fields. The study of computations on the level of elementary particles, whose behavior follows the rules of quantum mechanics, focuses on the efficient execution of computations in the microworld whose execution in the macroworld has failed. The theoretical model of a quantum computer already exists, but its implementation is a huge challenge to physicists. Currently, nobody has an overview of all consequences of the successful construction of quantum computers. Independent of the success of this project, the rules of the microworld are so surprising and counterintuitive for those who have gained their experience in the macroworld, that one expects many more “miracles” from the use of quantum theory. It is already obvious today that reliable and secure communication exists in the microworld, since every attempt to learn the message submitted will be detected and warded off by the sender. Another exciting area is computing with DNA molecules. DNA molecules are information carriers, and hence, it is not surprising that one can exploit them for information storage and transmission. Today, we are aware that DNA molecules are capable of imitating the work of electronic computers. This is not only theoretically obvious, several simulations of computations by chemical operations over DNA molecules have been performed in laboratories. One cannot exclude

the emergence of DNA molecular computers, where a few DNA molecules can take over the work of an electronic computer.

5. *Way of thinking*

Mathematicians attribute the special role mathematics play in education through development, enrichment and shaping the way of thinking, i.e., through contributing to the general development of one's personality. If this contribution by mathematics is so highly regarded, then one must also acknowledge the importance of computer science for the general education and the enrichment of the way of thinking.

Theoretical computer science encourages creating and analyzing mathematical models of real systems and searching for concepts and methods to solve concrete problems. Remember that precisely understanding which features of a real system are exactly captured by one's model and which characteristics are only approximated or even neglected is the main assumption for a success in science and engineering. Because of this, theoretical computer science calls attention to teaching the evolution of mathematical concepts and models in a strong relation to real problems. Thus, by studying computer science, one has the chance of learning how to combine theoretical knowledge with practical experience and hence develop a way of thinking that is powerful enough to attack complex real-world problems.

We have presented many substantial reasons why teaching at least some theoretical subjects of informatics is important. But quantum mechanics is also crucial for understanding the functioning of our world and yet we do not teach it in secondary schools because of its high degree of difficulty. In the next section we give arguments for teaching theory and refer to our experiments in this direction.

3 What to Teach and How to Teach It?

We already mentioned that computer science courses have to start with programming. We are not advocates of starting with object-oriented programming. We regard this as an additional, optional step. This opinion is related to the main aim we have in these courses and not only to the reasonable idea to follow the historical development of this field. If we want to build a language for instructing (communicating with) a machine that does not have any intelligence, and hence any ability to improvise, we have to be very careful. We have to teach programming as a skill to describe possibly complex behaviors by a sequence of clear, simple instructions. One can even start teaching programming by rewriting recipes for cooking for a cooking machine. One has to start with a very small number of computer instructions available and build new, more powerful instructions by combining the simple instructions available. In this way one not only follows the historical development, one also learns the principle of modularity that is fundamental for all engineering sciences. To be in context with other subjects, the choice of algorithmic problems in the programming course has to include tasks encountered in mathematics, physics, and possibly other subjects.

The programming course can be followed by the introduction of some basic concepts of data structures and fundamental methods for designing efficient algorithms. Again, one has to look especially for mathematical problems and the typical algorithmic problems such as sorting and searching.

Teaching fundamentals could start with automata theory. The reason is that finite automata provide the simplest model of computation, and hence one can learn to understand to some extent, the meaning of the fundamental notions such as computation, simulation, configuration. Moreover, using simple mathematics one can learn modular (structural) design methodology of hardware systems and additionally some important verification concepts that can be performed (for automata) by simple induction.

The last part of our courses of informatics in secondary schools is devoted to computability. Many peers are of the opinion that this is too hard at pre-university level. Everything is a matter of didactic mastery. Our experience is that this topic is considered to be the most fascinating one by the pupils. They visit the core of sciences by learning what infinity is, that there are infinities of different sizes, and finally that there are interesting algorithmic problems that cannot be solved automatically (algorithmically). The pupils are fascinated because we enable them to track the discovery of the fundamentals of informatics and to even walk this path step by step with full understanding of the nature of the discovery processes. For the disbeliever, we recommend the book [3,4], where the whole concept is presented in detail and the materials are available even for people with zero knowledge of computer science.

4 Didactic Concepts

Abiding by a few basic didactic rules is more important than applying many special didactic theories which are rarely experimentally well verified. Through the awareness of electronic means as a useful teaching aid, informatics is in a good standing position to develop teaching patterns useful for other disciplines too. Thus, we should to exploit this channel of teaching. Consequently the elementary didactic rules that we recommend are the following.

4.1 Support of Iterative Teaching

Assure the availability of good textbooks or teaching materials that contain the complete teaching material including all details of your explanations. The textbooks should aim to cultivate repetitive reconsideration of the presented concepts. Every chapter opens with a section “Aims”, in which the motivations, teaching objectives, and relations to the topics of previous chapters are presented. The core of the chapter is dedicated to the formalization of informal ideas by theoretical concepts and to the study within the framework of these concepts. At every essential development, we shall pinpoint its relevance to our aims. Each chapter ends with a short summary and an outlook. Here, the major highlights of the chapter are informally summarized and the relevance to other

parts of theory is once again reviewed. Briefly mention some further developments of the presented theoretical concepts, survey more advanced results and discuss the gap between the achieved knowledge and the research objectives. As usual, the learning process should be supported by exercises. The exercises should not be moved to some special subsections, rather, they should be distributed in the text with our recommendation to deal with them immediately after one has reached them while reading this textbook. They serve to confirm the successful application of the presented concepts and methods as well as to deepen the reader's understanding of the material.

4.2 Less Is Sometimes More or a Context-Sensitive Presentation

Many study guides and textbooks falsely assume that the first and foremost aim is the delivery of a quantum of information to the reader. Hence they often go down the wrong track: maximum knowledge within minimum time, presented in minimal space. This haste usually results in the presentation of a great amount of individual results, and thus neglecting the context to the entire course.

The philosophy behind our concept is different. We would like to build and influence the student's ways of thinking. We are interested in the historical development of computer science concepts and ways of thinking, and the presentation of definitions, results, proofs, and methods is only a means to the end. Hence, we are not overly concerned about the amount of information, preferring to sacrifice 10 to 20% of the teaching material. In return, we dedicate more time to the motivation, aims, connection between practice and theoretical concepts, and especially to the internal context of the presented theory. We place special emphasis on the creation of new terms. The notions and definitions do not appear out of the blue, as seemingly so in some lectures using the formal language of mathematics. The formally defined terms are always an approximation or an abstraction of intuitive ideas. The formalization of these ideas enables us to make accurate statements and conclusions about certain objects and events. They also allow for formal and direct argumentation. We strive to explain our choice of the formalization of terms and models used and to point out the limitations of their usage. Learning to work on the level of terms creation (basic definitions) is very important, because most of the essential progress happens exactly on this level.

4.3 Simplicity and Transparency

We explain simple notions in simple terms. Despite of being firm in our requirements as when teaching mathematics and physics, we avoid the use of unnecessary mathematical abstractions. Hence, we attempt to be as concrete as possible. Through this, we build the introduction on elementary mathematical knowledge. When presenting complicated arguments and proofs, we first explain the ideas in a simple and transparent way before providing the formal proofs.

Clarity takes priority over the presentation of the best known results. When a transparent argument of a weaker result can bring across the idea succinctly, then we will opt for it instead of presenting a strong but technically demanding and confusing argument of the best known result.

Throughout our teaching materials, we work systematically, taking small steps to journey from the simple through to the complicated. We try to avoid any interruptions in thoughts.

5 Some Experience and a Call for Experimental Work

Finally, we would like to test some of our experience in teaching computer science in primary and secondary schools. Let us start by teaching programming in the primary schools, which we have been doing for 8 years now. Our target group is the 3rd and 4th grades. We choose to use the languages SUPERLOGO and IMAGINE ([5,6]) based on the genius concept of the original programming language LOGO ([7]) of Papert. Reducing the syntax to minimum, the pupils are able to start writing programmings immediately at the very beginning of the course. Ten teaching hours are sufficient to bring the pupils to a level where they can:

- write short programs that use instruction repeat, even embedded within a second repeat.
- name procedures and use parameters to build programs in a modular way.

With the exception of the few motivated 3rd and 4th graders who were able to master the use of variables that change their values during program execution, this concept of “modifiable” variables seems to mark the edge of achievable abstraction at this young age. Only from the 6th grade onward is this concept graspable. But at this introductory level, the concept of variables (as equivalents of parameters) is generally comprehensible.

The speedy progress in the development of programming skills by leading LOGO-based languages strongly highlights its suitability as a introductory programming language (even if briefly) before switching to a more commonly used programming language. Another important reason advocating my preference for LOGO-based programming is the essential support it provides for teaching mathematics. This way, pupils are able to master topics, especially geometry, at a much earlier stage.

Teaching algorithmics, complexity or computability may be inappropriate for a teacher without the proper qualifications (e.g. masters) in computer science. As with all other subjects, advanced topics should be taught by experts to ensure success. We tested this out at various secondary schools in accordance to the concept detailed in [3,4]. Our positively encouraging experience stems from our approach to first create notions and concepts before deriving results. This way, we awaken fascination (so often missing in schools) and a thirst for discovery and exploration in search of deeper understanding. This fascination stems from the discovery of fundamental knowledge, and the way this fundamental knowledge interweaves several scientific disciplines. Encouraged by our observations, we are considering to develop sophisticated materials for such topics in the next phase.

To conclude, let me summarize the main problems faced by the computer science didactics community. Following the papers of and submissions to the

conferences, the main subject is to discuss and compare opinions. There are many arguments as to what to teach, and how to teach it. However, many of these arguments are unsupported theories and thus should not be taken seriously, because they are simply opinions. One of our main aims here is to point out the lack of recognition for the values of informatics in general education and the mistake of over-focusing on skills such as “driving” a computer. There is no serious experimental research (unlike in mathematics and physics) that can provide serious arguments for computer science teaching concepts. The situation is as a matter of fact even graver. We have very few reasonable teaching materials and we still have not started to think about the main contributions of teaching computer science, how to measure such contributions, and how to compare different teaching approaches statistically in experimental evaluations. We have not even started to consider standards for evaluating experimental teaching. It comes as no surprise that the acceptance of computer science didactics in the computer science is low. As such, we call for more effort in the following directions:

1. Search for means to measure and compare contributions of the different teaching concepts in the computer science subject. We should define a standard for a convincing measurement.
2. Define standards for executing and evaluating experimental teaching and really commence proper work by performing rigorous experiments.
3. Intensify work on teaching materials, textbooks, e-learning systems and teaching environments. These should by then evaluated through experiments, and such evaluations have to be shared with the whole community.

Now is the time to start enforcing computer science didactics a serious discipline and to match the standards of didactics in mathematics or physics.

References

1. J. Hromkovič: *Theoretical Computer Science*. Springer 2004.
2. J. Hromkovič: *Theoretische Informatik*. Teubner 2004, second edition.
3. J. Hromkovič: *Sieben Wunder der Informatik. Ein Spaziergang an der Grenze des Machbaren*. Teubner 2006. (in German).
4. J. Hromkovič: *Algorithmic adventures. Moving the limits of doable*. Springer 2006.
5. I. Kalaš, A. Blako: Exploring visible mathematics with IMAGINE: Building new mathematics calculus with a powerful computational system, *Learning in School, Home and Community*, 2002, 53-64.
6. I. Kalaš, A. Blako: Young students and future teachers as passengers on the Logo engine, *Secondary School Mathematics in the World of Communication Technology*, 1977, 41-52.
7. S. Papert: *Mindstorms: Children, Computers and powerful ideas. All about LOGO.*, BasicBooks.

Bridging the Gap Between School Computing and the "Real World"

Cecile Yehezkel¹ and Bruria Haberman²

¹ Davison Institute of Science Education, The Weizmann Institute of Science,
Rehovot 76100, Israel
cecile.yehezkel@weizmann.ac.il

² Computer Science Dept., Holon Academic Institute of Technology, and
Davison Institute of Science Education, The Weizmann Institute of Science,
Rehovot 76100, Israel
bruria.haberman@weizmann.ac.il

Abstract. For the last two years the "Computer Science, Academia and Industry" enrichment program has been conducted at the Davidson Institute of Science Education. The extra-curricular program was especially designed for high-school students who major in computer science (CS) or software engineering (SE). The program blends formal and informal learning and provides students with the opportunity to meet with leading representatives of the CS/SE communities of practice. One main goal of the program is to bridge the gap between the school and "real world" of computing that is related to content, learning style, and professional norms. We believe that exposure to the state-of-the-art academic and industrial research and development, to advanced technologies and methodologies, and to professional norms, will establish a different culture of learning, and will provide the students with an entry point into the computing community of practice. Moreover, it is imperative that academia and the high-tech industry will take an active part in educating potential newcomers and will contribute to making the computing professional domain more attractive, especially in the context of the recent high-tech crises. In the paper we describe the extra-curricular program, and discuss implementation aspects.

1 Background

During the last two decades a program in computer science [8] and a program in software engineering [2], especially designed for the high-school level, have been in operation in Israel. The aim of both programs is to expose young students to the fundamentals of computing, and to motivate them to seek expertise in this field. The computer science program introduces concepts and problem-solving methods independently of specific computers and programming languages, along with the practical implementation of those concepts and methods in programming languages. It includes two mandatory modules: Fundamentals of CS and Software Design, and two elective modules: Second Paradigm/Applications and Theory [8]. The aim of the software engineering program is to expose the students to a fundamental scientific

domain whose principles are characteristic of algorithmic thinking as well as system-level perception. The program consists of the following components: (a) an elective topic in natural sciences, (b) computer science (presented above), and (c) an elective advanced specialized topic.

Educators have long noted the importance of teaching software design skills to high-school computer science students [8,9,14]. Specifically, capstone projects have been recognized as an essential part of SE education [3,5,10,14]. The academic CS community believes that the role of projects in the curriculum is of major importance, since it is a means of effective learning, and also is a way of demonstrating the student's mastery of skills appropriate to professional practice [7,10]. Project development enables students to construct knowledge and to enhance cognitive and reflective skills; it also encourages students to become creative and independent learners. In addition, it enables students to encounter real-life experience as a project developer [1,4,9].

Software design skills and problem-solving abilities are gradually developed during the study of the CS and SE programs presented here: (1) the Second Paradigm/Applications module of the CS program requires a small project in which the student has to utilize the specific knowledge he has acquired while studying this module [8]; (2) when studying the Software Design module of the CS program, students solve problems that focus on various aspects of design [9]; and (3) during the last year of the SE program, the students develop a final project - a software system typical of the advanced topic learned, where they apply, in addition to the design methods and implementation tools that are suitable for the specific advanced topic, the integrated knowledge that they have acquired during their three years of studies [2].

The CS and SE programs presented here have evolved over the years in accordance with the changes in the discipline of computing; however, there still exists a gap between the school programs and the "real world" of computing: content, learning style, and the professional norms governing software development processes. Although these programs were not designed to train students to become software professionals, it is recommended that the students should be exposed, as part of their studies, to up-to-date computing research and development (R&D) processes both in academia and in the high-tech industries, and to be aware of actual common professional issues that members of the CS/SE community of practice cope with.

These considerations motivated us to initiate the Computer Science, Academia and Industry educational program in the Davidson Institute of Science Education. Similar approaches, which consider the importance of enrichment programs in motivating students to learn computer science in high school as well as in further academic studies, are presented in [13,14]. In this paper we describe how our program was designed to bridge the gap between school and the "real world" of computing.

2 The Extra-Curricular Program

2.1 Rational and Motivation

The gap between school education and the "real world" of computing especially concerns the following:

(a) Content - The fundamentals and the core technologies that are introduced in school and the state-of-the-art computing research and development as well as the new, evolving directions in the field.

(b) Learning culture - The traditional style of teaching/learning in school is usually designed so that students can acquire explicit knowledge based on a thorough understanding of the topic learned; meaning that besides introducing the main abstract ideas, also concrete details and procedural aspects are emphasized. Recently, there has been a consensus among educators that students should be educated to become self-learners who are capable of navigating in the rapidly growing world of knowledge [6, 11, 12]. Hence, students should be taught to employ a breadth-oriented "tasting"-based learning style, according to which the initial exposure to an unfamiliar topic will be accomplished by getting acquainted only with its essence (i.e. with the main high-level-abstract-related ideas). In other words, a complete understanding, including knowing the concrete details and mastering procedural aspects, should not be considered as the immediate aim of an initial exposure to a new topic.

(c) Professional norms of software development - The Software Engineering 2004 Curriculum states that incorporating real-world elements into the curriculum is necessary to enable effective learning of software engineering skills and concepts [1]. The final school project enables students to experience software design and development processes, and to acquire a system-based perception. However, it has several shortcomings: (a) the students develop individual projects (team-projects are not approved for formal external evaluation by the Ministry of Education); (b) usually, the specifications of the product are not provided by a real external client; the teachers are not members of the SE community of practice, and they usually lack practical industrial experience; the school labs are unable to provide infrastructures characteristic of high-tech industry. Even though the quality of the projects may provide evidence of students' high programming skills and their in-depth investment in the project, the development processes do not resemble actual R&D industrial processes, and the products are rarely applicable to real-world situations.

2.2 Goals

The main goals of our extra-curricular program are as follows: (a) to expose young students to an up-to-date field of computing; (b) to motivate students to seek expertise in the field; (c) to exemplify the synergy between theory and practice; and (d) to establish a learning culture that is based on blending formal and informal learning.

Specifically, the program aims at bridging the gap between the basic principles and the curricular material learned in school, and the "real computing world". We believe that the program may be beneficial in the following aspects: (a) the students and their teachers will have an opportunity to meet leading representatives from academia and the high-tech industry, and thus will be exposed "directly by leading experts" to the state-of-the-art computing research, advanced technologies and methodologies, and professional norms; (b) since the synergy between theory and practice will be manifested, the students will be able to realize that CS and SE involve much more than "just programming" and hopefully will gain a broad sense of the computing field; and (c) the students will experience a different style of learning that hopefully will prepare them to navigate in the rapidly growing world of knowledge and also will enable them to fulfill the extent of their talents.

Moreover, we believe that the interaction of the students with leading representatives of communities of practice from academia and industry, who actually become role models for the students, may motivate them to pursue their studies further or pursue a career in the field.

2.3 The Underlying Model

The model underlying our program is based on Blending Formal (in-school) and Informal (out-of-school) Learning. "It is increasingly apparent that informal and lifelong learning is the key solution to equipping people with the evolving knowledge and skills that will be needed to adapt to the continuously changing nature of society" [11]. Computing is a dynamic, rapidly evolving discipline. Hence, we believe that a blended-learning-based program will provide the students with a suitable entry point into the computing community of practice.

2.4 The Structure of the Program

Our extra-curricular program interweaves enrichment meetings and the development of software projects in a setting that simulates a "real world" environment. The program is planned to be conducted in two stages:

Stage 1 – In the first stage, a regional class, composed of highly motivated advanced students from several schools, accompanied by their teachers, will attend a 6-month preliminary enrichment workshop. Each monthly (after school) meeting consists of a lecture by a CS/SE scientist, a lecture by a SE practitioner, and related class activities. Industry's professional norms are discussed, and advanced technologies and methodologies are demonstrated. In addition, "visiting the industry" tours are conducted. Towards the end of this stage, a small group of students will be selected to continue to the second stage of the program. Scientists from academia and practitioners from industry are matched with students, and the students start to specify their final projects. The program's activities are supported by the site and provide an opportunity to communicate with students between meetings (<http://www.weizmann.ac.il/davidson>).

Stage 2 – The design of the second stage is totally devoted to the development of the final projects under the apprenticeship-based supervision of professional instructors. Some of the students actually participate in "real" industry projects, thus solving "real-world" problems for a real client; others utilize advanced industrial development tools. The school teachers are actively involved in guiding and supporting the students throughout the entire development process.

2.5 Selection of Student Population

2.5.1 Excellence - A Relative Property

The first stage of the program is designed for 11th grade students who major in CS/SE and who their teachers consider as "excellent students", meaning that they are highly motivated and high achievers. Note that we consider excellence as a relative property. We believe that students who demonstrate excellence relatively to their classmates should be appreciated and invited to attend our program; hence, we do not interfere in the students' selection process in the first stage of the program.

Towards the beginning of an academic year, we spoke to high-school principals and CS teachers, and suggested that they select a group of up to 10 excellent students that would like to attend the first stage of our program. As a result, the student population attending the first stage may be very diverse with respect to the students' socioeconomic, cultural, religious, and curricular (i.e. CS and programming knowledge) background, a situation that must be carefully dealt with when planning the agenda of the enrichment workshop and its setting.

The second stage of the program is designed for a small group of graduates of the first stage who are interested in developing "real" software projects under the supervision of experts from academia or the high-tech industry. In our opinion, this stage is suitable only for the "cream of the crop" students who exhibit the following characteristics: high motivation, creativity, self-learning and inquiry ability, persistence, consistency, and the ability to follow up a time table. Accordingly, the selection of students for the second stage is more rigorous than the first stage, and is based on the following criteria: (a) the teachers' recommendation; (b) the applicant's resume, which should provide information about CS knowledge, programming experience, knowledge of programming languages, participation in other relevant enrichment programs, and experience in developing software projects; and (c) the applicant's ability to persuade us that he is seriously interested in developing the project, and that he is capable of successfully accomplishing the development and can submit a working product (according to specification).

2.5.2 Student-Mentor Matching

The student-mentor matching is of great importance to the success of the development of a project. To establish an optimal matching, we decided to conduct an *Employment Fair* meeting in which, in a plenum session, the mentors will present to the students a variety of project subjects that they can advise (see Fig. 1). After the presentation, a



Fig. 1. Employment Fair

face-to-face mentor-student interaction takes place. The process ends when all possible interactions have been conducted. During the interaction the students ask the mentors questions about the suggested projects and examine whether the topics seem attractive and can be coped with and whether they want the mentor to guide them. At the same time, the mentors are implicitly investigating whether the students are qualified enough to develop the project that they suggested. Next, the students are required to submit a list of projects in order of their preference, and the mentors are asked to choose students according to their assessment. Finally, the managers of the program perform the mentor-student pair matching.

3 Implementation

The pilot implementation of the program began at the beginning of 2005. A group of 71 high-school students, accompanied by their teachers from 9 schools located in the area of the Davidson Institute of Science Education, attended the program.

Five monthly enrichment meetings were conducted in which a variety of advanced topics were introduced in plenum sessions by leading representatives of the CS/SE academia and industry:

- *Advanced programming paradigms*- scenario-based programming, aspect-based programming;
- *Development of complex systems*- model-based development, advanced software development tools, computing in space;
- *Artificial intelligence* - machine learning; neural networks, the control of motion in biological and robotic systems;
- *Professional norms* - standards, the importance of testing and of controlled reuse of code;
- *Computer science educational research* - misconceptions and their implication regarding the quality of software.

In addition, the following learning activities were conducted: construction and programming robots (with LEGO Mindstorms), challenging algorithmic problems, role-playing simulation games, creative thinking in computer science, and a competition in testing software. A one-day tour to Intel Kiryat-Gat was conducted, and the students became acquainted with the process of developing chips.

Twenty five students were selected in May 2005 to continue to the second stage of project development under the supervision of 12 mentors (5 talented CS graduate students, 3 lecturers who are active researchers in the department of computer science, and 4 professionals from the high-tech industry). The subjects of the projects actually reflect the mentors' background. Most of the projects mentored by industry representatives have practical characteristics; for example, computerized homes, programming a robot, and managing a multimedia-shop. On the other hand, the projects sponsored by CS faculty and graduate students focus on theoretical or research-based subjects such as computerized graphics, image processing, automatic text categorization, modeling-based development of a control system, simulation of the theory of natural selection, and games based on learning machine theory. The development of the projects is planned to end in April 2006. The projects will be

evaluated by external examiners that will be appointed by the Ministry of Education. *The second cycle of the program* began in November 2005. The current group of attendees consists of 140 students accompanied by their teachers from 20 schools throughout Israel.

4 Preliminary Assessment

Since our aim is to improve the model of our program and its implementation, we decided, towards the beginning of stage 2 of the pilot implementation, that the program should be accompanied with an ongoing evaluation. One main goal of our study was to evaluate the students' and teachers' attitudes towards the "different from school" style of learning. We conducted a post-assessment questionnaire with students who were candidates for stage 2 (Group A). Towards the second cycle of the program we decided to widen the study population and used the same questions at the beginning of the first enrichment meeting to identify the expectations of the newcomers to stage 1 (Group B). We plan to ask all Group B students to fill out a post-assessment questionnaire at the end of the first stage. Later on we plan to evaluate the students' projects and the mentor-student interaction. So far, we also conducted a few open interviews with teachers, groups of students, and instructors. The results of the questionnaires are presented in Table 1.

Content of the program: Both Groups A and B were interested in the program, and they appreciated more the wide exposure to a variety of new subjects than focusing in-depth on one subject. The students from both groups rarely expected the program to focus more on subjects learned at school. Moreover, the A-students identified, on the average, weak links between the topics learned at school and those introduced in the enrichment program; Group-B students had similar expectations. Interestingly, although the program was scheduled at late hours of the day, after learning all day long at school, neither A-students, nor B-students expressed the need to change the ratio of lectures vs. activities in the program (3.2 - Group-A, 3.8 - Group-B).

Contribution and opportunities: The A-students mentioned that the program played a role ($avr=3.4$) in enhancing their understanding of computer science field. We found that newcomers (B-students) to the program are aware of the expected contribution of the out-of-school enrichment program and that the program provides an acquaintance with up-to-date subjects; the A-students mentioned that they indeed had experienced it. The A-students highly appreciated the experience of meeting researchers and professionals in the field; similarly, the B-students expressed high expectations towards this opportunity.

The A-students mentioned that the program increased their motivation to pursue further studies of CS/SE and professional carriers in the field. Similarly, the B-students saw in the enrichment program the potential to increase their interest in the field (for further studies and a professional carrier). The A-students did not experience networking with peers from other schools; in contrast, the B-students apparently recognized the potential of networking with peers. It should be interesting to follow-up B-students' networking, considering the group's diversity (socioeconomic, ethnic, and

Table 1. Students' attitudes towards the program

Statements (Wording of Questionnaire Group A)	Group A N=27 avr (stdev)	Group B N=125 avr (stdev)
I found the enrichment program interesting. (High=5, low=1)	4.0 (1.3)	4.4 (0.8)
It is important for me to attend an out-of-school enrichment program. (High=5, low=1)	4.3 (1.3)	4.5 (0.6)
This program acquainted me with (High=5, low=1):		
▪ Up-to-date R&D in academia	4.0 (1.5)	4.5 (0.6)
▪ R&D in the hi-tech industry	4.2 (1.4)	4.4 (0.8)
▪ Advanced development tools and methodologies	4.2 (1.4)	4.4 (0.8)
It was important for me to meet researchers and professionals in the field.	3.7 (1.3)	4.2 (0.8)
This program increased my interest in (High=5, low=1):		
▪ Pursuing learning CS	3.9 (1.3)	4.3 (0.7)
▪ Pursuing learning SE	3.7 (1.4)	4.1 (0.9)
▪ SE professional carrier	4.0 (1.3)	4.2 (0.8)
I found that the links between the topics learned at school and those introduced in the program were weak. (Weak =5, tight =1)	3.7 (1.4)	3.5 (1.1)
I think that more topics should be linked to the school program. (More=5, less=1)	2.3 (1.3)	2.5 (1.1)
I would have preferred fewer lectures and more activities. (More activities = 5, fewer=1)	3.2 (1.5)	3.8 (1.0)
I would have preferred a program focusing in-depth on one subject. (High=5, low=1)	2.2 (1.3)	2.8 (1.0)
This program provided me opportunity to do networking with peers from other schools.	2.1 (1.4)	3.8 (1.2)
I am interested in participating in a similar enrichment program next year.	4.2 (1.3)	–
I would recommend the program to my friends.	4.0 (1.4)	–
The program played a role in enhancing my understanding of the Computer Science field. (High=5, low=1)	3.4 (1.4)	–

religious). We can conclude that overall, the students viewed the program as fulfilling their expectations. The A-group is highly interested in participating again in a similar program and students feel confident enough to recommend to their friends to participate in the program. The growth in the program's participants may be interpreted as well as a guarantee of its success.

4.1 Triggering a New Culture of Learning

It is too early to assess the long-term and even the short-term effects of the extra-curricular program on the learning styles of students and on the teachers' teachings.

However, there is evidence that the students underwent a change on both a personal and a class level. The teachers reported to us that new types of class activities were initiated by the attendees of the extra-curricular program. For example, attendees conducted sessions in which they presented to the rest of the class topics that they were acquainted with in the enrichment meetings. Sometimes such presentations took place before the enrichment meeting was conducted as a result of pre-meeting enquiries conducted by curious and motivated students. One can say that the attendees actually served as mediators between the class and the lecturers.

5 Concluding Remarks

We presented a novel model of an extra-curricular program based on a combination of plenum-enrichment lectures, group learning activities, and personal project development under the apprenticeship-based supervision of representatives of the computing (academia and industry) communities of practice.

Educators who are concerned about the future of education argue that the models of the learning space should be changed [11], and that curricula should be reorganized to meet the needs of future society and to enhance higher thinking skills of future intelligence [6, 12]. Long and Ehrmann (2005) stated that: "our ability to imagine the classroom of the future is shaped by changes in our beliefs about learning spaces: From focusing on formal education, to emphasizing learning in both formal and nonformal settings..." [11, p. 56]. The program presented here represents an initiative to establish a different "breaking out of the box learning space" [11] in the sense that it enables students to: (a) bridge the gap between fundamentals and up-to-date (tangent to the future) R&D; (b) experience a different learning style; and (c) apply different learning and thinking skills [12].

We hope that his program will trigger more representatives of academia and the high-tech industry to take an active part in educating potential newcomers and that way contribute to making the computing professional domain more attractive.

Acknowledgments

The authors gratefully thank Prof. Moti Ben-Ari, Dr. Yehuda Ben-Hur, and Dr. Miriam Kesner for their support. The authors also gratefully thank the lecturers and mentors for their contribution to the success of the program.

References

1. ACM/IEEE Joint Task Force on Computing Curricula, Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, *A Volume of the Computing Curricula Series*, August, 2004.
2. The Ministry of Education, Israel, A high-school Software Engineering program, 2004. Available: <http://csit.org.il> (in Hebrew).
3. Adams, L., Goold, A., Lynch, K., Daniels, M., Hazzan, O., & Newman, I., Challenges in teaching Capstone Courses, *Proceedings of ITiCSE'03*, Thessaloniki, Greece, 2003, 219-220.

4. Bracken, B., Progressing from student to professional: the importance and challenges of teaching software engineering, *JCSC*, 19(2), 2003, 358-368.
5. Chamillard, A.T., & Braun, K.A. The software engineering capstone: structure and tradeoffs, *Proceedings of SIGCSE'02*, Covington, Kentucky, USA, 2003, 227-231.
6. Computing Research Association, Cyberinfrastructure for Education and Learning for the Future: A vision and research agenda, 2005. Report, available: <http://www.cra.org/reports/cyberinfrastructure.pdf>
7. Fincher, S., Petre, M., & Clark, M., (Eds.). *Computer Science Project Work Principles and Pragmatics*, Springer-Verlag, London, 2001.
8. Gal-Ezer, J., Beerli, C., Harel, D., & Yehudai, A. A high-school program in computer science. *Computer*, 28(10), (1995), 73-80.
9. Gal-Ezer, J., & Zeldes, A. Teaching software designing skills. *Computer Science Education*, 10(1), 2000, 25-38.
10. Holcombe, M., Stratton, A, Fincher, S., & Griffiths, G, (Eds.), Projects in the computing curriculum, *Proceedings of the Project 98 Workshop*, Springer-Verlag, London, 1998.
11. Long, P.D., & Ehrmann, S.C., Future of the learning space: Breaking out of the box, *Educause*, 2005, 42-58.
12. Passig, D. Taxonomy of IT future thinking skills, In Tailor, H., & Hogenbirk, P. (2001). *Information and Communication Technologies in Education: The School of the Future*, Kluwer Academic Publishers, Boston, 152-166.
13. Pollock, L., McCoy, K. Carberry, S., Hundigopal, N., & You, X. Increasing high school girls' self confidence and awareness of CS through a positive summer experience, *Proceedings of SIGCSE'04*, Norfolk, Virginia, USA, 2004, 185-189.
14. Sherrell, L.B., & Shiva, S.G. Will earlier projects plus a disciplined process enforce SE principles throughout the CS curriculum? *Proceedings of SICSE*, St. Louis, Missouri, USA, 2005, 619-620.

Programming Versus Application

Péter Szlávi and László Zsakó

Lorand Eötvös University, Faculty of Informatics, Department of Teacher's Training in
Computer Science

1117 Budapest, Hungary
{szlavi, zsako}@ludens.elte.hu
<http://digo.inf.elte.hu>

Abstract. All over Europe including Hungary there has been serious disputes for years about teaching informatics, about its goals and its possible contents. In this field the sharpest question is the problem of teaching programming and/or application. Do you need one or the other? If both, what is their accurate proportion? Why might you need either of them? Which age group should be taught which of them? This article is aimed at finding the answer to these questions.

1 Introduction

There are several fundamental issues in teaching informatics that have not been satisfactorily solved so far. Informatics as a school subject varies from country to country, region to region. There are countries like France where there is not a compulsory school subject called informatics; in this case students are taught IT skills within the framework of other subjects. In many other countries, however, informatics is defined as an independent subject. This article is not aimed at making a choice between the two possibilities or adducing pros and cons. The authors as well as the Hungarian education government of the past 12 years definitely support an informatics as an independent school subject.

Where there is a school subject called informatics, the next question that arises is: which age groups are to learn it. Although this question is not tackled in this essay, either, we would like to state our position. We believe that in each year of primary and secondary education (i. e. years 1 to 12 in Hungary) students need IT skills therefore informatics should be taught as an independent subject minimum from the third year,¹ which we are trying to prove in section *Informatics at school*. On the other hand, we think that in the last two years of secondary education you cannot define a uniform informatics subject. Surely those who want to go on to university and study computer science must learn something different than those who will start work after leaving school.

¹ Moreover, we agree with Márta Turcsányi-Szabó, who launched the first informatics experiment at Hungarian kindergartens, that even in the kindergarten informatics may be an important tool to develop children's skills and stabilize the community [1].

Starting teaching informatics at an early stage is not a unique phenomenon: e.g. in a decree for primary and secondary schools published by the French National Ministry of Education, it is stated that even pupils of elementary schools i.e. age group 6 to 11 are to be encouraged to use computers, certain software, multimedia, electronic mailing and the internet. [2]. As for the English National Curriculum, Information and Communication Technology (ICT) appears at the very beginning of education i.e. for age group 5 to 7 (Key stage 1)! Although there is not an independent school subject called informatics, ICT is expected to be used every day [3].

Regarding Hungary, the National Institute for Public Education worked out a version of informatics as a school subject called Adventures in Informationland [4] for years 1 to 4. Schools themselves can decide whether they are to include it in their curriculum as a compulsory subject or not. The syllabus is structured as follows:

Year 1	Year 2	Year 3	Year 4
Information games (8)	Getting to know our environment (3+3+2)	Signs and codes (4+4)	Information coding (2+3) Sending a message (3+3+2)
Playing with algorithms (4+3+3)	Everyday algorithms (4+5)	Algorithms in our everyday life (4+5)	Using algorithms (3+3)
Getting to know the computer (15)	Making friends with the computer (8+7)	The computer is our help (14)	Creativity with the computer (10)
Making friends with books (2+2)	Visiting the school library (2+3)	Visits to the library (6)	In the library again (5+3)

Therefore, the starting point of this article is: there is a subject called informatics nearly in every year of primary and secondary education. We will not go into a detailed description of what topics should be covered, either. We will only tackle the problem of programming and application.²

We would also like to sum up why you need teaching programming and/or application (the Delusions of Informatics Education). Then we will move on to survey the goals of programming and application skills. The next part will be dedicated to the contents of algorithmization and application skills (Fields of Informatics). Finally, we will look into what kind of knowledge and skills should students possess at different ages.

2 The Delusions of Informatics Education [5]

We are trying to enlighten the two fields to be taught, their importance and proportion by describing the faulty extreme views and discussing them with a critical eye.

² Besides there are several important fields like infocommunication, media informatics etc.

**Informatics education =
teaching users only**

According to this, informatics is about being able to use computers (and other devices) properly. This delusion holds that the development of abilities, the improvement of problem solving skills, the practice of problem solving activities and the ability to invent are not part of informatics³.

This delusion was created as an opposite to the following delusion (informatics education = teaching programming), and it took nearly 10 years to fight it. It is interesting to note that people who fought against this idea were the ones that had earlier fought against the opposite delusion as well.

The supporters of this delusion often say that it is unnecessary to teach programming because only a few students will become programmers. This is nonsense, and can only be accepted by the blind. Some of the questions that illustrate this are as follows:

- *Why do we teach mathematics: do we want everybody to become a mathematician?*
- *If only a small number of students become a historian, why to teach history?*

Mathematics is taught because it improves thinking and other abilities. The role of informatics is very similar to this⁴, which means that the teaching of programming can only be justified if in the certain age groups there are such skills and abilities, in the development of which it may play an important role.

Our world is full of algorithms: we always do algorithms in our everyday

**Informatics education =
teaching programming only**

This is the opposite of the previous delusion and it claims everything that is denied by the former. Namely, there is no need for a new form of computer literacy, informatics does not change our everyday life, or at least not in a way that should be taught systematically.

They set the following examples: we are not taught to use the telephone or the television at school, and this aspect is true for all the applications of informatics. Even the starting point of this statement is false: most people use their mobile phones (even the remote control of their TV set) in a primitive, very limited way. Informatics invades our everyday life: even our simple devices are becoming more complex and multi-functional, with a number of opportunities. It is a well-known fact (both in pedagogy and in programming) that beyond a certain extent of complexity, frontal recognition, problem solving techniques become difficult, the process of acquisition slows down and it requires such an extent of abstraction skills and notion recognition to understand the logic of the system that can be developed much slower alone than in a well-constructed learning process.

This delusion appeared when computers could only be used for programming, which was typical in the 80's after the introduction of personal computers that could be programmed in BASIC language only. Later, when application systems became widespread, this delusion was pushed into the background and nowadays it is supported by mainly informatics experts working in

³ Even the caveman used his invention skills to rise over animals.

⁴ For mathematics education it is worth considering the following statement: programming can be an experimental device for mathematics! Nevertheless, pedagogy considers experimentation extremely useful in the process of recognition.

life, daily work or while studying. Therefore it is in our own interest to improve our knowledge to understand, execute, even to design algorithms.

secondary education, whose job is to distribute programming tasks.

To be able to avoid the two extreme delusions (teaching only users vs. teaching only programmers) the best course of action is to follow an advice of the famous Hungarian actor Gyula Kabos: to take a little bit of this and a little bit of that as well.

3 Why Do You Need Both?

Solving application tasks	Algorithmization, data modelling
IT tools invade our world. Only those can make good use of the opportunities of the new information society that regularly use these tools. Since they are fairly sophisticated, the stress is not on their routine use but on the knowledge of the opportunities they offer and their creative use. In this field it is important to approach computer use from the problem side, where the question is whether a certain – the given – general program can be applied or not (and we are less interested in the way how it can be used).	In school as well as in your everyday life you keep performing algorithms when filling in data structures – questionnaires, forms – designing action sequences, information-flow processes. This world cannot be fully understood by those who are not aware of the basics of these actions. In your everyday life including school and the various subjects you learn/teach you may have to face several problems that can be – moreover are sensible to be – solved by computers. First students must be able to realise whether a problem or any of its parts can be solved by using IT tools. The next step is solving the problem arisen with the aid of those IT tools.

4 Fields of Informatics [6]

Below we are trying to sum up what we mean by the two important fields of IT knowledge. We are primarily describing the goals and referring to connections with other fields of knowledge. [7]

Solving application tasks	Algorithmization, data modelling
This scope of knowledge emphasizes computer application from the point of view of the problem and the question is whether – the given – general program can be used for problem solving or not (rather than the way how it can be used). We do not concentrate on the tool: the computer or the software. Within the	Algorithmization is an important element because of the development of thinking skills. It is the ability of problem solving; actually not only of solving routine problems but those that require a kind of independence, sound judgement, originality and creativity. That means that a basic objective of teaching

frames of Informatics tools it is the hardware, while in the Application systems and Informatics tools part (that is operating system as program system) it is the software that is focused on. There is a similar idea in the Computer-assisted program solving scope of knowledge as well, but while there the choice is made on the basis of the ability of the 'whole' computer, here the software applied is fixed and only its 'philosophy' is to be studied.

We rely on the knowledge of certain notions and skills that belong to other areas of knowledge. Thus the existing or developing ability of algorithmic thinking is a definite advantage (this is meant to be developed within the scopes of knowledge in Algorithmization and Data modelling).

As it has become evident from the facts mentioned above, the teaching material of this area of knowledge does not exclusively belong to informatics as a subject. There are certain important areas that seem clear today, but their number can be increased with the rapid development of informatics:

- word processing: compilation of text documents, publications in traditional and electronic form;
- constructing graphic images and objects: constructing and processing diagrams, graphic figures, photos;
- spread sheeting: arranging data in a table, making calculations;
- database management: storing, arranging, grouping data, making reports;
- presentation: making presentations, electronic notice boards, billboards;
- multimedia design: designing video and audio files, animation.

Many different subject areas should be considered in order to draw up the problems. The many areas of application at a higher level can be studied within the subject informatics. Handling knowledge

informatics is to emphasize the systematic planning of problem solving.

We learn to understand the world around us with the help of models. *Programming* can be a useful way of developing modelling ability and making students think logically. Because it has to be formalized, it requires a precise, exact way of thinking. The scene of formalization in program composition is data modelling and algorithmization, thus elements connected to them should be taught here. Formalization should be extended with care: using examples, notions that are appropriate to the age group. We regard it very important to lay emphasis on this in education from the very beginning (e.g. it will be regarding the improvement of abstraction skills, or concerning the effectiveness of computer use).

First, algorithmization is not about computer-assisted execution. In most of the cases, the person who created the algorithm can perform it in his mind as well. It is only then that an automatic machine, the computer, can be made to process the precisely constructed algorithm.

The aim of the application of the computer is to create (new) output data from the input data with the help of programs. That is why teaching data structures and algorithms cannot be separated.

The point is that students should realize that the basis of computer-assisted problem solving is algorithm elaboration (and not coding)! However, the knowledge of programming languages is required to reach this, because programming cannot effectively be taught from books only, students need to try their programs on the computer, as well. We would like to note that teaching a *programming language should not be the primary aim* in studying programming.

should be included in a certain subject if the tool is closely linked to a special profession (e.g. CAD).

In this case the stress is on the use of the tool: the program, as a tool is used to solve the task.

It is important for the students to get acquainted with traditional programming structures independently of programming languages. Therefore this scope of knowledge describes computer-assisted problem solving as **tool improvement**, where the problem solving tool (the program) should be created.

5 Informatics at School

What you will find below is based on the Informatics Section of the Hungarian National Curriculum formed in the past 10 years [8], as well as on the requirements of the school-leaving examination. The authors' ideas were included in the last (2005) version of the Hungarian National Curriculum, as well. [9]

Solving application tasks	Algorithmization, data modelling
Years 1 to 4	
<p>The main goal of the first four years is making friends with computers and laying the foundations of a future "harmonic" relationship. Thus what you need here are playful programs and tasks; on the other hand, it is important in other fields of science, as well. Nevertheless, playfulness cannot become dominant and for its own sake; its direct goal i.e. why you are using the computer/program should remain absolutely clear. Playful programs usually do not mean traditional computer games though their introduction into the learning process should not be ruled out in advance, either.</p> <p>For this age group, application skills could primarily mean image and music editing. The very first application field – leading from the kindergarten to school – may be the use of the so-called <i>stamping programs</i> (similar to children's real rubber stamps).</p> <p>Drawing at this level means fusing simple line drawings and patches (colours and textures). If pupils combine drawing, music and some text, they can prepare simple multimedia displays and animations. When using music applications, they can play the music, do</p>	<p>Pupils must be able to formulate algorithms and to execute simple everyday algorithms (morning routine, crossing a street etc.). In order to form these skills, teachers are free to choose their own devices (e.g. Logo-tortoise, robot games, Lego etc.), but they should use a variety of them. Pupils must become aware of the fact that each step of the algorithms must be unambiguously executable and it is not the device that counts.</p> <p>They must realise everyday objects can be described with data, some of which are numbers, others are texts or others (e.g. colour, drawing, music etc.). They must be able to tell the difference between them.</p> <p>Here they should learn the basic concepts of orientation, directions and the measurability of distances. This goal can mainly be achieved by algorithmic games.</p> <p>Data can be sorted: numbers ascending/descending while texts and words alphabetically. In maths classes there are several manual data processing tasks that you can later make use of when</p>

some simple editing or write their own music.

They can prepare invitation cards to a birthday party, carnival posters, classrooms decorations etc. Naturally, teachers' guidance is essential here. It is just as vital that pupils create their own compositions on a traditional medium so that they can take them home and show them to their parents and friends.

It is important to note that good IT applications can highly develop pupils' manual, coordination, calculation, reading and writing skills.

writing algorithms e.g. When teaching colours and geometric forms you can ask a question such as what is more numerous? (counting), Is there a red triangle? (decision making), Select the shapes bordered with straight lines only (selection), Group the shapes by their colours (grouping), Sort them ascending by their size (sorting) etc.

Years 5 to 6

For the 10 to 12 year olds the scope of application possibilities widens.

The most widespread task types here are the ones related to their school and home life like creating, printing, storing and correcting documents in accordance with the interests of this age group.

These are mainly drawings: figures accompanied with a little text such as invitation cards to birthday parties and carnivals, greeting cards, school and class badges, various kinds of posters, playing cards, token money etc. They can prepare the layouts of flats, their classroom, schoolyard etc. on their own. We can state a basic principle: when compared to the previous age group, the difference is a bit more text in the documents created.

The word-processing tasks of the next age group can be introduced by the manual text editing methods: assembling a text from ready-made parts, cutting and pasting as well as swapping parts, making drafts etc.

In these tasks it is sensible to make pupils play the key part. However, it is true again that it is essential for the teacher (language and art teachers) to participate in the process of creation and teach the children the aesthetic and formal

Pupils must be able to formulate precise algorithms and to design simple everyday algorithms (morning routine, the recipe of making tea, crossing a street etc.).

Importantly, they are to create algorithms that they can act out themselves but at the same time they must be executable by computers, as well. The best area for this purpose is moving and drawing (Logo).

They are to formulate what is meant by an algorithm (they can be reduced to steps but the steps themselves are also algorithms; executable with a fixed order of execution; something happens to something at each step)! Relying on their abstraction skills, they should be able to divide data from algorithms that "operate" on them.

They should realise that you use three types of elements when constructing an algorithm:

- each of the atomic steps must be executed (in the given order),
- one of the atomic steps should be chosen and then executed,
- the atomic step should be executed iteratively.

Since the steps of an algorithm could be other algorithms, and they can be named, the concept of procedure can be evolved. [10] According to György

concepts i.e. the teacher's primary task is to add "theory" and to introduce the "methods" and the opportunities they offer.

Pólya: "If one observes it more closely, one may notice that the solutions of many problems actually consist of procedures, processes of actions and sequences of appropriately related operations i.e. a *modus vivendi*." [11]

It is worth introducing this age group the tools and methods of manual data management as they can meet tables and diagrams within the framework of other school subjects. For instance, their first data managing tool is their school report book. They also encounter tables of competitions and might want to sort the rows of the table by scores etc.

The descriptive concept of sorted an unsorted data. Sorting the same data set by various aspects. Systematic manual sorting (e.g. creating alphabetical order by pasting).

Years 7 to 8

As for teaching 13 to 14 year olds the philosophy does not differ greatly from those described above. Maybe just there should be less teachers' guidance (the length and detailedness of the first presentation). The students are already capable of independently using the computer and the programwarehouse of the school to solve the above mentioned tasks. Typically, teachers can set tasks to be solved with the help of a computer as homework.

Tasks related to their school and home life like creating, printing, storing and correcting documents and tables in accordance with the interests of this age group.

Here the stress is laid upon text documents which students should often attach figures, sometimes even tables. A possible task may be preparing a school newsletter, schedules of a summer camp, timetables, business cards, invitations to school competitions, their schedule and results. Before they start creating the documents, it is worth teaching them

Students must be able to write down input and output data necessary for information management and assign them to each other. (At this important stage of abstract thinking, they are able to select details of actions from the purpose of the action and handle them independently; i.e. they can operate with actions as a "black box".) They must be able to analyse the output data from a given point of view and use them for a given purpose.

Making students understand the concept of data: they must tell apart scalar (number, character etc.) and compound data (array, table, text etc.)

For this age they must be able to consciously apply the principle of refinement step by step. They must be able to independently formulate, name, parameter sub-algorithms and use them in constructing algorithms. As the technique of step-by-step refinement is an important problem solving principle not only in the field programming, understanding and learning it may be of great use for every one.

about the aesthetics and typography a text (relief effect, page setting, dividing content units, the role of highlighting etc.).

A table should first appear within a text file; relying on this, students can compute some typical values and illustrate the data in the table with a diagram.

You can also include searching in public computer information systems (task banks, cultural programmes etc.). Students must be able to collect information, paste it into an existing database and then select it from the database and process it.

Relying on text and graphic documents creating an electronic noticeboard or a slide show.

Understanding the tools of algorithmic abstraction (procedures, functions and recursion), realising their usefulness and using them.

Now the algorithmic structures learnt visually by experience at the previous stage should be used consequently and precisely. You may also introduce their common names widespread in computer science: sequence, branch, loop and procedure.

Years 9 to 10

In the curriculum of the 15 to 16 year olds playfulness is diminishing giving way to "reality": in the problems to be solved there are more data and their relations. These tasks require another kind of "creativity": collecting data and exploring their relations i.e. modelling also organically belongs to the problem at a basic level.

Tasks related to their school and home life like creating, printing, storing and correcting documents tables and databases in accordance with the interests of this age group.

Majority of the written materials are texts such as letters, essays etc.

A wider application of tables and diagrams created based on them, computation and plotting of simple statistic data and evaluation of physical and chemical measurements with spreadsheets. A possible task may be planning and calculating the finances of a school trip, the evaluation of school competitions/championships with spreadsheets.

Atomic and compound data (set, array, record, file, stack, row, graph), types of file (sequential and non-sequential), for task types (sum, decision, selection, linear search, count, maximum selection, at least one type of sorting) and realising them on the computer.

Understanding and systematic use of the tools of algorithmic abstraction (procedures and functions).

A basic requirement is that the programs written should be quite expressive i.e. provide enough information as well as a way to enable a dialogue between the user and the computer.

Students must understand that a program is a product and its writer is a product-making craftsman. They must also realise its consequences.

They should become familiar with the basic rules of data modelling and understand that a database is not simply a file but a planned structured system of data and their relations.

An objective of data processing is that the data entered in one way could

Beyond spreadsheets, it is worth mentioning the opportunities of GIS applications such as inserting maps and completing them with data.

Defining and using databases closely related to their everyday life like keeping records of their cassettes or CDs, making their own telephone directory as well as technical and school-subject related databases.

With the aid of text and graphic documents, students might be able to make an interactive electronic news-board or an information board.

be queried from an other point of view. Data query is a creative-analytical process based on exploring data relations. An essential feature of modelling that you are aware of what kind of information will the ready program be able to provide.

Students should understand the concept of database and the related elements (file, record and field). They should know that the logical and physical ways of data representation are different.

Years 11 to 12

As for teaching IT to 17 to 18 year olds, you may mainly come forward with application programs that best suit the profile of the given education institution. This primarily holds for vocational schools and partly for the students of general schools that do not want to continue their studies at higher education institutions after leaving school but want to acquire some more profound, more special IT skills that facilitate their success at the labour market.

Regarding those that go on to university, less stress is laid on this field: e.g. it contains the application of spread-sheets for solving mathematical problems.

Here – and partly at the previous age group – teachers should once again come forward with drawing, image editing and image processing tasks, now at a higher level, relying on the more serious mathematical knowledge students have acquired in the meantime. With the aid of the above, students might be introduced to multimedia design.

With the help of text and graphic documents, they may also try making presentations as a new area of applications.

Atomic and compound data (set, array, record, file, stack, row and graph), file management, relational data structures. Basic algorithms for task types (sum, decision, selection, search, count, maximum selection, at least one type of sorting). Recursion in the world of tasks, data and algorithms. Algorithm designing techniques. [12]

Program writing as a process of production (defining a task, designing, encoding, testing, debugging, efficiency and quality testing and documentation).

6 Conclusion

We do hope that surveying the fields of application based ICT and programming instruction in parallel, we managed to show their importance and the role they play in education. Their proportion is greatly affected by students' skills, interest and the type of school they attend. Based on the above, we believe that in an informatics for everyone the proportion of knowledge about algorithmization should be at least a third and at most half of the time devoted to teaching.

References

1. Turcsányi-Szabó, M.: Approaching Arts through Logo. Sixth European Logo Conference, Budapest, Hungary, pp20-23 August, 1997
2. Nouvelles Technologies. (1999) Mise á niveau informatique en classe de seconde – rentrée 2000. Bulletin Officiel du ministère de l'Education Nationale et du ministère de Recherche. N25 du 24 juin 1999. 1177–1181.
3. The National Curriculum for England (1999): Information and Communication Technology. Qualifications and Curriculum Authority. London. (<http://www.nc.uk.net/download/IKT.doc>)
4. Körösné Mikis, M.: Kalandozások Információországban, (Adventures in Informationland) <http://www.oki.hu/oldal.php?tipus=cikk&kod=oktatas-korosne>
5. Szlávi, P., Zsakó, L.: Delusions in informatics education. Teaching Mathematics and Computer Science, Vol. 2., No. 1., pp151-162, 2004.
6. Szlávi, P., Zsakó, L.: Informatics as a particular field of education. Teaching Mathematics and Computer Science, Vol. 3., No.2, pp283-294, 2005.
7. Zsakó, L.: Teaching Informatics in Hungary. The IOI'96 NewsLetter, No 2, pp5-6, No 3, pp5-6, No 4, pp5-6, 1995.
8. Turcsányi-Szabó, M., Ambrusztter, G.: The past, present, and future of computers in education – the Hungarian image, International Journal of Continuing Engineering Education and Life-Long learning., UNESCO, 2001 Volume 11, Nos 4/5/6.
9. The National Curriculum for Hungary (2005), <http://www.om.hu/main.php?folderID=391>
10. Hvorecky, J., Kelemen, J.: Algoritmizácia, elementárny úvod. ALFA, Bratislava, 1983.
11. Polya, Gy.: Mathematical Discovery on understanding, learning and teaching problem solving. John Wiley & Sons Inc, New York, 1962.
12. Kátai, Z.: “Upperview” algorithm design in teaching computer science in high schools, Teaching Mathematics and Computer Science, Vol. 3., No.2, pp221-240, 2005.

Databases as a Tool of General Education

Peter K. Antonitsch

Universität Klagenfurt
Institut für Informatiksysteme
Peter.Antonitsch@uni-klu.ac.at

Abstract. In informatics education working with databases is considered a matter either of basic ICT training or of informatics proper. This paper points at conceptualization as common ground to both of these seemingly different approaches. It presents a new concept of database instruction, based on the process of structuring. While current didactical concepts centre either on mastering the database program or on database design, the proposed approach is motivated by the primary aim of databases to facilitate information retrieval. It suggests development of structure awareness by starting with analyzing and querying ready-to-use databases provided by the teacher. This offers new aspects of database instruction and emphasizes the ability of databases to contribute to general education as well.

1 Informatics, Learning and the Perception of the World

According to constructivism “[...] education is reflexive creation of reality, which means not only a permanent construction and reconstruction of »world« but of the learner’s identity as well.” ([1], p. 29). Ernst von Glasersfeld (cited in [1], p. 103) points out, that therefore “knowledge cannot be absorbed in a passive way, neither as a result of sensory perception nor by means of communication; knowledge has to be constructed actively by the thinking individual.” The goal of this construction process can be seen as »achieving safety due to understanding the (complex) outside reality« (see [2], pp 105). John D. Nolan refers to Jerome Bruner, who suggested categorization to reduce the cognitive load (due to complexity), and states that “[the learner] can simplify the learning process by organizing the material, that is, putting together those items which share some common attribute, and, then rehearsing together what has been organized together.” [3]. Thereby the learner adds structure to his/her perception of the world.

It is evident that learning and structuring go together, at least in the cognitive domain. But structuring is no innate ability. Findings in neurobiology show that both construction and recognition of external structures correspond to internal cognitive structures that have to be developed during the learning process ([4], pp79). From this point of view the aim of teaching is to support learners in establishing the ability to structure. Focusing on the methods of instruction Sternberg and Martin state: “By helping students understand similarities and differences, the teacher is facilitating the development of an integrated and coherent set of cognitive structures on the part of the students.” [5].

On the other hand each subject matter offers its own specific way of structuring learning content. This holds for language teaching, where grammar formalizes the structure of sentences, this is obviously true for mathematics, which partly deals with applying rules to simplify a given expression but maintaining the structure. The list of examples can be extended to the domain of social studies where it is the society which is viewed under structural premises, from macro-systems like political hierarchies down to structural elements guiding small groups of persons. These different ways of superimposing structure on »reality« can provide learners with patterns they can rely upon when organizing their perceptions, provided that teaching manages to reveal these patterns. This shall be called the »content-driven« approach to build up a frame of reference for structuring.

Informatics has been a distinct subject matter in (Austrian) schools for more than 20 years, adding aspects of technology and »information management« to general education. Moreover, as the science of informatics is of rich structure, the subject matter informatics has the potential to contribute to that content driven viewpoint as well. Some examples may help to illustrate this:

- One of the oldest and still very prominent topics in informatics courses is programming, which is not to be mistaken for coding but stresses modelling and structuring [6]. In the case of procedural programming structuring stands for (structured) decomposition and composition (when programs consist of modules) and the organization of data flow and data storage by making use of control and data structures. Object orientation builds on these concepts but concentrates on the notion of modelling.
- As informatics to some extent depends on technical resources, hardware should be a topic in informatics classes (and it is sometimes). But rather than swamping learners with detailed information about computer components, hardware organization and design could serve as a model of how highly specialized parts have to cooperate to form a functioning whole. This aspect gains importance when contrasted with current developments in society (see [2] for a closer look on school and postmodern society).
- The World Wide Web as a major source for information retrieval relies on hypertext and hypermedia. This calls for different structuring of information on the part of the provider as well as on the part of the consumer. Though helping learners to deal with these new structures is the duty of all subject matters in school, it is the special task of informatics (didactics) to have a closer look at the organizing principles that lay behind (see [7], [8] for a didactical approach to the Web).

It should be easy to find further examples – just think of the structures that evolve when mapping algorithms onto abstract machine-models or of the well-structured guidelines in the process of software-engineering – but it should be clear from the above examples that informatics is structurally rich. Seemingly it is up to the art of teaching to utilize this richness to promote structural thinking.

2 Structuring and Databases

Database programs are missing among the examples for the content driven approach sketched above. Ignoring them would be a twofold slip, as working with (relational)

databases has been a prominent topic in informatics courses at school from the early days of informatics instruction up till now [9] and it relies on programs and on concepts both of which are of rich structure.

Database programs are commonly classified as application software. In the context of PC-software database programs are quite often made available in a bundle combined with some text processing, spreadsheet calculation and/or presentation software. They are designed for professional use and therefore offer overwhelming functionality to the learner. Thus novices to database programs first of all have to see their way to handle the software tool by means which make sense of the program features and the underlying data representation at the same time. While this holds true for any application software, the situation with database programs is still something special: Supposedly, everybody who is about to work with text processing programs is familiar with textual data representation. Similarly doing calculations with a spreadsheet program very likely is preceded by hand-made calculations making use of tables. Therefore learners can rely on prior knowledge when starting to use a software tool of this kind. On the other hand the relational data model is hardly used (and known!) by anyone before working with database programs. Thus at a user level structuring means the conceptualization of the relational (tabular) data representation and of the basic features of the program at the same time. Moreover, conceptualizing the database program usually means to consider features of a whole software suite providing possibilities for data exchange. This level of dealing with databases should be called the »application plane« (see Fig. 1).

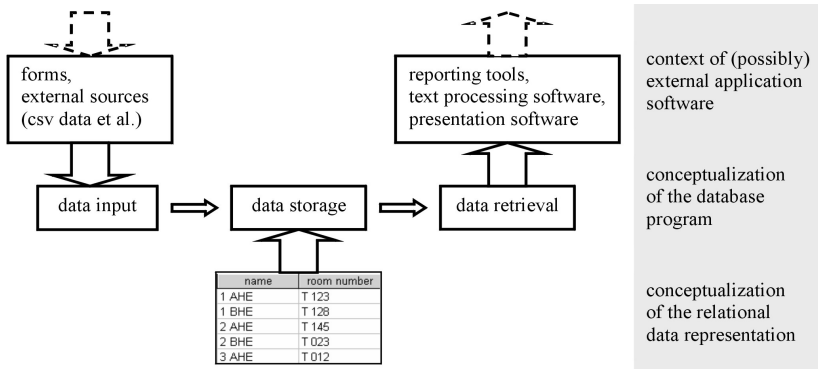


Fig. 1. Working on the »application plane«: Suggestion for a simplified mental model when looking at databases from the view of a user learning to master a database program

Learning to make use of application software is considered basic Information and Communications Technology (ICT) training. Consequently database programs are part of that skills-oriented »ICT-world«, separated from informatics proper. At least this is a common viewpoint of informatics didactics. But in fact databases are of hybrid nature. Modelling and abstraction are the underlying concepts governing the process of database design and these are considered basic concepts in informatics. This adds a second structural dimension when dealing with databases in informatics classes.

From the viewpoint of database design working with databases and programming aim at the same structural educational goal: In both cases the learner is expected to

map a real-world problem onto a model that is suitable for being processed by the computer. In other words, the learner has to add structure to a given real-world problem. This conceptual similarity points at the possibility to apply didactical concepts for programming courses to database design. But the structuring process is predetermined by the data structures or/and control structures of the used software concept. In the case of (procedural) programming these specified structures are arrays, records, branching and loops, the latter guiding the dynamic process of program execution. On the other hand relational databases offer a system of tables to represent a static but fragmented view of the universe of discourse. Therefore the outcome of the modelling process is different when comparing database design with programming. While an executable program can be used without any further structuring effort, databases become useful only if the information that is split among different tables is structured anew by the user. Thus the structuring process when building databases in informatics classes is separated into creating a representative model in the first place and querying the database afterwards. Although this seems to add complexity to database didactics it does help to form an integrated didactical view of databases since queries link this »modelling plane« and the »application plane«, at least from a structural point of view (see Fig. 2).

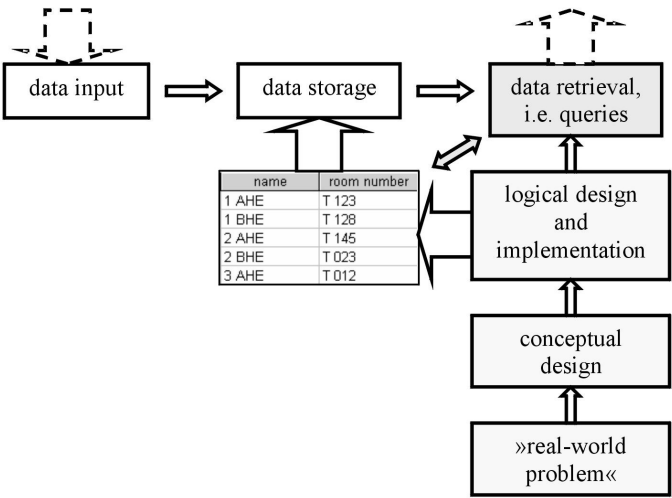


Fig. 2. Database queries seen as a link between the »modelling plane« and the »application plane« of database instruction, helping to get hold of the different structural framework needed by these different views

3 A Concise Survey of Current Didactical Approaches to Databases

Taking stock of current approaches to teach database topics reveals two main categories of instructional/didactical patterns which roughly correspond to the two structural dimensions of databases identified above (see Fig. 3).

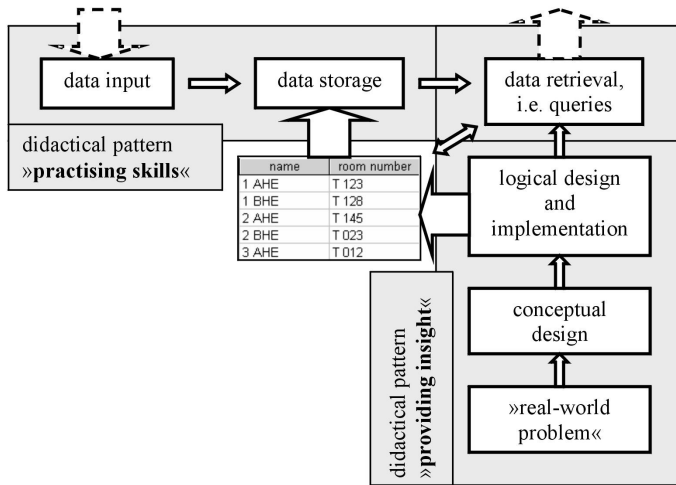


Fig. 3. The two current instructional/didactical patterns, referred to as »practising skills« and »providing insight« throughout this article, mapped onto the identified structural dimensions of the »application plane« and the »modelling plane«

»Practising skills« focuses on the ICT-aspect of how to use database programs. It is of widespread use in adult education and in informatics classes for students at the age from 10 to 15. This pattern grew out of the standardized goals of certificates, especially the European Computer Driving License, ECDL. To some extent courses following this pattern are still designed according to the ECDL syllabus [10], [11]. Aiming at computer literacy at a basic level, this approach presents database programs as an integrated environment for entering, storing, retrieving and presenting data, although data storage is commonly reduced to working with a flat-file database. Therefore a lot of effort is spent on tools like forms for data input or reporting tools (which have little to do with the database core of the program) or on data exchange between application programs at the most. Nevertheless this basic approach has its merits: Building on hands-on activities it allows inexperienced computer users to get acquainted with database software. Moreover it introduces the concept of modularization akin to structuring by dividing the problem into manageable pieces that are dealt with using the proper tools. It also triggered the development of concepts for database instruction in school that go beyond a certificate level. These advanced courses make use of more than one table and dig deeper into structuring by using built in graphical visualization tools and reflecting upon the visualized structure (see [12], [13] and Fig. 4 as an example for graphical visualization).

On the other hand »providing insight« emphasizes database design and corresponds to the structural dimension of the »modelling plane«. Keywords that are associated with this pattern are »conceptual design«, »redundancy«, »normalization« or »SQL« (see [14, 15, 16, 17, 18, 19] for different realizations of this didactical pattern). These keywords indicate that this approach has been inspired by database curricula at the university level and/or by the textbook of Elmasri and Navathe [20]. Informatics courses implementing this pattern at school are usually meant for students with prior computing experience, stemming from ICT instruction or programming

courses. The sequence of topics usually follows the steps of traditional database design, namely analyzing and structuring the posed real-world problem by developing some ER- or UML-model, translating it into a relational (table-) structure, eventually normalizing the relational model and finally exploring the created database by means of queries. Including a lot of structural issues this approach looks fine at the first glance. But there are shortcomings from the didactical point of view, two of which should be mentioned. The practical use of the database is very often dominated by theoretical overhead which (especially in school) could lead to motivational problems on the side of the learners, an aspect that has already been dealt with in earlier publications (e.g. [21]). Secondly, learning at its best should be based on examples [4]. This calls for working with well prepared example-databases to familiarize the learners with the specific structural peculiarities of the relational model before they are expected to design databases by themselves. Consequently there is a didactical necessity to alter the sequence of topics when applying this instructional pattern.

4 Stressing Structure: A Proposal for a Different Approach

As both of the described instructional patterns have their right in their own way it is pointless to ask whether to concentrate on “practising skills” or “providing theoretical insight” in database courses at school. On the contrary it makes sense to combine these approaches in order to get the “essence” of both. Currently and commonly this is achieved by offering successive courses where those implementing the pattern “practising skills” precede instructions aiming to “provide theoretical insight”. Unfortunately, doing databases this way is a matter of time and time is a scarce resource. It is particularly scarce when judging from the situation in Austrian schools, where in secondary academic schools informatics is limited to one year of compulsory instruction in the worst case, and informatics curricula in secondary vocational schools allow for a few months at most for dealing with databases.

The following proposal aims at a different way to combine applying and modelling in database instruction. It relies on well known and well tried concepts but proposes to start with querying ready-to-use databases. This means to rearrange and re-accentuate the parts, to shift the point of view and to use graphical visualization tools like those available with Microsoft Access, which is the most widespread database program in Austrian schools and which will be used as software reference throughout the rest of this paper.

4.1 Choosing the Proper Setting

Databases are used to retrieve information by rearranging relevant data scattered among tables. Therefore the motivation to deal with databases arises from being interested in information stored within a database. Sadly enough, due to the structure of rational databases a lot of information cannot be accessed directly. This is the problem of the user. But this combination of motive and obstacle has to be understood as possible starting point of a learning process. It is up to the teacher to pose tempting problems (queries) and to provide an example-database that is suitable for the posed problems and properly scaled in complexity [6].

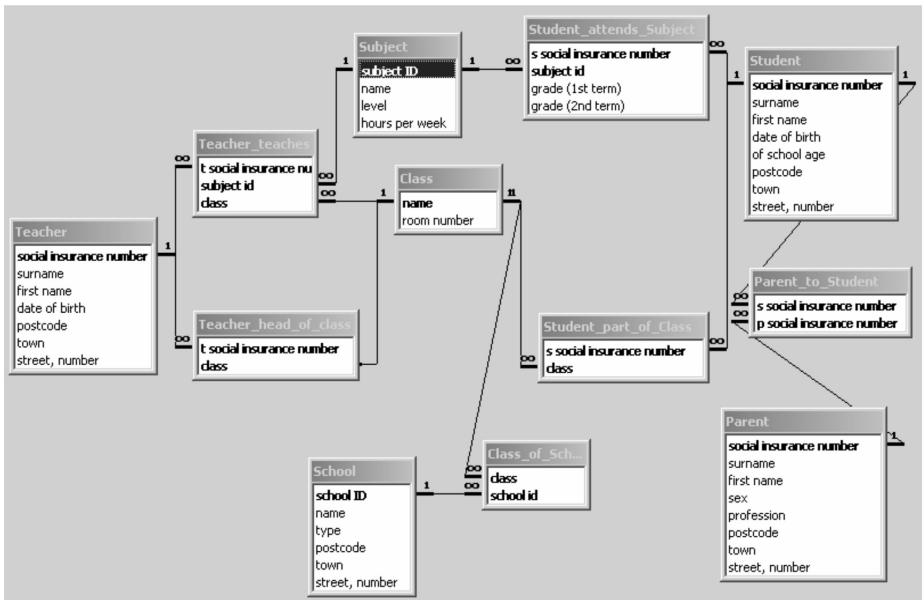


Fig. 4. Example-database of sufficient complexity to allow for non-trivial queries, at the same time illustrating the utilization of the complexity-reducing guidelines suggested in the text. This relation diagram provides a general view onto the structure of the whole database and has to be complemented by diagrams focusing on the structure of single tables.

A database that models some aspect of school-life has still to be considered a good (first) example, allowing for queries students might respond to as they are well acquainted with the modelled system. To raise interest, queries have to touch the students in one way or another, supposedly like: “Are girls doing better in school than boys?” or: “Which teacher gives the most bad marks?” or even: “Do children whose parents are teachers get better marks in school?” (examples taken from [22]). Therefore the database must not be too small, because otherwise queries of the proposed kind would be meaningless (see Fig. 4 for an example-database). On the other hand, the example database must not be too complex either so that the learners can see their way through the table-structure. This apparent contradiction originating in the necessity to add neither too much nor too little complexity to an example database can be solved by

- disregarding certain structural peculiarities of relational databases like aggregation and generalization,
- modelling each association as a table of its own, irrespective of its cardinality ratio,
- helping to distinguish tables that represent associations from tables representing entities (e.g. by applying different naming patterns),
- visualizing the structure of the database.

4.2 Comprehending the Structure

Facing an example-database of the suggested complexity (i.e. a database consisting of a few linked tables) learners first of all have to comprehend its structure and to learn about the structuring elements, especially tables and associations. But distinct from the described patterns of instruction this approach focuses neither on how to manipulate tables at the program level nor are the learners expected to build a structured and suitable model by themselves, starting from scratch. The learners can rather become “explorers” studying the structure of ready-to-use models. To support this, the teacher has to provide basic vocabulary along with “good” examples, guiding exercises and different views onto the data: While a relation-diagram (see Fig. 4) helps to obtain a general view, the microstructure of the database has to be clarified by presenting the design of the individual relational tables and the data contained within the tables (see Fig. 5). This helps learners to distinguish between the table definition and the actual data records.

Feldname		Felddatentyp							
surname		Text							
first name		Text							
sex		Text							
profession		Text							
postcode		Text							
town									
street, number									
	social insurance number	surname	first name	sex	profession	postcode	town	street, number	
	112	Schusser	Eberhard	m	teacher	9020	Klagenfurt	Bahnweg 11	
	113	Schusser	Franziska	f	teacher	9020	Klagenfurt	Bahnweg 11	
	124	Müller	Ernst	m	steel worker	9020	Klagenfurt	Bahnhofstraße	
	125	Müller	Rosalie	f	housewife	9020	Klagenfurt	Bahnhofstraße	
	235	Huber	Christoph	m	teacher	9500	Villach	Hauptplatz 3	
	236	Bartl	Antonia	f	secretary	9020	Klagenfurt	Meisengasse 1	
	422	Maier	Friedrich	m	salesman	9170	Ferlach	Dollichgasse 2	
	423	Maier	Maria	f	shop-assistant	9170	Ferlach	Dollichgasse 2	

Fig. 5. Diagrams sketching the design of the individual relations and actual data records to complement the view provided by relation diagrams

In order to direct the learners’ attention to the fundamental structuring issues, these visualizations have to be accompanied by posing questions like: “Some of the table definitions contain only two or three fields, all of which printed in bold face. Others contain up to eight fields, printed with bold face or standard font. What is the purpose of these different types of tables? Is there a clue in the names of the tables?” or: “What is the difference between fields printed in bold face and those printed with normal font?” or: “Can you think for a reason why there is always a connection between tables that contain fields printed with bold face?”

Clear enough these questions aim at the distinction between entities and associations and at the importance of primary keys, which in the first place should better be called “identifying keys” and “connecting keys” synonymously as these labels point at their role as key attributes in the structural framework necessary to retrieve information from the database.

4.3 Querying: Exploring and Exploiting the Structure

Querying the database is the main issue of this proposed novel approach to database instructions. While basic insight into the relational structure is necessary to specify

queries on a given database, in the suggested course of topics querying in its turn motivates and deepens the learner's understanding of the structural principles: Retrieving information from a database of the proposed kind means to join tables representing entities by making use of identifying/connecting keys to navigate through tables representing associations. Working within properly designed example-databases in this way should provide the learner with a framework for data modelling (using ER- or UML-notation).

But querying the database also introduces the concept of a new language, i.e. a new way to structure the process of problem solving, which calls for another preliminary step, namely to present the peculiarities of SQL or QBE by means of flat-structured queries like "Give the names of the students living in a specific town". From the structural viewpoint that introductory step aims at revealing that the result of a query again is a table which can be used for further queries. Knowing this, structured queries including subqueries can be formulated using modular design. This allows to create tables corresponding to subqueries at lowest level first and to reuse these tables for queries at higher levels. This "divide and conquer" strategy concerns structuring in a twofold way: On one hand, there is the structure of the query itself, on the other hand, the learner has to solve how to structure the modularisation process. According to Polya [23] the latter includes to "devise a plan" and to "carry out the plan". This shall be illustrated with the query "Do students whose parents are teachers get better marks at school?" applied to the example database of Fig. 5:

- Seemingly the first step is to create a table, say: "Parent_is_Teacher", of the students at least one of whose parents is teacher (as one possible interpretation of "whose parents are teachers"!).
- Collecting those data records of the table "Student" that are not in "Parent_is_Teacher" yields a table of students both of whose parents are not teachers ("Parent_is_not_Teacher").
- To use the information stored in the table "Student_attends_Subject" it is necessary to include the students' social insurance number in both tables, "Parent_is_Teacher" and "Parent_is_not_Teacher".
- Based upon the tables "Parent_is_Teacher" and "Student_attends_Subject" it is quite easy to obtain a table of all the grades of all the students "whose parents are teachers" (table "Grades_Parent_is_Teacher"); the same goes for "Parent_is_not_Teacher" and "Student_attends_Subject".
- The last step is to apply the average-function to the list of grades in the tables "Grades_Parent_is_Teacher" and "Grades_Parent_is_not_Teacher" respectively to obtain comparable results.

This is the plan. Devising the plan uses the structure provided by the database and extends it successively by creating new tables. To carry out the plan, Microsoft Access offers "Query By Example" (QBE) and/or SQL for query definition. The QBE-interface offers some additional (structuring) visualization, while SQL allows expressing the query quite similar to everyday English¹:

¹ With Microsoft Access the similarity of SQL statements and everyday English does not hold for JOINS which therefore should be formulated via the QBE-interface.

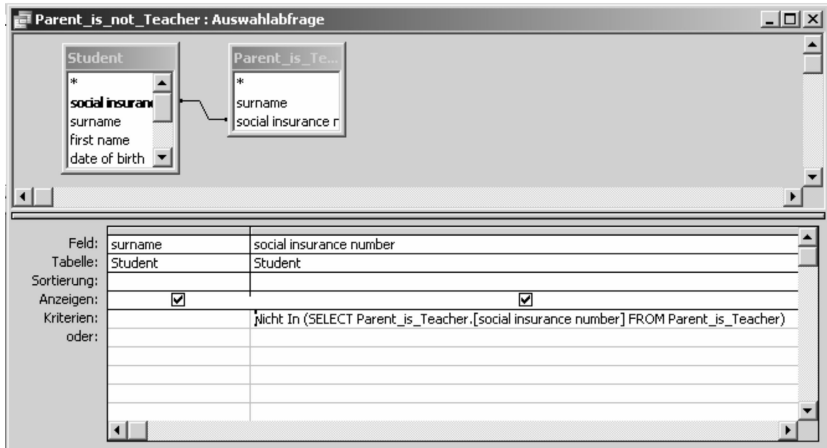


Fig. 6. QBE interface of Microsoft Access, facilitating query definition and offering some visualization

4.4 Applying the Results

Comprehension of the database structure and the specification of well structured queries in particular can form an interface between the “application plane” and the “modelling plane” or the “practising skills pattern” and the “providing insight pattern” respectively: To achieve this, queries should retrieve information that is meaningful to the learner and they should result in insight into the database structure.

Thus »applying the results« on one hand quite uncommonly stands for the learners’ ability to use this structural insight, either to create conceptual data models by themselves or to unveil shortcomings of a given structure. For instance, it is impossible to answer “Are girls doing better in school than boys?” using the database model depicted in Fig. 4, as the relation “Student” does not contain the attribute “gender”. This points at a meaningful completion of the model and stresses the fact, that a didactically well chosen database example usually is not a perfect example in the sense of a real-world representation but an example that triggers questioning and exploration.

On the other hand, the learners should be able to apply the results of the queries in a straightforward manner by creating reports to communicate the results, thus identifying “applying” with “dealing with the application program”. But with queries of the proposed kind it seems to be more rewarding to consider the original meaning of “applying” and to apply these results to the real world by reflecting on the structure of the modelled system. In case of the school-example such a reflection could focus upon socialization processes at schools. Without aiming to neglect the value of “practising skills to master the database program” the last suggestion points at the possibility (the necessity?) to view the application plane in a broader sense, extending it beyond the horizon of informatics. This and the ability to stimulate structural competence prove databases to be a tool of general education.

5 Summary and Further Work

This paper unfolds a concept for a novel approach to database instruction stressing applicability and comprehension of central database issues and the development of structural thinking. This should be accomplished by means of ready-to-use database examples and database queries as essential tools for learners to explore the structure of databases and to retrieve information that is relevant to the learners.

A first effort to apply this concept will be made throughout the rest of this semester (spring 2006) with a group of 16 year-old students at a secondary vocational school in Austria. As these students have no prior experience with databases the central points of this didactical experiment will be to find suitable database examples that are sufficiently complex, to find proper ways to guide the structuring process and to look for ways to support the process of query definition by providing visual aids that go beyond the visualization tools of database software products. All of these issues seem to be absolutely necessary for developing structural competence on the side of the learners by means of database instructions.

References

1. Siebert H.: Pädagogischer Konstruktivismus. Luchterhand, München/ Unterschleißheim (2003)
2. Girmes R.: [Sich] Aufgaben stellen. Professionalisierung von Bildung und Unterricht. Kallmeyer, Seelze (2004)
3. Nolan J.D.: Conceptual and Rote Learning in Children. In: Teachers College Record Volume 75 Number 2, 1973, p. 251-258
4. Spitzer M.: Lernen. Gehirnforschung und die Schule des Lebens. Spektrum, Heidelberg, Berlin (2003)
5. Sternberg R.J., Martin M.: When Teaching Thinking Does Not Work, What Goes Wrong? In: Teachers College Record Volume 89 Number 4, 1988, p. 555-578)
6. 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)
7. Swertz C.: Didaktisches Design. Ein Leitfaden für den Aufbau hypermedialer Lernsysteme mit der Web-Didaktik. Bertelsmann, Bielefeld (2004)
8. Meder N. et. al.: Web-Didaktik. Bertelsmann, Bielefeld (2006)
9. Micheuz P.: 20 Years of Computers and Informatics in Austria's Secondary Academic Schools. In: Mittermeir R. (ed.): From Computer Literacy to Informatics Fundamentals. Lecture Notes in Computer Science Vol. 3422, Springer, Berlin-Heidelberg (2005)
10. <http://www.ecdl.com/main/download/ECDLV4SWG110159.pdf> (ECDL/ICDL Syllabus Version 4.0; access on Feb. 25th 2006)
11. <http://www.ecdl.com/main/download/MAM5.pdf> (Module AM5, Database, Advanced – Level, access on Feb. 25th 2006)
12. Habenicht K.: Grundlagen der Datenverarbeitung mit MS Access, ADIM-Schriftenreihe Wissen in Beispielen, Band 205, 2005 (<http://www.adim.at/wib/>)
13. Habenicht K.: Fortgeschrittene Datenverarbeitung mit MS Access, ADIM Schriftenreihe Wissen in Beispielen, Band 206, 2005 (<http://www.adim.at/wib/>)

14. Borg B.: Didaktisch-methodische Aspekte des Einsatzes von Datenbank-systemen. In: Log In 7, Heft 5/6 (1987)
15. Witten H.: Datenbanken – (k)ein Thema im Informatikunterricht? In: Log In 14, Heft 2 (1994)
16. Penon J. und Spolwig S.: Video-Center. In: Log In 14, Heft 2 (1994)
17. Mittermeir R., Kofler E.: Informatik-Einführungsunterricht für Lehramts-kandidaten mit besonderer Berücksichtigung geisteswissenschaftlicher Fächer. In: Hüffel C., Reiter A. (eds.): Praxis der EDV/Informatik, Jugend & Volk, Wien (1996)
18. Bierschneider-Jakobs A.: Datenmodellierung und Datenbanksysteme. In: Log In 127 (2004)
19. <http://www.e-teaching-austria.at/AINF/> and http://www.e-teaching-austria.at/ainf_/datenbanken (access on Mar. 5th 2006)
20. Elmasri R. and Navathe S.B.: Fundamentals of Database Systems. Addison Wesley, Reading et al. (3rd ed. 2000)
21. Urban S.D., Dietrich S.W.: Integrating the Practical Use of a Database Product Into a Theoretical Curriculum. In: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, Philadelphia, February 15 - 17, 1996, 121-125 (1997)
22. Modrow E.: Zur Didaktik des Informatik-Unterrichts, Dümmler, Bonn (1992)
23. Polya G.: How to Solve It, Princeton, New Jersey (2nd ed., 1988)

Handling the Diversity of Learners' Interests by Putting Informatics Content in Various Contexts

Evgenia Sendova

Institute of Mathematics and Informatics, Bulgarian Academy of Sciences,
"Acad. Georgi Bonchev" str. Bl. 8, Sofia 1113, Bulgaria
jsendova@mit.edu

Abstract. In an attempt to answer the need of a more natural integration of informatics with other school disciplines this paper presents a set of scenarios for putting informatics tools in the context of mathematics, art, and literature. Having introduced scenarios of this kind into informatics courses for in-service and pre-service teachers, the author shares her impressions from this experience. These impressions are drawn from her personal observations and the feedback from her students at Sofia University and the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences.

1 Introduction

Learning is generally accepted to be a process of gaining experiences that give rise to particular viewpoints, or ways of developing abilities *to see, think, do and be in the world* [1]. Therefore, it must be rooted in context.

In the last decades there have been many discussions on the role of informatics and ICT for secondary school education. Two extreme views in the Bulgarian educational system are those of "students as programmers and successful participants in Informatics Olympiads" vs. "students as proficient users of software packages" [2]. Consistent with the first approach, authors of informatics textbooks concentrate heavily on the content, leaving out the context – one could see numerous examples of textbooks in which the notion of *cycle* is introduced in the context of adding up natural numbers as the *easiest possible illustration of a cycle*. Paraphrasing Godfrey Harold Hardy, we could claim that if the fine arts were taught in the same way as informatics, they would be reduced to *studying techniques for clipping stone and mixing paints* [3].

To help students reach a higher level (i.e. seeing patterns in ideas), it seems natural to introduce informatics notions in the style of *how to* rather than of *what is*. This has already had an impact on the way we teach mathematics, art, music and literature since they deal with symbols, images, transformations and operations [4]. Thus, the modeling of phenomena in each of these domains becomes natural if we use appropriate computer environments facilitating experimentation, hypothesis generation, testing, and open-ended exploration.

1.1 A Glance Back

Let us briefly touch on some significant attempts to integrate informatics in the Bulgarian school curriculum. The first attempt to integrate informatics in the context of languages and mathematics was the introduction of the subject *Language and Mathematics* for the junior high school in the frames of an educational experiment (1979–1991) of the Research Group on Education (RGE) [5].

Extending the success of the RGE informatics project, a team of mathematicians and informaticians has written the textbooks *Mathematics and Informatics for High School Students* (8th–12th grade) for the general educational system [6]. These textbooks were introduced in 1988 in many regular secondary schools. Some general ideas adopted in these textbooks included: offering a means to clarify and extend the mathematical concepts studied, using problem solving scenarios [7], and specifying branches in mathematics, linguistics and arts in which students could gain new ways of expressing themselves by means of informatics.

Within the small pilot schools, this idea worked reasonably well, since motivated teachers were supported by the researchers. But the results were very different on a larger scale, because the novelty was too big for the non-informatics teachers, who would either skip the informatics corners or leave them to the informatics teacher, thus reducing the chance for a real integration. The informatics teachers, in turn, did not like the limitations imposed by the project scenarios; they favored the more traditional systematic introduction. After all, if the methodology doesn't resonate with the teachers, it will not resonate with the students.

1.2 A Step Forward

Based on the experience gained and the recommendations of the teachers a new textbook, *Informatics in Logo Style*, was developed [2] to introduce some basic concepts and methods of informatics in educationally rich context. One of the textbook's modules discussed applications to Arts, Literature and Linguistics. It was designed especially for *those who thought they did not like informatics*; it let them choose their own way towards making best use of informatics means. Again, the hopes of the authors that such a rich context would change the image of informatics as a "dry" subject were not justified on a large scale. Before this can occur, a critical mass of teachers must feel comfortable in open-ended explorations, in crossing the borders of the subject of their expertise. The solution was to start again from the beginning.

1.3 Educating the Teachers

The traditional core of mathematical disciplines, as taught at the Faculty of Mathematics and Informatics at the Sofia University to pre- and in-service teachers has been enriched (since 1989) by courses in which the introduction of informatics foundation was put in a rich educational context: *Teaching Mathematics in a Laboratory Type Environment*, *Informatics in the Secondary School Mathematics*, and *Problem-Oriented Languages*. To this end, the lecturers involved their students in the developments of projects in the spirit of *programming to learn something rather than just learning to program*. After being exposed to some scenarios relating informatics

to mathematics, arts, language and other subjects, the students would choose a project on a topic of their interest [8]. As a result, the future informatics teachers became better prepared to teach the subject in various contexts. Further, they are open to collaboration with colleagues from other fields.

In the professional development of teachers from other fields, it is important to convince them both that it is the spirit of their subject that should dominate and that use of computers in their activities should be well grounded in the context of their subject. What follows are some examples of scenarios illustrating these ideas.

2 Informatics Marrying Mathematics and Arts

The formal definition of a *fractal*, given by Mandelbrot, is: *a set for which the Hausdorff–Besicovitch dimension strictly exceeds the topological dimension* [9]. Hardly anyone would dream of introducing such a definition in high school. A looser definition for the term is a *self-similar object*, a geometric shape that can be subdivided into parts, each of which is a smaller copy of the whole. Let us see how the self-similarity makes the introduction of fractals quite natural, upon integration of their exploration with one of the most powerful informatics tools, the *recursion*.

2.1 Fractals as Non-traditional Functions

For most students, the word *function* brings to mind a quadratic or trigonometric function. The input and output of a function may be given in a less traditional way, e.g. to define a fractal (Fig. 1):

$$F(\text{—}) = \text{—}\text{┐}\text{┌—}$$

Fig. 1. Defining the Koch fractal with a square bent as a function of segment

In words, this correspondence implies that we replace every input segment by a ‘bent segment’, where the shape of the bend is a square of side length $1/3$ of the input segment. The Koch curve is the limiting curve obtained by applying this construction infinitely many times. The recursive procedure **F** below reflects the nature of self-similarity of the Koch curve (**s** is the segment’s length and **i** – the iterations remaining):

```
to F :s :i
  if :i = 0 [forward :s stop]
  F :s / 3 :i - 1 left 90
  repeat 3 [F :s / 3 :i - 1 right 90] left 180
  F :s / 3 :i - 1
end
```

When executing the procedure **F** for different values of the inputs, even experienced math teachers were surprised to recognize patterns from everyday life, as in, for instance, motives of traditional Bulgarian folk embroidery (Fig. 2).

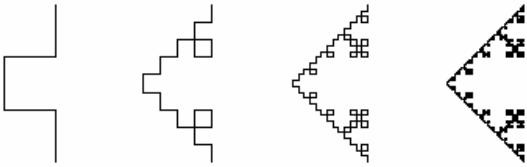


Fig. 2. Four consecutive iterations of **F** (the Koch curve with a square bend)

If we replace the sides of a square by Koch curves, we obtain Koch flakes (Fig. 3). Creating Koch flakes and computing their area and perimeter helps students gain a good intuition about such fundamental mathematics notions as *infinite sequences* and *series*, as well as a good visualization of *convergence*. Besides, it is not an everyday experience to see curves which could surpass the distance between the Sun and the Earth and still fit in the computer screen [10].

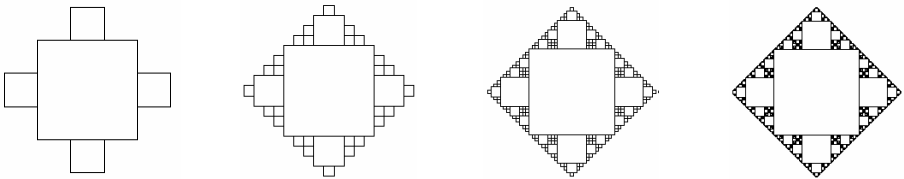


Fig. 3. Four consecutive iterations of the Koch flake imposed on one another

When working with fractals the students are involved not only in problem solving, but also in the posing of problems meaningful to them. They explore questions of the kind: *What if we change the initial conditions? Are there fractals of bounded perimeter and unbounded area, of unbounded both perimeter and area? What generalization of the function F would produce “interesting” fractals?* And as we know from the title of Cantor’s Ph.D. thesis, *In mathematics the art of asking questions is more valuable than solving problems*. Having learned to apply the “what if “strategy, students could vary the function and observe the effect (Fig. 4).

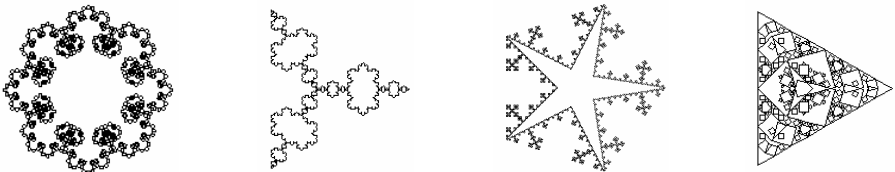


Fig. 4. Variations on the Koch flake with a bend of different shape and direction

It was interesting for the in-service teachers to learn that the Koch curves belong to the more general class of *fractals* that today have a great technical application in the computer graphics. Not less important for them was to learn that some of the topics in the context of fractals are typical for modern mathematical fields, such as fractal analysis, the theory of dynamic systems, coding and image compression.

The first attempts to enrich the traditional teacher-training curriculum with elements of the fractal geometry show that the pre- and in-service teachers are highly motivated to experience new topics, to formulate interesting problems, and to discuss methods for solving them. They experience mathematics as a creative activity in which they *not only learn the rules of the game but play the game themselves*. The hope is that after experiencing this style, the teachers implement it in the classroom, thereby conveying the approach to their students.

2.2 The Amorous Turtles Problem and Vasarely

The tools of informatics, in addition to helping us attack problems that were previously considered “monsters”, could also be used to gain new insights into well-known problems from the popular math books. For example:

In every corner of a square there is a turtle. Each turtle is in love with the next one (forming a cycle), and starts walking directly towards the object of its feelings with the same constant speed. Will they ever meet and what will their paths be?

The problem can be solved in a variety of ways, but one can get a good idea of the movement through a relatively simple computational model. By commanding each Logo turtle to turn towards its *beloved* and move one step forwards and repeating this sufficiently many times, the students get the following picture (Fig. 5):

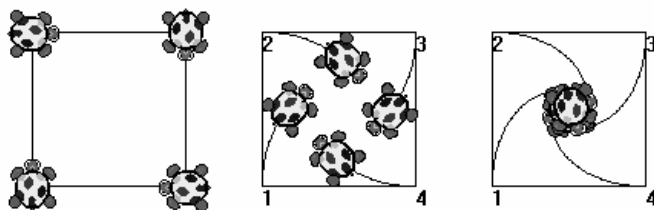


Fig. 5. Consecutive positions of the amorous turtles and their trajectories

The path of each turtle is a part of the *logarithmic spiral*, a famous curve occurring in nature. It is the shape of the shell of a *Nautilus*, and also in the graph of a deposit with fixed compound interest (Fig. 6). Indeed, when constructing a computational model of a problem, we usually solve an entire class of related problems.

It is easy to vary the starting conditions (e.g. the number of the turtles, their initial positions). Further we can connect the consecutive positions of the turtles by segments and notice that they are always in the corners of a square whose side shrinks with the speed of the movement of the turtles.

The interesting graphics obtained by varying the starting conditions suggest the idea that this model could serve to generate computational variations of paintings by Vasarely, e.g. six turtles with constant and different speed, two turtles moving along the coordinate axes and a third one describing curves of Lissageou, then the first two moving along polygons (Fig. 7). The position of the drawing turtle could also be determined as a vector sum of the vectors determining the movements of the first two turtles [11].

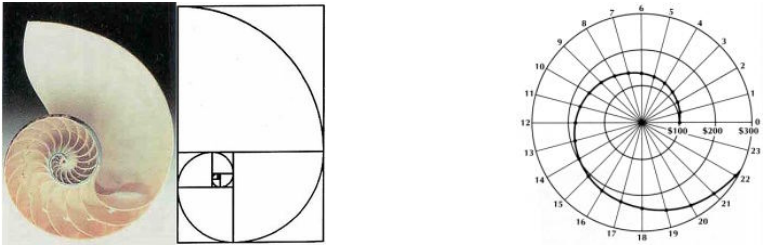


Fig. 6. Logarithmic spiral as seen in a *Nautilus* shell (left) and in growth of a deposit with fixed compound interest (right)

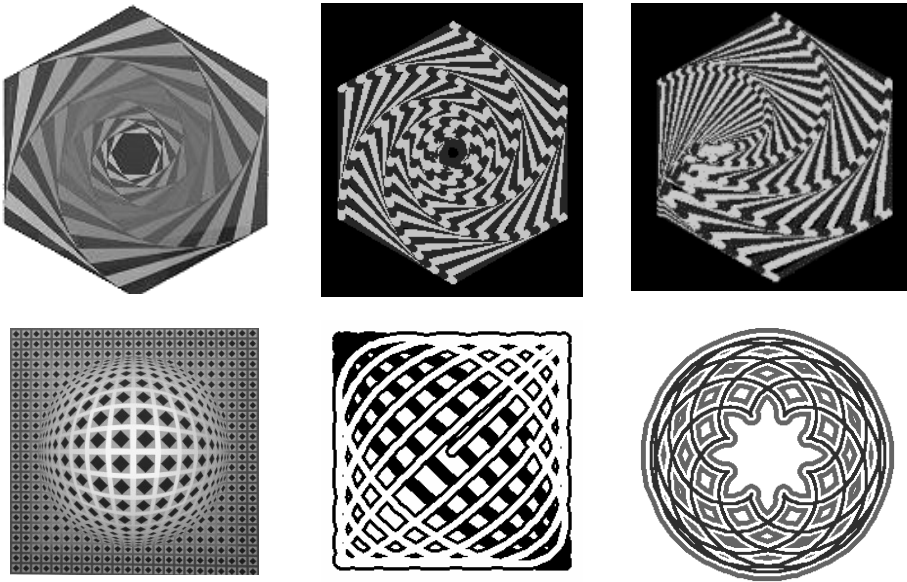


Fig. 7. Vasarely's *Los Angeles* and *Vega* (left, top and bottom, respectively) and four computational variations based on the Amorous Turtles problem

In this exercise, the creative process could be interactive: after leaving the frames of the strict imitation, the students could be inspired by new combinations of forms and colors and could get new insight, which could in turn lead to a new formalization. Establishing such interactivity makes the process creative in the standard understanding of *creativity* as a human quality. At minimum, the students gain a new appreciation of abstract art through visual modeling: “I am looking at the paintings of Vasarely and Kandinski with new eyes now!” said a teacher-to-be who brought with her a small reproduction of Vasarely's *Los Angeles* and asked if it would make a good topic for a project.

3 Modeling Some Phenomena in Literature

The word “poetic” somehow seems contradictory to the words *informatics* and *information technology*. But when it comes to the *concrete poetry*, the connection with informatics looks more natural.

3.1 Making Concrete Poetry Even More Concrete

The *concrete poems* are characterized by a form related to their content. This makes the computer very suitable for illustrating an underlying concept in a concrete poem. A good example of such a poem is ‘40–Love’ by Roger McGough [12] in which the words follow the trajectory of a tennis ball and their motion could be well modeled by using the so called *turtle text* in Comenius Logo (Fig. 8):

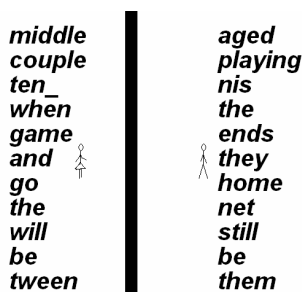


Fig. 8. The final state of a dynamic presentation of ‘40–Love’ by Roger McGough

Together with dynamic appearance of the text, the user could listen to the sound of a tennis ball and follow its image jumping as a shape of the turtle. Such forms of poetry are stimulating to the imaginations of the students. Students could then be further involved in finding and illustrating existing concrete poems or in creating their own animated poems – about the rain or the snow (the letters falling like raindrops or snowflakes), or about objects of specific shape, the words forming dynamically the shape. Of course, using the potential of the computer for more adequate presentation of poetry requires important skills. But it is much more challenging to *make the computer become a poet*.

3.2 Are There Rules in Poetry?

A well-known poet has said that *poetry without rules is like a tennis match without a net*. The great masters in various fields of art are innovative creators who on one hand extend our imagination of what is possible in the frames of specific rules and formulae and on the other hand create new rules to be followed by the next generations. Thus, what makes their work great is not the lack of rules but rather their ability to be flexible and inventive in the frames of limits they themselves choose.

While the form of a poem is not one of its basic characteristics, it plays an essential role in this genre. Nonetheless, it is hardly motivating for the students to learn the

structure of poetic works as part of the traditional literature classes. Even Mayakovsky in his essay *how to make poetry* [13] writes that in his work he has never had to know what *iamb*, or *dactyl* were: *These things take 90% of the textbooks in literature but only 3 % in my practice*. Still when describing his difficulties in a poem about Essenin, he uses a formal structure description of the kind (**ra** shows an emphasized syllable):

ra-ra-**ra** / ra-**ra** / ra-ra-**ra** / ra-**ra** /

and tries to find the most suitable words and phrases fitting the structure.

Modeling various poetic forms by means of informatics is one more way to analyze and formalize a specific poetic structure and to cultivate the ability of recognizing valuable works in this genre.

One of the tasks for the students in the textbook *Informatics in Logo Style* [2] is to write variations on the poem *Chant* by Kenneth White, whose poetry combines the influence of the old Eastern poetry with modern lyrics.

The rhythm of each verse can be described symbolically by reflecting the number of words, syllables and the emphasis as sequences of **DA** and **di** (where DA stands for an emphasized syllable). Here is the Bulgarian translation of the first verse (originally in French), its symbolic representation and the English translation:

<i>Пръч/ки от бре/зи/те</i>	DAdi_di_diDAdi	<i>Birch rites</i>
<i>тре/во/ля/ци нус/ти</i>	didiDAdi_DAdi	<i>Empty moors</i>
<i>не/бе/са су/ро/ви</i>	didiDA_diDAdi	<i>Raw skies</i>
<i>сняг не/ве/ро/я/мен</i>	DA_dididiDAdi	<i>Incredible snow</i>

If we are interested only in the melody of the verse and neglect the number of the words we could reduce the rhythm figures to the three basic types: 1) DadididiDAdi, 2) didiDAdiDAdi, and 3) DadiDAdiDAdi. Then the structure of the whole poem (which consists of four verses) could be written symbolically as follows:

1-2-2-1__1-2-1-1__2-1-1-3__1-1-1-2

If we enter the structure as a list whose elements correspond to the number of the rhythmic group we could use a Logo procedure of the kind:

```
to verse :structure :rhythm
  if empty? :structure [stop]
  print run item first :structure :rhythm
  verse butfirst :structure :rhythm
end
```

Then in order to get poems as close to the original as possible we could give the command:

```
verse [1 2 2 1 1 2 1 1 2 1 1 3 1 1 2]~
      [[DAdididiDAdi] [didiDAdiDAdi] [DAdiDAdiDAdi]]
```

where DAdididiDAdi is the name of a procedure generating a phrase of the corresponding rhythm through combining randomly chosen words from lists provided by the students.

The verse procedure is general enough as to be applied as a generator for verses with various structures and rhythm. Here again, the idea behind modeling the creative

process can be formulated as follows: we study and analyze a concrete poetic work and create a procedure that could generate this work and close variations. Then, we make a generalization which could still generate the original but could also generate other works of the same author. Finally, we could generalize the procedure so as to generate works with an arbitrarily chosen and then fixed structure.

Depending on the types of poetic structure that the teacher would like to consider, s/he might encourage students to explore some structure ideas, e.g. rhymes of the type ABAB, or verses following the structure of various representatives of the Eastern poetry (e.g. *haiku*, *cinquain*, *tanka*, *sedoka*, *sijo* and related genres).

Identifying the structure of a poem would not be a goal *per se* as it is often the case in the traditional classes of literature, but also a *part of creating a model of the process behind its creation*. Such a modeling would hopefully motivate the analysis of a given literature work in a learning context. But not only. Our experiments with students and teachers have shown that we should not be too skeptical about the *computer works*. They are neither purely *computer*, nor *random*. In poetic models, we always specify the structure of the poem and the sources the computer is to choose from. Thus, an important application of computer-generated works is that they are a sort of “invitation for an intellectual dance” [14] in which the best verses, containing word combinations unexpected for the human, could be used as a source for further processing according to his taste and visions.

3.3 Harnessing Associative Lists for Aphoristic Associations

Let us consider an example of teaching the computer to generate variations of two aphorisms in order to demonstrate how the analysis of their humoristic nature could be integrated with some linguistic notions:

The horse started resembling a donkey – like a translation from German to Dutch.

Georg Lichtenberg

Poetry is the synthesis of hyacinths and biscuits.

Carl Sandburg

When analyzing the first aphorism, the students realize that in order to convey the idea how much can be lost in translation the author compares couples of objects, the first being *similar* in a certain sense to the second but excelling it. Therefore they could use an associative list of the kind:

[[horse donkey]][masterpiece kitsch][wine vinegar][watermelon pumpkin]]

to generate variations of the above aphorism

One possible generator of aphorisms in the style of Lichtenberg is given by:

```
to APHORISM :List1 :List2
  make "Couple1 pick :List1
  make "Couple2 pick :List2
  output(sentence "The first :Couple1 ~
    [started resembling]~
    last :Couple1[- like a translation from]~
    first :Couple2 "to last :Couple2 )
end
```

The students are usually surprised by the great variety of computer aphorisms, especially if they have created long enough associative lists. Here are some results after executing the command:

```
PRINT APHORYSM [[horse donkey] [wine vinegar][masterpiece kitsch]~
[[German Dutch][French Italian][English German][Russian Bulgarian]]
```

The wine started resembling vinegar – like a translation from Russian to Bulgarian.

The masterpiece started resembling kitsch – like a translation from English to German.

Of course the success of a joke depends on many factors but the important thing in such an activity is that students grasp the structure of the genre, that they understand what is fixed and what could be modified so that the sense of the aphorism would remain the same.

When modeling the second aphorism given above, the students are expected to make an associative list of words which differ semantically, e.g.:

```
[[hyacinths biscuits][diamonds puddles][stoves orchids][perfume glue]]

to SYNTHESIS :List
  make "couple pick :List
  output (sentence [Poetry is the synthesis of]~
           first :couple "and last :couple )
end
```

Here are some *creations* of SYNTHESIS with the above associative list as input:

Poetry is the synthesis of diamonds and puddles.

Poetry is the synthesis of perfume and glue.

Making associative lists of semantically *close* or *distant* words might be an interesting learning activity in a language context.

As for the computer-generated poems or aphorisms, the literature teachers are better suited to judge their value than informatics teachers are.

4 What Next?

Recent surveys in the labor market identify the so-called *soft skills* as crucial for the knowledge-based economy. These skills include: knowledge presentation, working on projects, problem solving, and communication skills. Such *soft skills* could be viewed as an essential part of the ICT skills, thus we can speak about *enhanced ICT-skills*.

4.1 I*Teach – An International Project for Innovative Teachers

The *Innovative Teacher (I*Teach)* is a *Leonardo* project [15] developing methodologies, approaches, and tools targeted at day-to-day utilization by the teacher trainers and teachers of these *enhanced ICT skills*. In its first phase, a set of *active learning methods* relevant to the effective acquisition of *enhanced ICT skills* has been identified [16]. In the next phase, an *I*Teach handbook* with methods and tools for building up *enhanced ICT skills* will be developed. In harmony with the project's ideas, a series of scenarios of the above kind will be presented as tasks in the context

of an active learning environment. Students will be directed to a general goal (producing a specific product) via a path (the learning process) traced by *milestones* (intermediate objectives). At each milestone, pupils are expected to have finished a concrete stage of the product development and mastered a concrete skill. By passing along the *milestones*, the students will be expected to build up a set of *enhanced ICT skills* interwoven with predetermined teaching objectives.

4.2 The Fractal Scenario in New Clothes

We imagine the *Fractal Scenario* in the context of *I*Teach* as follows: the students develop an interactive version of the program in which they invite the potential users to explore the effect of all the parameters involved. A further task for them is to model the generation of some famous fractals (e.g. the curves of Koch, Hilbert, Peano, Cantor) and to explore their dimension. Another interesting assignment would be to add a new functionality to their programs, viz. to enable the users to generate their own fractals in an interactive mode. All these activities are in harmony with the *learning by doing* method and could contribute to building up many *enhanced ICT skills*. Indeed, these tasks emphasize planning, dividing a task into subtasks, and, more generally, skills closely related to the teamwork and care for the potential user and team partner (coding standards, collective ownership).

5 Conclusions

Our experience, both with students and teachers in informatics, confirms the belief that most interesting results from an educational point of view are observed in the cases where the power of programming is harnessed in exploratory activities of significance to the user. When given the chance to concentrate on the phenomena to be modelled by means of informatics, the representatives from other fields become more powerful in their own domain, both as teachers and researchers. And when more teachers enter the school as partners of students in a research process, informatics will itself become a favorite partner for the other subjects.

Acknowledgments

This research has been partially supported by the Leonardo da Vinci project *Innovative Teacher (I*Teach)*, BG-05/PP/166038. I would like to thank Lance Rhoades whose expertise in comparative literature didn't deter him from considering computer works. My deepest gratitude goes also to my students Scott Kominers and Brown Westrick for helping the donkey resemble a horse.

References

1. Wertsch, J.V. (1998). *Mind as action*. New York: Oxford University Press
2. Dicheva, D., Nikolov, R., Sendova, E. (1997) *School informatics in Logo style: a textbook facing the new challenges of the Bulgarian informatics curriculum*, in M. Turcsanyi-Szabo (Ed.) *Learning and Exploring with Logo*, Proceedings of the Sixth European Logo Conference Eurologo'97, Budapest, Hungary, 20–23 August

3. Grozdev, S., Derzhanski, I. Sendova, E. *For Those Who Think Mathematics Dreary*, <http://www.math.bas.bg/ml/iad/dremat/dmathen.html>
4. Sendova, E. (2001) *Modeling Creative processes in Abstract Art and Music*, Proceedings of the 8th European Logo Conference 21–25 August, Linz, Austria
5. Sendova, E. (2000) *Technology in Education from different perspectives*, Informatika, Nr. 2 (36)
6. Sendova, E (2002) *Computer Revolution – A Revolution in the Way We Express Ourselves*, Zborník príspevkov z 3. celoštátnej konferencie INFOVEK, Modra – Harmónia, 9.–12. 10. 2002
7. Sendova, E., Nikolov, R. (1989) *Problem Solving Scenarios in Secondary School Textbooks in Informatics and Mathematics* (with R. Nikolov), in G. Schuyten & M. Valcke (eds.) Proceedings of the Second European Logo Conference, Gent, Belgium
8. Nikolova, I. Sendova, E. (1995) *Logo in the Curriculum for Future Teachers: A Project-based Approach*, in Micheal O Duill (Ed.) Building Logo into the School Curriculum – Eurologo Proceedings & Outcome, affiliated to IFIP WCCE/95
9. B. Mandelbrot (1977) *The Fractal Geometry of Nature*, W.H. Freeman and Co., NY
10. Goldenberg E. P. (1989) *Seeing Beauty in Mathematics: Using Fractal Geometry to Build a Spirit of Mathematical Inquiry*, Journal of mathematical Behavior 8
11. Blaho, A., Kalas, I., (1998) Learning by Developing, Logotron
12. McGough, R., 40 Love, <http://www.rogermcgough.org.uk/cd/sound/40-love.mp3>
13. Маяковский, В. (1959) *Полное собрание сочинений*
14. Goldenberg, E.P., Feurzeig, W. (1987) *Exploring Language with Logo*, The MIT Press, Cambridge, MA,
15. Leonardo Project PP-166038 *I*Teach (Innovative Teacher)* <http://i-teach.fmi.uni-sofia.bg>
16. Sendova, E., Stefanova, E. (2006) *Analytical report on active learning method, appropriate for building up Enhanced ICT Skills*, <http://i-teach.fmi.uni-sofia.bg>

Computer Science in English High Schools: We Lost the S, Now the C Is Going

M.A.C Clark and R.D. Boyle

School of Computing, University of Leeds, UK
{martyn, roger}@comp.leeds.ac.uk

Abstract. High Schools in England began to teach computing long before the advent of the personal computer and the graphical user interface made it possible to teach aspects of information technology skills to non-specialists. We demonstrate that early high school computing was highly consistent with academic and professional practice. This consistency was eroded from the mid-1980s when the English school curriculum began to emphasise IT skills at the expense of computer science. We explain this development and argue that it helps to account for the growing difficulty in attracting students to university study in the discipline. Finally, we consider some current developments in the English high school curriculum, and reason that they raise concerns that they worsen prospects for the study of computer science.

1 Introduction

From about 1969 onwards a qualification in computer science¹ has been available to high school students. However, the widespread availability of personal computers, with their graphical user interfaces, has coincided with a decline in the study of computer science as alternative qualifications in information technology become more popular.

As a part of a broader project investigating the relevance of school level qualifications to success in university study (see, for example [1], [2], [3], [4]), we begin here by presenting an overview of the first phase of computer science in English high schools. This phase began in the late 1960's and ended in approximately 1985. We argue that, although take up was relatively small, the discipline as it existed in schools was highly consistent with professional and academic practise.

Secondly, we show how the curriculum changed as the accessibility of computer technologies.

Secondly, we show how the curriculum changed as the accessibility of computer technologies increased dramatically and national priorities led high schools to focus on the concept of information technology (IT) literacy. Thus from the mid-1980s a variety of qualifications were developed that required learning relating to topics such as files and directories, and the development of skills in using

¹ We use the term 'computer science' and 'computing' interchangeably, but emphasise an important distinction between these terms and 'information technology'.

the applications that have become familiar in ‘office’ suites – word processing, spreadsheets and database management systems. Probably the most well known of these qualifications is the European Computing Driving Licence (ECDL) [5]. Finally, we consider some current developments in the English high school curriculum and consider what the future might be for teaching and learning of computer science. Specifically, we raise concerns that the new generation of IT courses will make it yet more difficult to recruit the computer scientists of the future. The paper is presented as follows. Section 2 presents background to the English school system and section 3 offers an account of the project methodology. Subsequent sections consider respectively, the first phase of high school computer science, the emergence of IT skills teaching, and current and future developments. Recurring themes of these sections include the subject matter of the curriculum, the availability of computing resources and teachers, and the pattern we perceive whereby computer science is driven out of the high school curriculum in favour of low-level skills in using computer applications. The final section provides a summary and conclusions.

2 Background

Although other examination systems exist in English high schools – most students will take GCSE examinations at age 16, there are vocational alternatives to both GCSE and A-levels, and interest in Baccalaureate type examinations appears to be growing – our focus is primarily on the A-level system.

As we noted in [1], compulsory education in England ends at age sixteen but university entrance is generally predicated on the passing of A-level examinations taken at age eighteen. These examinations are subject-based and high schools students typically attempt three or, perhaps, four subjects ². Despite periodic revision, including a major overhaul at the end of the 1990s, the A-level qualification has existed since the 1950s and is familiar to politicians, parents and employers. Indeed, trust in the standing of the qualification explains why A-levels have acquired the cachet *gold standard* – a reference to a financial system in which the currency is kept at the value of a fixed weight of gold.

As we explained, A-level examinations are run by boards that exist for the purpose: their number and names change with time but it is important to note that they are independent of government. Boards publish specifications setting out the syllabus to be examined and devise the methods of assessment. They also organise the setting and marking of assessment instruments. The modern supervisory regime, which is overseen by the Qualifications and Curriculum Authority (QCA) – a non-departmental public body sponsored by the Department for Education and Skills (DfES) – ensures that for a given subject the specification of each board is broadly comparable, though they are not identical.

For university entrance passing two A-level subjects at grade E or better is typically a minimum requirement but popular and prestigious institutions or

² The examination of each subject may involve more than one paper. Similarly, there may be an element of coursework or the requirement to undertake a project.

courses will require more: Three subjects at grades BBB is a likely minimum for admission to one of the ‘Russell Group’ [6] of research universities. Further, many disciplines impose more specific requirements. For example, degree programmes in physics will, typically, require recruits to have passed an A-level in the subject.

3 Methodology

As suggested above, this work represents part of a project investigating the preparation of students for the study of computing in university degree programmes. We began by examining the performance of a cohort of undergraduates and considering the correlation between achievement in a computing degree programme and qualifications on entry. In particular, we considered whether undergraduates who had passed A-level computing attained better grades at university than their peers who had no prior experience in the discipline. Aspects of the research are reported in [1,2,3,4].

Based upon this work we developed the working hypothesis that school level qualifications in the discipline are somewhat removed from expectations at undergraduate level and, therefore, we set out to investigate the content of the high school computing curriculum. Data were gathered from syllabi, examination papers, examiner’s reports and other relevant documents. Following a grounded approach [7,8,9], analysis was conducted concurrently with the collection of further data. The working hypotheses were discussed in the light of new data and amended accordingly. For example, discussion with colleagues who remembered tackling A-level computing as high school students suggested that the connection between the university and high school curriculum in the discipline was once much closer. This was incorporated into the working hypothesis and further data relating to previous curricula were gathered from examination board archives. We present here the current version of our understanding, acknowledging that the discovery of further data may result in further modification of the theory.

A feature of this methodology is that review of the literature is delayed. Indeed, secondary sources are treated as just one more source of evidence in the evolution of the working hypothesis. Therefore, rather than including a literature review section, relevant materials are cited as they become relevant.

4 1969–1985: Computer Science

Computer science has been taught in English high schools since at least 1969 – remarkably early considering that before 1957 computer science at English universities was restricted to the pioneering computer building projects at Manchester and Cambridge and that the first degree programme in the discipline was launched only in 1964 [10,11].

Archival material offers an interesting insight into the reasons for the development of an A-level in computer science. Apparently, the Joint Matriculation Board’s syllabus ‘resulted from the submission to the Board, by a Merseyside

group of teachers, of an Advanced scheme for special approval’³. Perceiving the potential for wider demand the Board co-opted the teachers into developing a syllabus available to all schools.

The first A-level syllabus of which we are aware was examined by means of two written papers each of three hours duration. Paper one, entitled computer science, included, *inter alia*, the topics: history of computing; number systems; fixed and floating point working; problem specification and analysis; computer hardware; and analogue computing. Paper two required candidates to answer questions on two of: numerical methods; statistics; and data processing. This subject matter tends to support the hypothesis that school level computer science was highly consistent with university curricula. For example, one English university launched its first single honours degree in 1971 including the topics: introduction to computer programming; numerical calculus; Boolean algebra and its applications; introduction to computing systems; and introduction to data processing [12].

Interestingly, from the earliest syllabus onwards A-level computer science has required that students engage in practical work. The syllabus described above is specific that for paper one ‘candidates will be required to produce evidence of having written 5 programs of between 20 and 100 instructions in a machine language or suitable mnemonic code . . . to illustrate counting, looping and sub-routines’ and for paper two candidates would be ‘required to submit a selection of 3 programs covering topics included in the sections of paper 2 which they are studying and written in a suitable problem-oriented language. The programs are to be fully documented and include a flow diagram’. The syllabus states that schools ‘will be required to give details of the equipment available at the [school] and/or details of access to external computer facilities’.

Similarly, another board’s report on their 1971 examination observes that ‘the principal limit to the growth of the subject at present appears to be the need for [schools] to obtain approval for the computing facilities which can be offered to candidates for practical work and for the preparation of projects’. In 1981 the same board’s syllabus included the instruction: ‘Heads of schools must not enter candidates for this subject unless prior approval of the arrangements for practical work has been given by the [board]’. Anecdotally, we learn that a common arrangement was for school students’ programs to be sent to a local university or government computing facility. Despite an implied time lag of certainly days, and perhaps weeks, before the output could be returned, this meant that school students’ experience of computer science would be highly consistent with that of university students and computing professionals.

A further running theme in the archive material is the shortage of people with appropriate qualifications and experience. For example, a meeting of computer science examiners at the University of Cambridge Local Examinations Syndicate in April 1982 sought to address, *inter alia*, the ‘shortage of persons with sufficient expertise to undertake all the aspects of examining the subjects’.

³ Unattributed quotations in this section are taken from archive material including minutes of meetings and other documentary sources.

A note in the same archive dated August 1986 observes that ‘There is a severe shortage of people with the necessary experience, ability and willingness to set question papers of high quality’. The note goes on to assert that this situation was replicated at at least one other board. The problem persists: in 2001 one of the boards noted that ‘the shortage of examiners is a problem, 10% of them are lost each year because their marking is below the required standard’.

An interesting detail of the statistics maintained by the Boards is the numbers of boys and girls entering their examinations. Figure 1 illustrates the percentage of females entering the computing A-level, aggregated over various Boards. It will come as no surprise that this percentage always ran at less than 50%, but the sharp drop at or about 1984 demands explanation – this feature is observable in the data from all Boards.

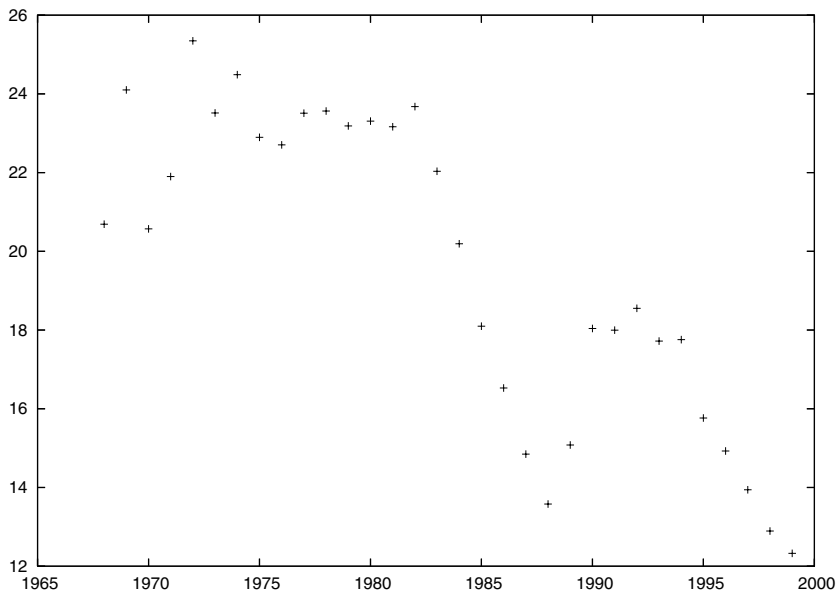


Fig. 1. Percentage of females taking A-level computing, 1968-1999

Coincidentally, the mid-1980s saw the widespread and institutionalised introduction of computers into UK high schools. As we suggest, prior to this date, it is likely that students’ interactions with computers were remote rather than ‘hands-on’. As it happens, the total number of candidates dropped in the mid-eighties, but the effect on gender balance is marked. We are confident (and anecdotal evidence from teachers at the time supports this) that what was intended to be beneficial actually had a negative impact *on computing*; these events marked the dawn of IT.

5 The Dawn of IT Skills

The effects on society of the microprocessor revolution are widely documented and understood; its attendant effects on education are understood but perhaps less widely documented. We contend that its influence on education in England has been profound and in some respects unfortunate.

It is no surprise that the spread of PCs (as they became) in schools in the 1980s led to pressure for qualifications different from the then established A-level. Around 1990 an A-level in IT was introduced that was subject to the same rigorous Board supervision as other qualifications. Its accent was on the development of skills in the use of packages (notably spreadsheets and databases). This fell in line with government policy and the new qualification received enthusiastic support.

It is important to understand that we are not suggesting the IT A-level is of low value: it is, by design, no easier or harder to pass than any others the Boards offer. On the other hand it is not computing: it does not require an understanding of architectures or, critically, computer programming. This means that for school managers the allocation of teaching staff is easier – by its nature IT is easier to teach without a computing degree. Similarly, the recruitment of economically viable cohorts of students is easier – in schools, large classes of IT students are economic sense while small classes studying computer science are not.

Thus while more ‘academic’ subjects are often seen to thrive more easily in the school curriculum than more practical or vocational subjects [13], the reverse has happened in respect of computer science and information technology. Indeed, Board statistics reveal today a very wide popularity for the IT A-level and a steep decline in numbers registering for the parallel A-level in computing.

An unfortunate consequence, in our view, is the emergence of a widespread and difficult misunderstanding about the nature of the discipline(s). Elsewhere [1] we have suggested that the computing A-level can be reasonable preparation for university study (and represents an advantage to students). The incidence of applicants tabling it, however, is diminishing while those with IT qualifications (of which there are several in addition to the A-level) is growing.

This would not matter if there were not a widespread belief that the overlap between IT and computing was considerable (or worse, complete). It is easy to see how this belief arises; for many school pupils there is no exposure to or even discussion of what computing ‘is’. In contrast, for example, even among those who make no post-16 study whatever of, say, physics, there will be a clear understanding that it is ‘hard’, and probably at least a knowledge of words such as ‘quantum’, ‘relativity’, ‘nuclear’ and ‘Einstein’, and no corresponding danger of entry into a career or degree on a misapprehension of what the subject is.

Further, a particularly interesting consequence of the IT qualification is the way in which it offers an outlet for the student who has (laudably) self-taught skills from home use. Anecdotally, we hear consistently of students who have convinced themselves, at least, that their skills outstrip those of their teachers. Sadly, however, such students often find it difficult to adjust to the more disciplined computing practises of an undergraduate programme.

We should further note that the world of ‘IT skills’ has created a rash of qualifications addressing different markets that exacerbate this issue. Within the 16-18 range the vocationally oriented BTEC National Award [14] is an example, and the BCS, professional body for computing, purveys the ‘European Computer Driving Licence’ a skills qualification applicable on a wide (but low-skill) basis [5]. A quote from the promotional literature illustrates that authoritative agencies are openly confusing the issue:

ECDL is designed specifically for those who wish to gain a benchmark qualification in computing to enable them to develop their IT skills and enhance their career prospects.

Thus the status quo is of significant popularity within schools for the study of IT, often under the title or guise of computing, and often in the charge of highly fluent IT-users with little understanding or knowledge of computing. This state of affairs is encouraged by government and its agencies, not to the deliberate detriment of the core discipline but rather in support of wide IT-fluency.

Meanwhile, universities are active but ineffectual in their efforts to correct matters (see, for example, [15]). Particularly in the wake of the precipitous decline in interest in computing degrees since 2001, the unreadiness and misapprehensions of university recruits to computing has been exposed and shows signs of worsening. There is an obvious tension in a world in which there is huge pressure to recruit to keep university departments solvent, but a foreknowledge that the recruits will be unpleasantly surprised by what they are required to learn. Attrition rates in computing have been and remain above the higher education average [16].

6 Developments and Implications

We have argued that the introduction of computer science to the school curriculum offered students an experience that was highly consistent with the professional practice of computing. Further we have suggested that the introduction of computers to high schools coincided with the replacement of computer science with IT skills provision. We turn now to consider potential future developments.

We speculate, for example, that as home PC ownership and the use of computers in the early stages of schooling both increase, the need for high schools to teach IT skills of the type embodied in the ECDL will decline rapidly. On entry to high school students will be increasingly proficient in word processing, the use of spreadsheets and even database management systems, and this will create something of a void in the curriculum.

Indeed, there are signs already of the likely replacements. For example, the examinations board Edexcel have begun to offer ‘a revolutionary new suite of ICT qualifications’ – the diploma in digital applications. A key advantage of this qualification is that it can be ‘taught as a discrete subject or in a cross-curricular context’ [17]. Indeed, the potential for teaching the subject matter without the

need for timetabling dedicated IT classes and, therefore, without specifically IT qualified teaching staff is highlighted by the specification of the ‘Multimedia’ module:

This unit develops students’ ability to design and create effective on-screen multimedia products such as websites, e-books and simulations with a strong emphasis on fitness for purpose.

Content

- *Investigating multimedia products*
- *Designing and developing multimedia products*
- *Prototyping and testing*
- *Presenting multimedia products in an e-portfolio*
- *Project planning, monitoring and evaluation*
- *Creating a multimedia e-portfolio*

We intend no criticism of the new qualification which will not, of course, become embedded successfully in the high school curriculum if it does not serve a useful purpose. We contend, however, that their inherent reliance on manipulating applications means that qualifications such as this make the transition to computer science more difficult than was the case before computers were installed into schools. That is, where their forebears learnt to program computers using the same machines, languages and techniques used in their daily work by computing academics and professionals, today’s high school students learn to become only users of computers.

Further, we suggest that the dramatic decline in applications to study computer science at university can be attributed, at least in part, to such developments. Applications to study computer science at UK universities peaked in 2001 and have declined rapidly thereafter: from a high of over 27,000 applications have fallen each year and were down to 15,357 in 2004 [15]. Although this phenomenon is not well understood and the effects of Y2K and the ‘dot-com bubble’ cannot be discounted entirely, it is increasingly clear that high school students’ appetites for computing are sated by their experience of IT skills courses. For example, one study of high school students in Scotland found that many consider that there is no need to take IT beyond what they experienced in school – indeed, some 55% say that their experience of IT in school has put them off further study [18].

This issue should be of interest outside university computing departments. The UK IT workforce numbers some 1.2 million and, despite off-shoring – which is expected to claim 12% of IT jobs by 2010 – is forecast to grow at between 1.5% and 2.2% per annum for the next decade [19,20]. If these jobs are to be filled by well-qualified graduates there is a pressing need for the distinction between computer science and information technology to be recognised and for the policy agenda, that has been successful in emphasising the need for the populace to gain IT skills, to recognise the importance of computer science.

7 Conclusions

We have described the 16-18 computing education landscape in England. This is a developing landscape in which the number and nature of qualifications is changing, but in which the A-level continues to be regarded as the primary qualification. We have noted the conflicting demands and pressures on the systems delivering these qualifications: on the one hand, government wants an IT-fluent population while on the other, universities seek to develop computing skills in a specialised manner – these simultaneously acceptable aims are being confused in school delivery.

Specifically, there are strands of activity that are computing, and others that are IT. Qualified practitioners can easily distinguish between them (even if they will bicker about terminology), but there is deep confusion evident among the unqualified. This problem is exacerbated by the high proportion of IT and computing teachers in the school system who are not qualified in computing.

We have traced the origins of this issue back to the introduction of desktop computing (now the ‘PC’), and noted a drop at that time in female interest in the topic.

We have noted that the landscape is undergoing change: a primary driver is the wish of government to exhibit an IT-fluent population. We would not detract from this aim, but have also noted that this interferes with preparation for university study of computing – this interference is an issue seen in few if any other disciplines. This has become an acute issue for higher education during years that have seen plummeting student demand, while society’s requirement for qualified computer scientists remains strong.

There is in fact a three-way confusion in evidence. At one end sits the core discipline: usually this is evidence only in universities and will comprise the sometimes esoteric of algorithms, formal methods, and the wide range of subject areas that depend upon them. Some limited aspects of this might appear in schools inside the computing A-level. This contrasts with the requirements of commerce and industry: while employers want more than IT-fluency, they do not (often/always) seek the skills developed by academic departments. At the other extreme lie IT skills; these are certainly, at the expert level, intellectually challenging although the term can also be applied to very elementary skills as well – whichever, they are outwith the academic boundaries of computing.

We conclude that there is a need within the school system to remove the confusion to the benefit of all, and allow people to study topics for well understood reasons. Consequent on this, there is a need to ensure that those doing the teaching are fully versed in the particular area they are purveying; this is an acute problem for computing.

It is a truism to say that computing is new, and still finding its way – in fact we have seen that it has been studied in schools and universities for 40 years, and that argument is fast losing its validity. Where it is ‘special’ is being the victim of a terminological and political confusion that is to the detriment of education. Resolving this would be to the benefit of us all.

Acknowledgements

Some of this work was conducted with grant assistance from the Learning and Teaching Support Network for Information and Computer Sciences, University of Ulster, which we gratefully acknowledge.

We are also most grateful for the significant assistance provided by the archive sections of AQA (Hilary Nicholls), AEB (Jane Bradshaw) and UCLES (Gillian Cooke).

References

1. Clark, M., Boyle, R.: The transition from school to university: would prior study of computing help? In Mittermeir, R., ed.: *Informatics in Secondary Schools: Evolution and Perspectives*, Berlin, Springer-Verlag (2005) 37–45
2. Alexander, S., Amillo, J., Boyle, R., Clark, M., Daniels, M., Laxer, C., Loose, K., Shinnars-Kennedy, D.: Case studies in admission to and early performances in computer science degrees. *SIGCSE Bulletin* **35** (2003) 137–147
3. Boyle, R., Clark, M.: A-level computing: Its content and value. University of Leeds School of Computing Research Report No. 2005.15 (2002)
4. Boyle, R., Carter, J., Clark, M.: What makes them succeed? Entry Progression and Graduation in Computer Science. *Journal of Further and Higher Education* **26** (2002) 3–18
5. ECDL: European Computer Driving Licence. available at <http://www.ecdl.co.uk/> (2004) [accessed 25.08.2006].
6. The Russell Group: The Russell Group. available at: <http://www.russellgroup.ac.uk/index1.html> (2006) [accessed 25.08.2006].
7. Glaser, B., Strauss, A.: *The discovery of grounded theory: strategies for qualitative research*. Aldine de Gruyter, New York (1967)
8. Glaser, B.: *Basics of grounded theory: emergence versus forcing*. Sociology Press, Mill Valley (1992)
9. Glaser, B.: *Theoretical sensitivity: advances in the methodology of grounded theory*. Sociology Press, Mill Valley (1978)
10. Clark, M.: A case study in the acceptance of a new discipline. *Studies in Higher Education* **31** (2006) 133–148
11. Giordano, R.: Institutional change and regeneration: a biography of the Computer Science Department at the University of Manchester. *IEEE Annals of the History of Computing* **15** (1993) 55–62
12. Clark, M.: *Constructing the discipline of computer science: implications for the curriculum*. PhD Thesis, University of Kent (2004)
13. Goodson, I.: *School subjects and curriculum change*. Croom Helm, London (1983)
14. Edexcel: *Qualifications: BTEC nationals*. available at: <http://www.edexcel.org.uk/quals/nat/itc/> (2006) [accessed 25.08.2006].
15. Lovegrove, G.: BCS proposal to HEFCE: Widening participation, preliminary work to increase the supply of IT-students. (2005)
16. Roddam, M.: The determinants of student failure and attrition in first year computing science. available at: <http://www.psy.gla.ac.uk/~steve/localed/roddampsy.pdf> (2002) [accessed 25.08.2006].
17. Edexcel: The DiDA suite. available at: <http://dida.edexcel.org.uk/home/aboutdida/> (2006) [accessed 25.08.2006].

18. Mitchell, A.: Computing science: What do pupils think? available at: http://www.ics.heacademy.ac.uk/student_retention/workshops/pres/Alison%20Mitchell.ppt (2006) [accessed 25.08.2006].
19. British Computer Society: Offshoring: A challenge or opportunity for british IT professionals, Swindon (2004)
20. E-Skills UK, Garnter Consulting: IT insights: Trends and UK skills implications, London (2004)

Teaching Computing in Secondary Schools in a Dynamic World: Challenges and Directions

Bruria Haberman

Computer Science Dept., Holon Institute of Technology, and
Davidson Institute of Science Education, Weizmann Institute of Science,
Rehovot 76100, Israel
bruria.haberman@weizmann.ac.il

Abstract. The field of computing is relatively young compared with other mature sciences, but it has been rapidly developing since its recognition as a stand-alone discipline. The dynamics of the field has led to its inadequate external image and posed challenges in educating newcomers. As a result, educators have been deliberating how to portray the field to others in a compelling way, and how to make computer science studies more appealing to prospective students. One main challenge for educators is to bridge the gap between school and the "real world" of computing. In this paper I discuss two major aspects of the existing gap that relate to (a) the perception of what computing is about, and (b) the educational milieu. I conclude with a description of computing programs, especially designed for high school, which have been in operation in Israel. The aim of the programs is to expose young students to scientific knowledge and the fundamentals of computing, and to motivate them to achieve expertise in this field.

1 Introduction

The field of computing is relatively young compared with other mature sciences (e.g. physics, chemistry, and biology), but it has been rapidly developing since its recognition as a stand-alone discipline [8]. New technologies have emerged since then, and have spread into new application areas [9]. In particular, the rapid development of the field led to the existence of a huge gap between the computing learned at school and the "real world" of computing. Programming languages and tools used to teach computer science have rapidly changed and have become increasingly complex and "the situation has reached a point where it is difficult for individual computer science teachers to keep up" [26, p. 115]. The dynamics and the increasing complexity of computing led to an inadequate external image and pose new challenges in educating newcomers to the field [28]. As a result, educators have been deliberating how to portray the field to others in a simple and compelling way, and how to make computing studies more appealing to prospective students [9,10,14]. For example, the following questions arise: which pedagogic strategies and instructional tools should be applied to reduce the complexity and instability that characterizes the developing computing discipline? [26]. How should the educational system cope with prospective students' preconceptions and expectations? How should

the students be better educated regarding the long-standing fundamental and core principles of the discipline (emphasizing that they are above specific technologies) [14]. On the other hand, how can we prevent them from acquiring a narrow/obsolete view of the contemporary computing?

2 The Gap Between School and the Real-World of Computing

One main challenge for educators is to bridge the gap between the school and "real world" of computing. For example, Denning and McGettrick (2005) stated that "A large communication gap separates us [i.e. computer science educators, B. Haberman] from the masses who want to use computing technology. To cross it we need to learn to speak to them about things they care about" [10, p. 16]. In particular, the gap between learning computing in school and real-world computing relates to the external (public) image of the discipline, and the educational milieu. Both will be discussed in the sequel.

2.1 The External Image

The external image of the discipline seems to be inadequate. The media portrays computing as "stodgy and nerdy compared to other fields" and the public has a narrow view of computing as a field of programmers [10, p. 16].

The school image – Although computer science (CS) is recognized by professionals as a field with scientific and engineering orientation [22], there is still a gap between the status of computer science and the status of natural sciences in the high-school education system. Usually, computer science is considered as a second-class subject compared with natural sciences [14]. In fact, in some countries the status of computer science education in high schools has gradually declined and is no longer considered essential as an independent subject; instead it has been integrated with cross-curricular themes [20, 16]. This phenomenon is the result of the misconception that "computer science is a tool only for other subjects' studies" [14].

Students' conceptions – The current external image is that "computer science equals programming" and it does not stimulate an appreciation of the full depth and breadth of computing. Moreover, it is neither attractive to CS majors nor to prospective students [10]. Indeed, students exhibit misconceptions about computer science that are compatible with the "computer science equals programming" view. This is especially true concerning high-school education and it might also affect students' expectations of further computer science studies in college, namely, emphasis on programming languages [27]. This view is especially strong in most of the computing curricula because introductory courses are about programming and technology, which are interwoven in almost every core course [8].

2.2 The Educational Milieu

The Software Engineering 2004 Curriculum (for the undergraduate level) states that incorporating real-world elements into the curriculum is necessary to enable effective learning of software engineering skills and concepts [1]. To address these goals

requires the organization of a suitable educational milieu. However, in this context, the high school environment has major shortcomings:

Curriculum - There is a gap between the content learned in high school and contemporary computing. The fundamentals and the core technologies that are introduced in school are essential to create a solid basis for further studies of the field; however, they rarely resemble the state-of-the-art computing research and development as well as the emerging directions in the field. One main question relates to the choice of the suitable programming language to teach the fundamentals to novices.

Teacher competence - There is a gap between the desired professional profile of a computer science teacher and the teachers' actual professional education. Recruiting competent computer science teachers at the secondary level might be problematic due to the lack of adequate teacher training programs, and the relatively low pay compared to that of industrial computer scientists [25].

The majority of school teachers are not members of the computing community of practice; usually they lack academic and practical industrial experience, and often they lack practical "hacker-style" knowledge, which is highly appreciated by novices. Consequently, the students might not consider the teachers as representatives of the real-world of computing that they desire to be acquainted with [4].

Style of learning - The traditional teaching and learning in school is usually designed so that students can acquire explicit knowledge based on a thorough understanding of the topic learned. Recently, there has been a consensus among educators that students should be educated to become self-learners who are capable of navigating in the rapidly growing world of knowledge. Hence, students should also be taught to employ a breadth-oriented learning style, according to which the initial exposure to an unfamiliar topic will be accomplished by getting acquainted only with its essence [23].

Software design - The software design and development processes that students experience at school rarely resemble actual research and development industrial processes and their products are rarely applicable to real-world situations. Usually, the students lack the experience of team-work. They develop individual projects according to specifications that are provided by the teacher instead of by a real external client, and in the end they are not acquainted with real-world professional norms. Moreover, the school labs are unable to provide infrastructures characteristic of high-tech industry.

3 Bridging the Gap – Pedagogic Approaches

Here I present pedagogic approaches of computer science educators that relate to various aspects of bridging the gap between learning computing in school and in a real dynamic world.

3.1 Reducing Complexity and Managing Instability

How should the curriculum be organized and how should computer science teachers cope with the increasing complexity and rapid changes characterizing the field?

The eternal question - choosing the preferred first language: Educators continue debating whether it is really necessary to teach up-to-date programming languages (e.g. Java) or whether it is better to use languages that were originally designed to facilitate learning (e.g. Pascal, Logo, and Python). For example, Grandell et al. (2006) recommended using Python (not Java) to teach programming at the high-school level [16]. The findings of their study supported the results of previous studies that students have difficulties in dealing with abstract concepts even when the syntax for implementing them is simple [18]. Those who prefer teaching Java as a first language seek suitable environments that are both simple and stable in order to teach the essential concepts of object-oriented programming. To reduce complexity, the computer science education community must find a way "to separate the essential characteristics of object-oriented programming from those accidental features that are merely artifacts of the particular ways in which technology is implemented today" [26, p. 116]. In particular, it is necessary to develop a stable and effective repertoire of Java-based teaching resources that will meet the needs of the computer science education community [26].

The eternal question – emphasizing fundamentals vs. up-to-date practices: Ben-Ari (2005) stated that the public expectation of CS educators to "bridging the gap" is due to the lack of public legitimacy "to dictate a learning sequence that is not affected by trends and fashions" [6, p. 9]. However, this is not the situation with physics education from the point of view of the legitimacy of the curriculum and the teacher. Physics students in both high school and in their first year at the university spend much time studying "old theories" (i.e. Newtonian mechanics) and solving associated problems. "This material is hardly relevant to the current activities of practicing physicists, but it would be highly unusual for a student, parent, or prospective employer to complain and to demand that introductory students begin by studying relevant subjects like the quantum mechanics of semiconductors or black holes" [6, p. 9]. The legitimacy of organizing the formal school-computing curriculum around (a) fundamental (and lasting) concepts, and (b) problem-solving methods independent of specific computers (and programming languages), along with the practical implementation of those concepts and methods in programming languages [13] and in attractive environments [17] may assist high-school teachers to cope with the increasing complexity and the rapid changes characterizing the field.

Blending formal and informal learning: Instead of trying to "bridge the gap" between learning computing in school and the real dynamic world, the goal can be addressed, for example, by informal out-of-school enrichment programs, in which students encounter representatives of the computing community of practice, and experience real-world software development [30]. This kind of initiative based on blending formal (in-school) and informal (out-of-school) learning can assist in achieving legitimacy for the fundamentals-based school-curriculum and enable the teachers to organize school learning sequences that are not affected by trends and fashions.

3.2 Rebuilding the External Image

The field of computing continues to grow rapidly and new technologies have emerged and spread into new application areas. Even back in 1989, when the number of core

technologies was much smaller, Denning et al (1989) stated that the field cannot be simply viewed as a set of core technologies, and that their goal was "to create a new way of thinking about the field" [8, p. 9]. They suggested a new intellectual framework for the discipline that, besides the core technologies, included three major paradigms- theory, abstraction, and design, and described how the paradigms are used by the core technologies to accomplish their goals.

Later, with the intention of portraying a compelling view of the rapidly developing (and now more complex) field, Denning (2004) suggested that computer science should portray itself in the same manner that the mature sciences portray themselves, namely with a principle-based approach that "promotes understanding from the beginning and shows how the science transcends particular technologies" [9, p. 337]. Specifically, he recommended that computer science, like other sciences, use a set of interwoven stories about the structure and the behavior of the field elements. Since "a list of core technologies does little to convey the great principles of computing" [9, p. 337], Denning suggested a simple framework of *great principles* and computing practices, independent of the number of core technologies. He stated that "a list of core technologies does little to convey the great principles of computing" [9, p. 337]. This framework, which basically is "a new organizing principle for our field" [9, p. 336], provides a stable context for the core technologies of computing, and relates to (1) Mechanics – how computation works; (2) Design – how we organize computation; and (3) Practices – one must be competent when constructing computations. The suggested framework aims at overcoming four main difficulties involved in teaching computer science: (1) Understandability: the framework is not dependent on the growing number of core technologies; (2) Curriculum complexity – the framework aims at resolving newcomers' difficulties such as "trauma of the first language"; (3) Computing practices – the framework offers a new balance between concepts and practice; and (4) Public image – the framework aims at dispelling misconceptions such as "computing is still perceived as a programmer's field" [9, p. 336]. Actually, Denning's suggested framework of Great Principles in Computing Curricula may be employed to cope with the complexity and instability challenges that were discussed by Roberts (2004) in his search for simplicity and stability in computer science education [26].

Recently, Denning and McGettrick (2005), and Denning et al (2006) stimulated a debate about "how to reverse, once and for all, the CS=programming myth" [10,11]. Since most of the curricula use programming as an organizing theme, it is important to seek alternatives for organizing themes, such as innovation, experimental science, and cross-discipline relationships [11]. This approach is compatible with Stevenson's earlier (1993) recommendation that computer science should not be taught isolated from mathematics and other sciences, since students who major in computer science need to acquire a scientific-engineering interdisciplinary approach to solve complex problems in various domains [29].

3.3 Application of Learning Theories to Computer Science Education

How should computer science teachers be best prepared to cope with the increasing complexity and rapid change? How they will be able to portray the notion of the field to their students in a compelling way, and assure that the students acquire viable

models of the learned concepts? Adequate education of prospective teachers as well as appropriate in-service teacher training may reduce this problem [14,21,28].

Most computer science educators lack formal training in education and are unfamiliar with educational theories; hence, they rarely attempt to apply these theories to computer science education [5,14].

Study findings indicated that novices possess an inadequate knowledge of programming and alternative (mis)conceptions of basic computing concepts because of a lack of viable mental models of the computer [12]. The common difficulties experienced by novices could be explained by too much and/or too early use of the computer. Ben-Ari (2001) analyzed the constructivism theory of learning in the context of computer science education, and argued that (1) beginning computer science students do not have an effective model of the computer, and (2) the computer forms an accessible ontological reality. He concluded that the central implication of the constructivist learning theory in the context of computer science education is that models must be explicitly taught before abstractions; meaning that programming exercises should be delayed until class discussion has enabled students to construct a good model of the computer. Moreover, from the social constructivist viewpoint, closed labs that facilitate social interaction should be preferable, and the type of the lab problems should encourage reflections and explorations [3].

In considering the central place of the constructivist learning theory and its influence on pedagogy, Ben-Ari (2001) suggests that computer science educators should "study the theory, perform research and analyze their educational proposals in terms of constructivism". Teachers must guide students in the construction of a viable model of the learned concept, so that they will be able to interpret new situations in terms of the model and accordingly, formulate correct responses. Moreover, it is recommended that software and languages be developed according to constructivist principles to provide learners with the individual construction of viable models [3].

Recently, "many specialists in education have come to the conclusion that cognitive approaches - which investigate the mental processes of the individual learner- need to be supplanted, or at least supplemented, by social approaches- which investigate the effect on learners of the social interactions with the classroom and elsewhere" [5].

Regarding bridging the gap between learning in school and real-world situations Ben-Ari (2004) concluded that most likely decontextualized schooling will continue to be a fundamental method of computer science education [5], since, in particular, in this high-technology world, "a newcomer must have a significant amount of basic and background knowledge before entering into meaningful participation in technological communities of practice" [4]. However, some principles governing a learning situation could be utilized to benefit pedagogy, especially when teaching beginning computer science students. Computer science educators should "devote time to analyzing what actually happens in real communities of practice, and then to create learning activities that simulate such tasks as well as possible within the constraints of a school." Learning activities should be "influenced by the nature of the activities that occur in the community of practice that students will encounter in the future" rather than by its actual detailed practices [4]; moreover, they should motivate students "to be willing to grant the teacher legitimacy as a representative of the community of practice to which they aspire" [5]. Students should be given tasks that could deepen

their sense of meaningful participation in the community, such as working with given complex programs instead of just writing "toy programs" [5]. For example, students should receive experience early in examining, finding, and correcting flaws, as well as in filling in missing parts of given large artifacts, "with the intent of showing them the complexity of the field they are about to study" [7].

Another learning theory that can be applied to computer science education is the *Situating Constructionism* learning theory (suggested by Papert and Harel (1991)) [24]. Its application may enhance meaningful learning of computer science principles and trigger innovative thinking (note that innovation was recommended by Denning & Mcgettrick (2005) as a possible organizing theme of the CS curriculum) [10]. According to this theory, meaningful learning-by-making occurs "in a context where the learner is consciously engaged in constructing a public entity". For example, project-based learning and software development assignments, performed by students in meaningful contexts while applying *Systematic Inventive Thinking* methods [19], may facilitate meaningful learning as well as contribute to making computing more appealing.

4 Teaching Computing in Israeli High Schools

For the last two decades, two programs, one in computer science [13] and one in software engineering [2], especially designed for the high-school level, have been in operation in Israel. The aim of both programs is to expose young students to the fundamentals of computing, and to motivate them to continue their academic studies in that field. Both programs are elective. Recently, a new program for talented students in *Computational Science* has been initiated (<http://www.hemda.org.il/English/>).

The *computer science* program emphasizes the foundations of algorithmic, introduces concepts and problem-solving methods independently of specific computers and programming languages, along with the practical implementation of those concepts and methods in programming languages. The 5-unit (450 hours) program is modular and includes two mandatory modules: Fundamentals of Computer Science (2-units; 180 hours) and Software Design (1-unit; 90 hours), and 2 elective modules: Second Paradigm/Applications (1-unit; 90 hours) and Theory (1-unit; 90 hours). The underlying principles and pedagogical framework of the program are presented in [13]; the curriculum and course syllabi are presented in [15]. The advanced study unit of the curriculum, Software Design, is actually an introductory course in software engineering. It aims at taking the students beyond stand-alone algorithms, and introduces them to various aspects of software systems design. One important goal of the unit is to demonstrate full integration of the conceptual material and the actual hands-on experience in designing and constructing a real system. After studying this unit the students are better prepared to proceed to a higher level of system design and development.

The *software engineering* program aims at exposing the students to a fundamental scientific domain whose principles are characteristic of algorithmic thinking as well as system-level perception. The program has evolved over the years in accordance with the changes in the discipline of computing. It introduces students to scientific

methods, principles of design, and implementation of computer systems. The program consists of the following components: (a) an elective topic in natural sciences, (b) computer science (presented above), and (c) an elective advanced specialized computer science topic. The specialization phase is called: Design & Programming (D&P) of Software Systems since it is designed to teach, in addition to theoretical principles, also design methods and implementation tools that are suitable for the specific advanced topic. Currently the program suggests 6 alternative topics: Information Management Systems, Computer Graphics, Operating Systems, Expert Systems, Web Services, and Network Systems. Students study the advanced topic for 450 hours during the two last years of high school. Theoretical principles and practical experimental issues of the topic are introduced and practiced in the laboratory. The studies include the learning of a programming language/environment that is suitable for implementing the theoretical material of the advanced topic. During the third year, students are required to develop as a final assignment a comprehensive software project.

The computational science program (MOAH) is designed to provide students with very broad-based, in-depth scientific knowledge along with the foundations and skills needed for using the computer as a scientific tool. Outstanding high-school students who study 5 units of mathematics and either physics or chemistry can participate in the program. The students are engaged in solving complex problems and in the computerized control of experiments, creating models and simulations of natural phenomena in physics, chemistry, biology, communication, economics, and other fields. During the third year, students are required to develop, as a final assignment, a comprehensive simulation software project (<http://www.hemda.org.il/English/>). In contrast to the software engineering program, which is organized around the principles and methods of software development as the main theme, the computational science program emphasizes the use of the computer as a scientific tool.

5 Concluding Remarks

In this paper I discussed the challenges of teaching computing in secondary schools in a dynamic world. I presented experts' recommendations related to portraying computing to newcomers in a compelling and appealing way, to ensure that students acquire viable models of the learned concepts.

Especially when the goal of "bridging the gap" between fundamentals and the dynamic world of computing has become so challenging, I would like to stimulate "second thoughts" on this issue, and to conclude with a citation from Ben-Ari's keynote address (the winner of *SIGCSE's Award for Contributions to Computer Science Education*): "I believe that CS education must fundamentally change in order to equip the student with a firm, deep and broad theoretical background, long before specialization is undertaken. We are grown up now and with growing up comes the responsibility to build a mature system of education... I hope that ten or twenty years from now, the recipient of his award will be able to report on a convergence of the educational practices in computer science with those in other scientific and engineering disciplines" [6, p. 9].

References

1. ACM/IEEE Joint Task Force on Computing Curricula, Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, A Volume of the Computing Curricula Series, August, 2004.
2. The Ministry of Education, Israel, A high-school Software Engineering program, 2004. Retrieved June 11, 2006, from <http://csit.org.il> (in Hebrew).
3. Ben-Ari, M. (2001). Constructivism in computer science education, *Journal of Computers in Mathematics and Science Teaching*, 20(1), 45-73.
4. Ben-Ari, M. (2003). Situated Learning in This High-Technology World, *the 7th International History, Philosophy and Science Teaching Conference*, Winnipeg, Canada, 2003. Journal edition published in *Science & Education*, 14(3-4), 2005, 367-376.
5. Ben-Ari, M. (2004). Situated Learning in Computer Science Education, *Computer Science Education*, 14(2), 85-100.
6. Ben-Ari, M. (2005). The Concorde Doesn't Fly Anymore, Keynote Talk, *SIGCSE'05*, St. Louis, MO, Retrieved March 28, 2006, from <http://stwww.weizmann.ac.il/G-CS/BENARI/files/concorde.pdf>
7. Bergin, Fourteen pedagogical patterns for teaching computer science, Retrieved March 25, 2006, from <http://csis.pace.edu/~bergin/PedPat1.3.html>.
8. Denning, P. J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A., Turner, A.J. & Young, P.R. (1989). Computing as a discipline, *Communication of the ACM*, 32(1), 9-23.
9. Denning, P. J. (2004). Great principles in computing curricula. *Proceedings of SIGCSE'04*, Norfolk, Virginia, USA, 336-341.
10. Denning, P. J. & McGettrick, A. (2005). Recentering computer science, *Communication of the ACM*, 48(11), 15-19.
11. Denning, P. J., McGettrick, A., Rosenbloom, P. & Snyder, L. (2006). Re-centering computer science, *Proceedings of SIGCSE'06*, Norfolk, Virginia, USA, (Special Session).
12. du Boulay, B., O'Shea, T., & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway & J.C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 431-446). Hillsdale, NJ: Lawrence Erlbaum Associates.
13. Gal-Ezer, J., Beerli, C., Harel, D., & Yehudai, A. (1995). A high-school program in computer science. *Computer*, 28(10), 73-80.
14. Gal-Ezer, J. & Harel, D. (1998). What (else) should CS educators know? *Communications of the ACM*, 41(9), 77-84.
15. Gal-Ezer, J. & Harel, D. (1999). Curriculum and course syllabi for high school CS program. *Computer Science Education*, 9(2), 114-147.
16. Grandell, L., Peltomaki, M., Back, R.J. & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. *Proceedings of ACE'06*, Hobart, Tasmania, Australia. Retrieved March 25, 2006, from <http://crpit.com/confpapers/CRPITV52Grandell.pdf>
17. Guzdial, M. and Soloway, E. (2003). Computer science is more important than calculus: The challenge of living up to our potential, inroads – *SIGCSE Bulletin*, 35(2), 5-8.
18. Haberman, B. & Ben-David Kollikant, Y. (2001). Activating “black boxes” instead of opening “zippers” – A method of teaching novices basic CS concepts. inroads – *SIGCSE Bulletin*, 33(3), 41-44.

19. Helfman, J. & Eylon, B. (2003). Systematic Inventive Thinking, Department of Science Teaching, Weizmann Institute of Science, Rehovot, Israel, 250 p (in Hebrew).
20. Kavander, T. & Salakoski, T. (2004). Where have all the flowers gone? – Computer science education in General upper secondary schools, *Proceedings of the Kolin Kolistelut-Koli Calling Conference*, Koli, Finland, 112-115.
21. Lapidot, T. & Hazzan, O. (2003). Methods of Teaching Computer Science course for prospective teachers, inroads – *SIGCSE Bulletin*, 35(4), 29-34.
22. Lee, P. (2004). *The computer science brain drain: A call to revitalize computer science education*. Retrieved April 23, 2004, from <https://www.wiki.cs.cmu.edu/public/uploads/Main/it-talent.pdf>.
23. Long, P.D., & Ehrmann, S.C., Future of the learning space: Breaking out of the box, *Educause*, 2005, 42-58.
24. Papert, S. & Harel, I. (1991). *Costructionism*, Ablex Publishing Corporation.
25. Poirot, J.L., Taylor, H.G. & Norris, C.A. (1988). Retraining teachers to teach high school computer science, *Communication of the ACM*, 912-917.
26. Roberts, E. (2004). The dream of a common language: the search for simplicity and stability in computer science education, *Proceedings of SIGCSE'04*, Norfolk, Virginia, USA, 115-119.
27. Schollmeyer, M. (1996). Computer programming in high school vs college, *Proceedings of SIGCSE'96*, Philadelphia, PA, USA, 378-382.
28. 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, February 2005. Retrieved June 11, 2006, from <http://csta.acm.org/Publications/CSTA-WhitePaperNC.pdf>
29. Stevenson, D.E. (1993). Science, computational science and computer science: At a crossroads, *Proceedings of the 1993 ACM conference on Computer Science*, Indianapolis, Indiana, USA, 7-14.
30. Yehezkel, C. & Haberman, B. (2006). Bridging the gap between school computing and the "real world", *ISSEP'06: Informatics Education – The Bridge Between Using and Understanding Computer*, Vilnius, Lithuania, 39-49

Functions, Objects and States: Teaching Informatics in Secondary Schools

Peter Hubwieser

Fakultät für Informatik der Technischen Universität München
Boltzmannstr. 3, 85748 Garching, Germany
Peter.Hubwieser@in.tum.de
<http://ddi.in.tum.de>

Abstract. Recently we developed a new part of our educational concept for secondary schools. The concept was in total designed for the grades 6,7 and 9-11 of the Bavarian Gymnasiums, where we introduced a new mandatory subject informatics that started in 2003 at about 400 schools. After giving a short summary of the educational context and the other parts of the concept, we describe in detail the recently developed course of lessons that combines object oriented modelling (including a close look at the states of objects and attributes) with object oriented programming.

1 Introduction

As pointed out in a variety of publications (see e.g. [2]), there are many apparent arguments for a early systematic school education in informatics, for example:

- The attitude towards computers and information technology in the childhood is deeply influenced by many factors like gender, social position, ethnic origin or the attitude of the parents. The public schools have to compensate these differences as early as possible in order to give every child a fair chance to master the challenges of the information society.
- The students intensively make use of information technologies at home as well as at school. We have to prevent them from building wrong “self-made” mental models of these technologies or from using an inappropriate vocabulary to describe them. Thus at least the important aspects of IT-Systems should be properly introduced by teachers which have a sufficient theoretical background in informatics.

Following Piaget [9], the students reach at the age of 11 (corresponding to the 6th grade in the German school system) the stage of *formal operations*, which is a necessary precondition to learn informatical concepts on a sufficient abstract level. Thus in our opinion this is the perfect point of time to start a mandatory subject of informatics, which should go far beyond the pure drill of user skills [2], [4]. As already described in other publications [5], [8], we developed a new educational concept (based on modelling) and finally succeeded in introducing a new subject “informatics” at all Gymnasiums (the type of secondary schools that directly leads to

the University) of the state of Bavaria. As the reader might know, in Germany every one of the 16 states has its own school system.

In this paper I will focus on the (still unpublished) part of the educational concept for the 10th grade, that we recently developed, forced by the shortening of the education duration at Gymnasiums from 9 to 8 years. Our new subject now takes place in the grades 6, 7 and 9–12.

2 Focussing on Object Orientation

There are many courses of lessons in informatics that are designed for elder students at the end of secondary education or already at universities. In contrast to these concepts we created a course that starts at the lower end of secondary education in grade 6. Thus we had to be very careful concerning the level of abstraction in the first years of the course. On the other hand it is a characteristic quality of our gymnasiums that the students should (really) *understand* every concept they are applying (we have other types of secondary schools that concentrate on applying concepts in a more black-box way).

From the point of view of general education it seems that among all themes of informatics it is object oriented modelling that promises the most benefit for the students. Thus we chose it as the central theme of our course.

Preliminary considerations and experiences showed that we had to differentiate neatly between three *fields*:

1. *Object oriented modelling (OOM) of hard- or software components*, which is comparatively unproblematic because it focuses on concrete, already structured objects like documents, data structures or computer components. Following our experiences it is possible to start teaching in this field without any preliminary informatic education.
2. *Object oriented modelling of “real world” systems* beyond simple technical components, which frequently leads to very demanding concepts like inheritance, concurrency or might require deep philosophical deliberations. Exemplary systems are car rental companies, flight booking systems, banking systems, social structures or biological taxonomies.
3. *Object oriented programming (OOP)*, which requires profound insights into specific implementation concepts like references or rather pointers, polymorphism or access modifiers. Additionally we had to take up the challenge of the choice of a suitable programming environment which should make these concepts transparent without bothering the students with too complicated syntax details. Additionally it should be possible to produce at least some programming language code directly out of the graphically represented models.

In arranging the introduction into these fields over our course, we had to take into consideration that students don't like to produce models without a chance to implement them and that some concepts like functions, basic data structures, data types and states are essential for mastering object oriented programming. We also closely paid attention not to introduce more than one concept at the same time.

The result of our considerations was the following three-stage strategy of education in informatics:

1. In the grades 6 and 7 the emphasis lies on object oriented modelling of *standard software documents* (field 1, see above) like vector graphics, texts or hypertexts, supplemented by some basic concepts of algorithms.
2. In the grades 9 and 10 the students concentrate on modelling *real world problems* (field 2) as well as on implementing and simulating their models on specific software systems, starting with state charts and data base systems and finally using object oriented programming environments (field 3).
3. In the final grades 11 and 12 (within a college type educational system where the majority of the lessons takes place within elective courses) we will offer specific courses of lessons on different intensity levels with specific learning contents like recursive data structures, concepts of software engineering or theoretical informatics.

The concept for the grades 6 and 7 has already been described in detail (see [5]). Thus we will restrict the presentation of the teaching concepts of these grades to a short summary (see section 3). The total educational structure for the grades 11 and 12 at the gymnasiums is still in a development stage, therefore we are not yet able to construct the teaching concept. Thus this paper will focus to the grades 9 and 10. Recently we have developed a very new concept of introducing object oriented programming using the state concept, which will represent the main topic of this paper.

3 Summary of Grade 6 and 7: Objects and Simple Algorithms

As described in [5] our novice courses start in grade 6 with the elaboration of the basic concepts (“object”, “attribute”, “method” and “class” in the context of vector graphics. In a second step we develop the models by introducing the concept of aggregation (e.g. paragraphs containing characters) in the context of text processing. Subsequently the students encounter the first recursive aggregation, presented by the example of folders that contain other folders. This leads to the construction of folder trees, used to represent hierarchical structured information. The regarded graphs are extended by the concept of links in the context of hypertext systems, allowing the construction of cyclic graphs (see Fig. 6). Finally we start to program our own methods using a virtual roboter (see Fig. 7), which is regarded as the only instance of a class “Roboter”, with some “built-in methods” like “forward”, “turn left” etc.

4 Grade 9: Functions and Data

As already mentioned above, we are strongly convinced that it is absolutely necessary to teach some foundations before entering the fields 2 and 3, which are the following:

- the concept of function which is needed for methods as well as for the functional description of complex systems,
- data types: numeric, character, text and boolean,
- basic data structures: records and fields.

Following these considerations we start the 9th grade with the concept of function. The students construct functional models (data flow diagrams), where the information processing units are restricted to functions and implement them on spreadsheet systems like *MS-Excel* or *StarCalc* (see Fig. 1) which has been described in Detail in [6].

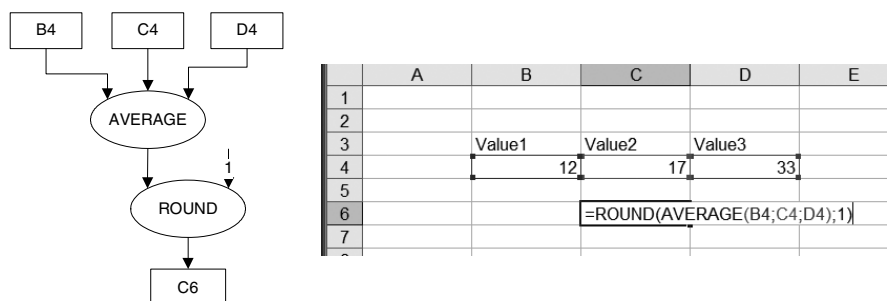


Fig. 1. Functional model implemented in a spreadsheet system

The rest of the 9th grade is dedicated to (object oriented) data modelling. Starting with a system that consists of only one table the students encounter the basic concepts of data base systems: record, table, query, using a common data base system like MS Access or MySQL. Following this introduction they start with entity relationship modelling of more complex systems that consist of several tables, connected by relationships, and implement their models in relational data bases. As we continue to use the UML-like object oriented modelling technique that the students already know from the 6th and 7th grade (instead of the proper ER-technique still used by the DB-community), they perceive the data base structures as object models without methods.

While working with queries students notice that the functional modelling approach helps them to understand the structure of the SELECT statement in SQL, for example

```
SELECT CustNr, Name, FirstName
FROM Customer C, Account A,
WHERE C.CustNr = A.CustNr AND A.Balance > 100.000
```

which is interpreted as the functional composition

$$P_{\text{CustNr, Name, FirstName}}(S_{A.Balance > 100.000}(\text{EquiJoin}_{C.CustNr=A.CustNr}(C, A))),$$

where P, S stand for the relational calculus operations project and select, respectively, see Fig. 2.

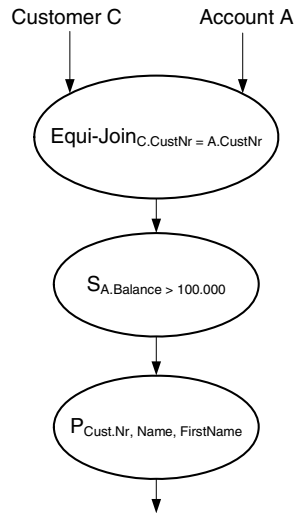


Fig. 2. Functional view of the SELECT statement in SQL

5 Grade 10: Objects and States

In the original conception of our mandatory subject, we would have started the 10th grade with state modelling combined with imperative (non-object-oriented) programming, followed by a second part of functional modelling, dedicated to the introduction of procedures and functions with specific respect to parameter passing mechanisms. The 11th grade was planned to deal with OOM and OOP in great detail.

After loosing grade 8 with the shortening of the Gymnasium, we had to compress the second stage of our educational concept from 3 to 2 years (from grades 8-10 to grades 9-10, respectively). Thus we decided to integrate state modelling into the original OOM/OOP-part, which turned out to improve the whole concept considerably.

5.1 Repetition of Object Oriented Concepts

In order to recall the object concepts they have already learned in grades 6 and 7 (object, attribute, method, class, relationship), the students construct some vector graphics, using a suitable programming framework, as provided by Barnes and Kölling [1] in combination with their BlueJ programming environment, see Fig. 3. Currently we strongly recommend this system because of the following advantages:

- It works with Java-Code, which is a very widespread professionally used language. This causes strong motivational effects to the students.
- It uses class and object diagrams that are very similar to the notation we use in our course.
- Objects are graphically displayed, so that the students are continuously informed about the active objects.

- Methods are called and parameter values are passed interactively. The students call the method nearly the same way they are used to do method invocation while using standard software systems (e.g. calling the method “flip horizontally” in a vector graphic software like Corel Draw, see [5]).
- There is no need to consider the (hidden) main method, because the students work directly on the objects. This spares them from dealing with the cryptic „public static void main..” syntax of original Java.
- The states of the objects can be observed at any point by the object inspector. This leads directly to the state concept of attributes.

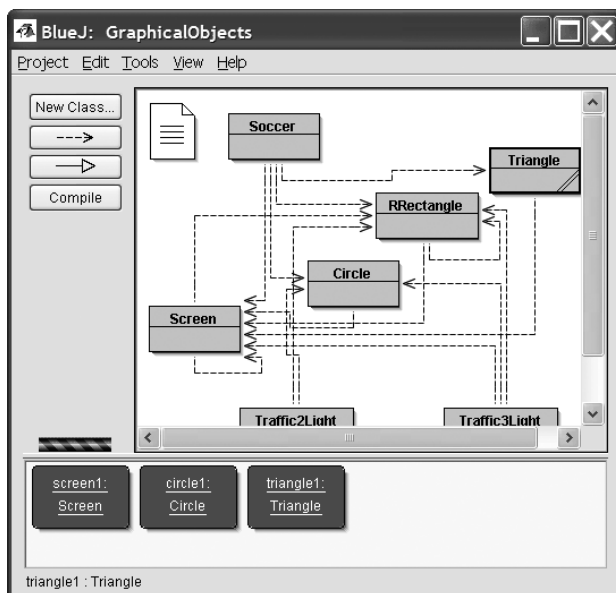


Fig. 3. Graphical objects in the BlueJ Development Environment

As already mentioned above, the BlueJ environment enables students to create and manipulate the objects by invoking their methods (e.g. `setColour`, `setCenter`, `moveUp`) directly. The graphical structures will be constructed therefore by creating objects, modifying and arranging them by invoking their methods interactively.

5.2 Classes in Programming Languages

After the repetition step we transfer the class concept (including the concepts of attributes and methods), which is well known to the students from the grades 6, 7, and 9, into a programming language, e.g. Java, by comparing a class card (e.g. of the class `Circle`) with the definition of the same class in the program text, see Fig. 4. This is motivated by the desire of automation.

The students might remember at this point their experience with programming virtual robot systems in grade 7 (see [5]), which might be regarded as programming new methods of an object `robot1` of the Class `ROBOT`.

CIRCLE	public class Circle
diameter	public int diameter
centerX	public int centerX
centerY	public int centerY
...	...
moveUp()	public void moveUp() {
...	... }

Fig. 4. Comparison of class card and program class

On this occasion we have to introduce some new concepts like the return type of functions, especially `void`, the constructor concept and the access modifiers `public` and `private`.

5.3 States and the Assignment Command

One of the greatest challenges of the informatics education is the proper introduction of the assignment command (e.g. “assign the value 5 to the variable number”), particularly its distinction from the equality statement (e.g. “the value of number is equal to 5”). In the syntax world of C and its successors (e.g. Java), this is made even more difficult by the unlucky choice of the “=” sign for the assignment, which is well known by the students from Mathematics, symbolizing equality there (This nonsense was topped by the choice of “==” for the equality).

Thus in our opinion it is not advisable to introduce the assignment by using the “=” sign without any further support of proper perception. In our concept this is provided by the restriction to use the assignment only within “set” methods of the attributes, which should help to prevent the students from confusing assignment and equality:

```
public void setColour(int newcolour) {  
    colour = newcolour ;}
```

As this is just a part of the encapsulation concept, it is easily motivated towards the students by the usual arguments of data security and cooperative software development.

The semantics of the assignment command will be explored by closely looking at the values of the attributes during sequences of assignments. Proper understanding might be tested by the task of exchanging the values of two attributes. This leads directly to the state concept of attributes (respectively of variables) which is essential to understand an assignment like `counter=counter+1`, in which the attribute (variable) `counter` appears in two different states, represented by its values before and after the execution of the assignment.

5.4 States of Objects

As we have now introduced program objects, we are in the position to bridge the gap between the already well known principles of OOM and the implementation of object

structures by OOP. As the execution of an algorithm that is implemented in an object oriented program is realized by a sequence of object states (represented by the values of the attributes), the state concept is essential to understand this. We start with considering the states of program objects, immediately followed by the states of real world objects.

We start by defining the state of an object by the combination of the states of all its attributes, each of them represented by the actual value of the attribute (corresponding to a single point in the n -dimensional state space of the object that is defined by the cross product of the state spaces of its n attributes). This means that an object is changing its state if one of its attributes changes its value. The transition between two states is triggered by a method that changes the values of an attribute. It is simple to calculate the number of possible states of an object as the product of the number of possible values over all its attributes. Apparently this might be a very big number, for example: An object with two attributes of the type `int` in Java has $2^{64} \approx 10^{19}$ states.

In order to describe the states of an object we are introducing state charts, see Fig. 5, showing the change of the colour of an object `circle1` of the class `Circle`.

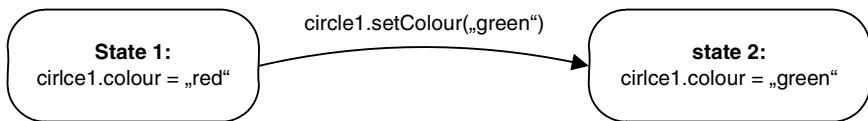


Fig. 5. An attribute changing its value, displayed in a state chart

Now we turn our attention towards the states of real world objects and model them in further state charts, for example the chart of German traffic lights, see Fig. 6.

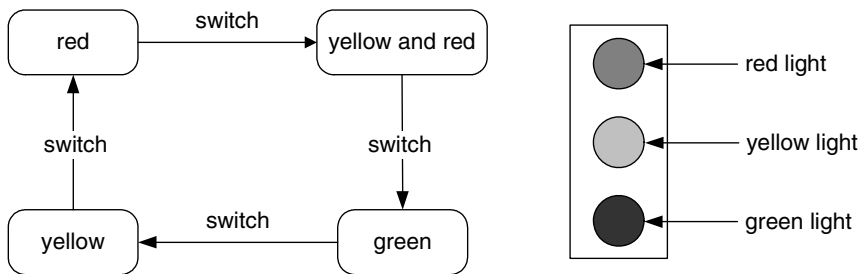


Fig. 6. State chart of German traffic lights with three colored lights and four states

For better understanding we have to mention that German traffic lights use a combination of their red and their yellow light as a forth state on their way from red (“stop”) to green (“go”). One notices that the traffic light system uses 3 indicators (red, yellow and green light, each with the states “shining” and “dark”) to display 4 different system states, which the road users interpret usually as: “stop” (red), “prepare for stopping if not very close” (yellow), “go” (green), “prepare for forthcoming go” (yellow-green). We discover that the real technical system “traffic

light” uses only 4 of its 8 technically possible states (defined by all possible combinations of the two states of each of the 3 lamps). For example the state “shining-dark-shining” (in colours red-green) is technically possible, but not used.

At this point we have to explain the correspondence between the states of *real world objects* (e.g. the real traffic light system) and *program objects* (e.g. the implementation of a traffic light system within a Java-Program by a combination of three circles) which are simulating the real-world objects.

In order to simplify things, we illustrate this using a pedestrian light with *two* lights instead of the usual three-lamp traffic light (mostly used at car lanes). Thus its implementation class `TrafficLights` contains two objects `circle1`, `circle2` of the class `Circle` in order to simulate the two lights. Each of these objects has to switch between two different colours: `circle1` between red and black, `circle2` between green and black, see Fig. 7 and Fig. 9.

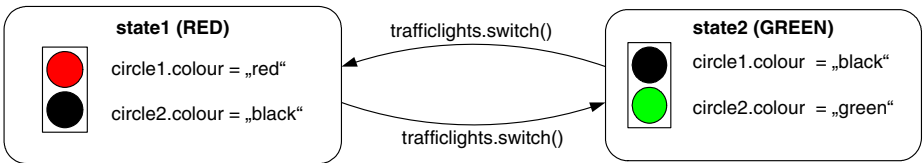


Fig. 7. States of program and real world objects, referring to a two-color pedestrian light

A closer view on the relation between the states of the real world object and the program object shows that the program object usually has much more states but uses only some of them to simulate the real word object.

If we restrict our state consideration on the colours of the circles (as objects of the program class `Circle`), then in our example the filling colour of each of the two circles might have 16 possible values (e. g. red, black, blue, yellow, green, brown, ..) and thus the objects of the class `TrafficLights` have 256 possible states, while they only use 2 of them to simulate the real traffic lights, see Fig. 8.

circle1.colour→ circle2.colour↓	red	black	blue	yellow	green	brown	...
red							
black	state1:RED						
blue							
yellow							
green		state2:GREEN					
brown							
...							

Fig. 8. The states of a simple program object

By this way the students should get a rough idea of the mathematical concept of state space of an object (as the Cartesian product of the state spaces of its attributes).

At this point the students should be able to transfer this insight to the traffic light system with three lights (see Fig. 6). Its implementation class use three circles with 16 colours each and thus its objects might take one of 4096 different states, while the simulation of the real system only covers 4 of these. As a consequence of these considerations, from now on every object is perceived as a state machine.

5.5 Referencing Objects

One of the most challenging tasks in teaching informatics is the introduction of pointers or references, which is essential for the understanding of certain effects that arise for example if two references point towards the same object. Fortunately our students already know the aggregation as a special type of association (marked with “contains”), for example: an object of the class *Folder* may contain an object of the class *File*. This leads us exactly to the right concept: if an object `obj1` of `Class1` contains an object `obj2` of `Class2`, the latter will be implemented as an attribute of the type `Class2` in `Class1`, whose value is actually a reference towards the external object `obj2`, see Fig. 9.

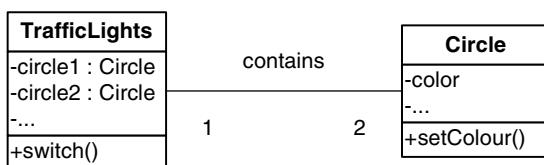


Fig. 9. Aggregation between the objects of the classes *TrafficLights* and *Circle*

It is crucial for the proper perception that the students keep in mind that the aggregation association is always connecting objects, although it is drawn in the class diagram.

5.6 Algorithms

Until this point the behaviour of the objects was explored by interactively invoking their methods, thereby causing single state transitions. Now we set us the goal to make the state machine run automatically, for instance by programming the methods `start`, `stop` and `switchState`, see Fig. 10.

On the way to achieve this goal, the students have to recall the structural elements of algorithms that they have learned already in grade 7 while working with the robot

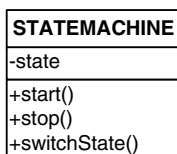


Fig. 10. Class card of the class “Statemachine”

systems: elementary, sequential, conditional and repetitive processing structures. Put together, we now have all the tools to construct an automatic state machine. The resulting program structure is shown in Fig. 11, written in pseudocode.

5.7 General State Modelling

After enabling the students to implement simple state machines, we expand the transitions in our state models by *triggered* actions and *conditional statements* (as precondition for the transition), following the syntax of UML state charts. With that the students are able to model and implement more complex state systems. Concerning the choice of the examples, we set the emphasis on the relevance for the everyday life, which leads for instance to banking or mobile phone processes or dispenser machines for a variety of products (see Fig. 12).

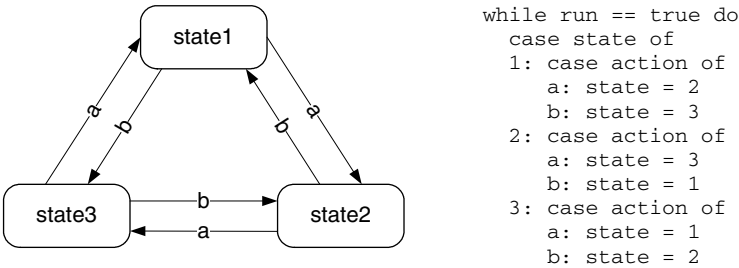


Fig. 11. Schematic implementation of a simple state machine

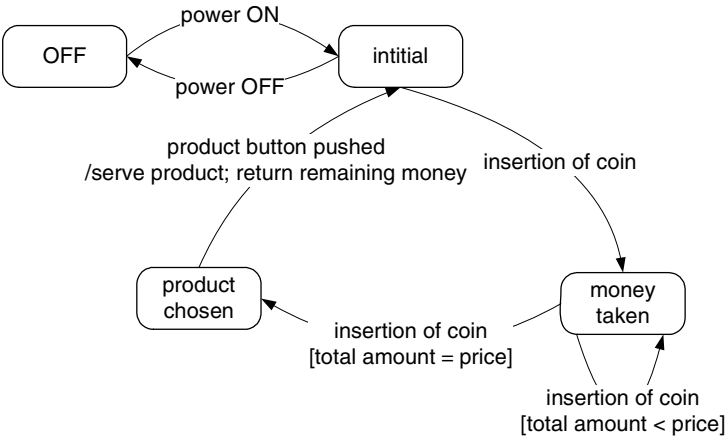


Fig. 12. Simplified state chart of a dispenser machine

5.8 Inheritance

In order to model systems that are more complex, we introduce inheritance at a suitable point of time. It is very important to make clear that this is a class-to-class association in contrary to the aggregation. From a didactical point of view it is also

advisable to stress the abstraction aspect (from subclasses to their superclass) of inheritance (in contrary to the pure technical “inherits attributes” view of many textbooks).

6 Teacher Education

In order to secure that the course will be taught by properly educated teachers, we started in 1995 with the first experimental in-service teacher qualifying programme at the Technische Universität München. These experiments led to a two year training course that contains modules about data base systems, imperative programming and state modelling, functional and object oriented modelling and programming, technical and theoretical informatics. In the first year the participants work predominantly at home, supported by e-learning media and by an experienced tutor, while in the second year they have to attend one day per week presence lessons at the University. Meanwhile four other universities have joined this programme that will end in September 2006 and was attended by about 300 teachers in total.

Currently we are working on a modified continuation programme that sets even more emphasis on self studies and restricts the presence part strictly to the module about theoretical informatics.

7 Concluding Remarks

While the education concept for the grades 6 and 7 is already running in a very large scale (at about 400 gymnasiums with a total of about 1200 classes and about 30.000 students currently) and thus heavily explored, the continuing concept will be rolled out finally at September 2006. Currently it is tested in parts at about 20 gymnasiums that had to test the shortened education duration of 8 years. We will evaluate the outcomes within the next year and publish the results.

In order to provide learning materials to students and teachers, we are producing schoolbooks with the Klett Verlag, Stuttgart. The first volume for the 6 and 7 grade was published in 2004 [3] while the second volume for grade 9 will be available in Summer 2007 [7].

Our next project will be the concept for the two final grades 11 and 12 of the gymnasium. The education structure for these grades is not yet clear, but it seems that there will be some mandatory subjects (German language, mathematics, English and history) as well as elective courses in different fields like natural or social sciences and languages. Additionally there will be two “seminary subjects” with dedication to project work and scientific writing. Informatics will be included as an elective subject on the one hand and as a strong focus of the seminary subject on the other hand.

References

1. Barnes D. J., Kölling M.: Object First with Java. A Practical Introduction using BlueJ. Pearson Education Limited, Harlow, UK, 2003.
2. Breier N., Hubwieser P.: An Information-Oriented Approach to Informatical Education. Informatics in Education 1(2002), pp 31-42.

3. Frey E., Hubwieser P., Winhard F.: Informatik 1. Objekte, Strukturen, Algorithmen. Klett, Stuttgart, 2004.
4. Hubwieser P., Broy M.: Educating Surfers or Craftsmen: Introducing an ICT Curriculum for the 21st century. In: Downes T., Watson D.: Communications and Networking in Education: Learning in a Networked Society. IFIP WG 3.1 and 3.5 Open Conference. Aulanko-Hämeenlinna, Finland, June 13-18. Yliopisto, Helsinki 1999. pp 162-170.
5. Hubwieser P.: Object Models of IT-Systems Supporting Cognitive Structures in Novice Courses of Informatics. In van Weert T., Munro R. (Eds.): Informatics and The Digital Society: Social, Ethical and Cognitive Issues, IFIP TC3/WG3.1&3.2 Open Conference on Social, Ethical and Cognitive Issues on Informatics and ICT, July 22-26, 2002, Dortmund, Germany. IFIP Conference Proceedings 244 Kluwer 2003, pp 129-140
6. Hubwieser P.: Functional Modelling in Secondary Schools Using Spreadsheets. *Education and Information Technologies* 9 (2): 175-183, June 2004
7. Hubwieser P., Schneider M., Spohrer M., Voß S.: Informatik 2. Klett, Stuttgart, to be published 2007.
8. Müller P., Hubwieser P.: Informatics as a Mandatory Subject at Secondary Schools in Bavaria. In: Open Classrooms in the Digital Age - Cyberschools, e-learning and the scope of (r)evolution. Proceedings of the 2000 Open Classroom Conference - Barcelona, Spain. EDEN, Budapest, 2000. pp 13-17. (PDF, 24kB)
9. Piaget, J.: The stages of the intellectual development of the child. *Bulletin of the Menninger School of Psychiatry*, March 6, 1961.

From Intuition to Programme

Michael Weigend

Holzcamp-Gesamtschule Witten, Willy-Brandt-Str. 2,
58452 Witten, Germany
michael.weigend@fernuni-hagen.de

Abstract. In computer science classes you can observe that students are able to solve a problem – say sorting a list – but fail completely writing a programme that does the job. This barrier between intuitive solution and the finding and explication of an algorithm can be overcome, if the programmer learns to analyse her or his own intuitions connected to the task.

1 Intuition

Whenever we develop a programme, talk about a programme or try to understand how a programme works, we use intuitions. According to Fischbein [6] an intuition is a mental concept, which is self evident, holistic, persistent and certain. A typical example in the field of informatics is the container model. Using this intuition an assignment statement like

$$x = 1$$

is interpreted in the following way: There is a box labelled with x , in which someone puts a piece of paper with the number 1 written on it. We are very familiar with containers and therefore use in connection with programme phrases like "the content of variable x is the number 1". Of course the container model – like any analogy – has its limitations. So one has to be aware that – in contrast to a usual container in every day life – the former content of a container representing a variable will be deleted in the moment it is filled with some new object.

In cognitive psychology often a distinction is made between declarative and procedural knowledge (see: [1]). Intuitive models like the container analogy are declarative knowledge about the world ("knowing that"). But there is also intuitive procedural knowledge ("knowing how") like riding a bike or sorting a deck of cards. A procedural intuition is a task a person can effectively solve but cannot necessarily tell how she/he made it. That does not imply that the subject doesn't use an algorithm. But eventually the algorithm is not conscious or the person is not able to represent it (verbally or visually) and therefore cannot explicate and communicate it. Still there is the subjective certainty (a characteristic of intuition) to be able to solve the task.

Leiser [8] observed that procedural knowledge can coexist independently with declarative knowledge about the same domain. He asked people what would make the "ideal couple". The interviewed persons could tell criteria for stable relationships. But when they were asked to pick concrete persons from their circles of acquaintances in order to form "ideal couples" they deviated from these principles.

2 Barriers

Students who are to write a programme often have procedural intuitions how to solve the task. Every student is able to find a book in a library, to sort a deck of playing cards and to search a way from A to B on a street map. Bell, Witten and Fellows [3] have designed classroom activities for elementary school children to teach demanding computer science topics (such as text compression, searching algorithms or routing in networks) "unplugged," that is, without any computer. The ability of even young students to play these games proves the existence of much intuitive knowledge within the domain of informatics.

Nevertheless it is widely observed, that novices fail writing a programme, even when they already know an intuitive solution. There seems to be a gap between the intuition and the corresponding programme code. Out of the many obstacles a student has to cope with during programming, this article focuses on four barriers, which are connected with the use of intuitive models: The problem of breaking up the Gestalt of a holistic concept, the fixation on non-implementable intuitions, missing connections between intuitive models and programming principles and finally misconceptions.

2.1 Breaking Up the Gestalt

Each programming project starts with the subjective certitude that it will be a success. (Otherwise nobody would begin this demanding and time-consuming process.) In many cases a programmer has a vision, a holistic intuitive model of the programme she or he is going to develop.

In Extreme Programming [2] each project starts with a metaphor, an evocative description of the programme architecture, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as an intuitive model for an agent-based information retrieval system [7].

When you try to implement an intuitive idea you have to destroy its holistic Gestalt first. The Intuition is simple – breaking it up means making it more specific and more complicated. The confidence in the whole idea is gone, when you look at the pieces. So it seems to be plausible to assume some kind of resistance to splitting an intuitive model (splitting barrier).

Spohrer et al. [13] model the process of solving a programming task using goal-plan-trees (GAP-trees). The root of this tree is a plan that represents the original intuitive idea for a solution of the problem. Several subgoals correspond to a plan and all these subgoals must be reached to realize the plan. For each subgoal there exists a couple of different plans representing alternative implementations. The problem solver has to choose one of them. The researchers analysed semantic bugs in 46 novice programmers' first syntactically correct programmes. Two observations support the assumption of a splitting barrier. Firstly 103 out of 549 errors were missing guards of entry data ([13] p. 373). One explanation of this is that the novice programmers tried to solve a simplified version of the original task first in order to extend the functionality later. This is basically the strategy of agile software development methodology like Extreme Programming (XP). In XP the developers start with an "architectural spike solution" which is a minimal executable programme that models the architecture of the big system that has to be developed. Note, that the

simplified programme is not just a *part* of the whole system. It is a *model* and a preliminary implementation of the *same* intuitive concept that the final solution would be an implementation of.

The second observation, which should be mentioned, is that novice programmers tend to merge goals. That means they use a single plan to achieve several goals in an integrated manner. Unfortunately the avoidance of splitting a plan into separate goals leads to more complicated programme code and as a consequence to more bugs.

2.2 Difficult Procedural Intuitions

An intuition can happen to be unsuitable for implementation. To be more specific, it can include aspects the programmer is not able to implement because she or he does not know appropriate programming concepts. Consider approaches for the solution of the sorting problem. A popular exercise to get some ideas about sorting is that students are asked to file in front of a wall sorted by height. Every student can do that without any special knowledge. After some reflection, how they solved the problem, an intuition might evolve that could be verbalized like that:

“Everybody moves to a place that on the left hand side there is no taller person.”

It is immediately reasonable that a group can be sorted in that way. The simplicity and Gestalt property is partly achieved by using the concept of parallel processing. The objects to be sorted are supposed to act independently. The intuition describes the behaviour of a single object and ignores the necessity of a central control mechanism.

A person, who tries to use this idea for a programme that sorts an array of numbers, would encounter severe difficulties. Each number would have to be represented by a separate thread. The programmer would have to develop complex protocols for the communication between the objects taking into account the possibility of deadlocks and infinite loops. Each object would have to find out its left neighbour and contact it to compare values. Neighbours might have to be informed when an object changes its position in the array and so on. It is quiet obvious that this intuitive idea – though easy to grasp – is not appropriate for implementation by a novice. The art is to recognize this as early as possible and search for a different approach.¹

2.3 Missing Connections Between Intuitions and Formal Programme Code

At least in Germany, where card games are very popular, all school children are able to sort a deck of cards intuitively. Nevertheless many students fail, when they have to write a program that sorts an array of numbers, even if they know all necessary programming concepts like if-statements and loops. They know for sure what to do but are not able to express this knowledge using the programming language.

It seems that some mental connections between intuitive models and corresponding programme code are missing. Furthermore some intuitions might be unconscious. Consider a fictive girl named Sandra, who is sorting some playing cards holding in her hand (figure 1). She intuitively masters the straight selection sorting algorithm – not knowing that she is applying this algorithm. She does not have much reflected declarative knowledge about what she is doing (otherwise we wouldn't have a case of

¹ Bell, Witten and Fellows [3] describe other class room activities for sorting (using balance scales), that are much easier to implement.

genuine problem solving). But she is fully aware that she *can* sort the cards in her hand. Sandra holds the cards in her left hand, removes the smallest (two of hearts) and puts it at the beginning of the sequence. Between two of hearts and the second card she leaves a gap. Then she searches in the subsequence on the right of the gap for the smallest value (five of hearts) and moves it to a place directly on the left of the gap. She knows that the cards on the left hand side of the gap are sorted and those on the right are not. She continues, the gap "moves" to the right until it reaches the end and the whole sequence is sorted. Sandra is well aware that she did this systematically and that this might be a good intuition for a computer programme that sorts a sequence of numbers.

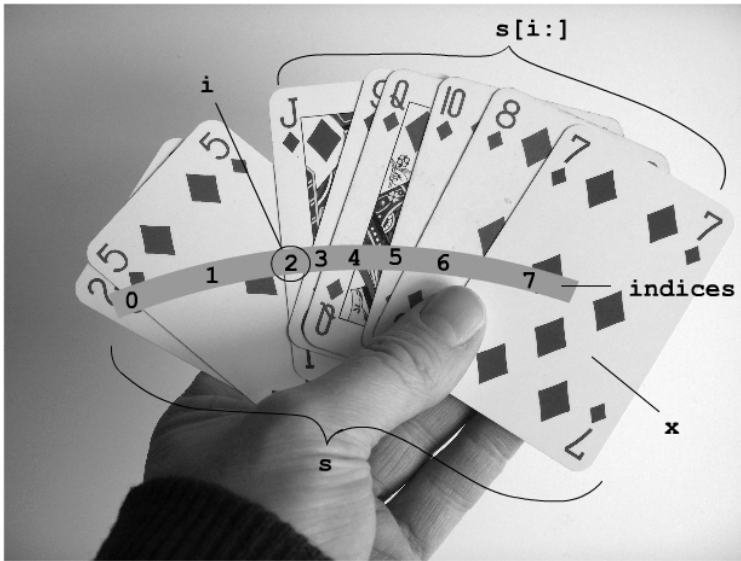


Fig. 1. Snapshot from the process of sorting cards applying straight selection

If Sandra is trained to manage intuitions and knows how to represent programme concepts by different intuitive models she should be able to analyse her intuitive action and create a programme. The cards in her hand are arranged in a linear sequence, so they can be represented by a list - a data structure that is provided by the programming language she knows (in this case she uses Python). The search for the minimum is restricted to a part of the list, which starts at a variable position (right of the gap) and extends to the right end. This part can be defined using a special feature of Python ($s[i:]$). The gap marks the index of the first item of the unsorted sublist. Such a gesture-like marking corresponds to a name in a computer programme – for example i . Sandra knows from her own experience that in every-day life there are many things that are – from a computer scientist's point of view – actually names. A pin on the street map can mark a scene of crime (and is thus a name of a place) and a bookmark is a name for the page of a book where someone has stopped reading. One of the most important functions of names is addressing. They serve to identify and find a specific object among many objects. If Sandra has an intuitive understanding of

lists, she should know that removing a card from a list is something different than reading a card and copying it.

By analyzing her partially unconscious sorting procedure, reading the clues in her action, using her knowledge about metaphorical representations and realizing her tacitly used intuitions she might succeed in developing a programme like this:

```
def sort (s):
    indices = range(len(s))
    for i in indices:
        x = min (s[i:])
        s.remove (x)
        s.insert(i, x)
    return s
```

2.4 Misconceptions

Misconceptions about the meaning of programme code can be obstacles in the process of programme development. Misconceptions are often tacit and become only obvious in critical situations. It has been observed that students were able to write quite complex programmes and still had not understood some basic concepts. This might be a consequence of an example-based problem solving strategy (see: [1]), in which students search for the solution of a similar problem, use it as pattern and just change a few things without deep understanding what they are doing. Some typical misconceptions in informatics concern the interpretation of assignment statements like these two lines of Python Code

```
a = 3
b = a
```

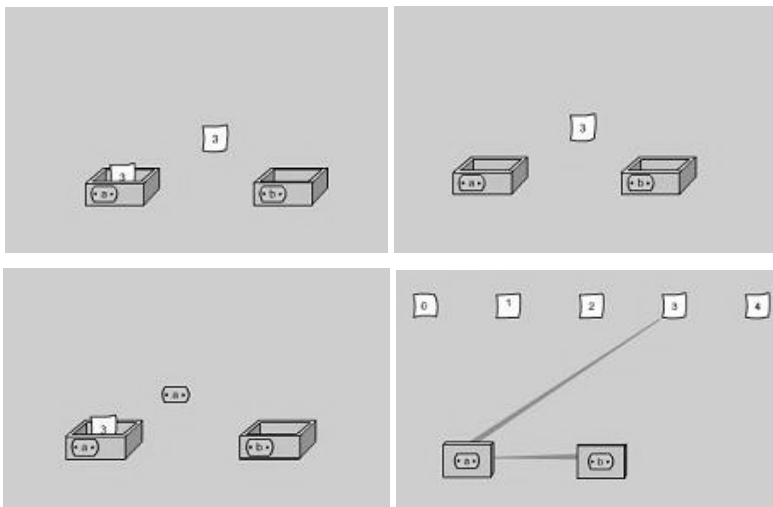


Fig. 2. Four screenshots from animations of the Python Visual Sandbox visualizing the execution of the assign statements `a = 3`; `b = a`

Figure 2 shows screenshots from animations that visualize the execution of this programme. The first animation visualizes the assignment as copying a value. This is a quiet appropriate intuition. The other three models are misconceptions. The second animation models an assignment as transfer of content. This is obviously not appropriate, because in the model container *a* is empty after the execution of the statement. In the third model not the content (object) but the name of the variable is copied and stored as content. In the last animation *b* does not point to the object 3 as it should be but to the name *a*. This is a visualisation of the (wrong) idea that *b* is connected to *a*. Any change of *a* would affect *b* too.

Table 1. Percentage of high school students ($N = 92$), who accepted a model as appropriate to explain the execution of the statements $a = 3$; $b = a$

Model	Percentage of students ($N = 92$)
Copying data (top left)	91,3 %
Transferring data (top right)	50,6 %
Copying name (bottom left)	68,5 %
Pointing to the name (bottom right)	61,54 %

From October 2005 to January 2006 92 high school students in Germany and Hong Kong (China) judged these animations (among many others) while playing a game of the Python Visual Sandbox (see section 3). Table 1 shows that a majority did also accept the misleading animations as appropriate models.

The problem of misleading intuitions is not their existence but the fact that they are often unconscious. diSessa et al. [11] point out, that an intuitive concept is never wrong per se but might be inappropriate in some contexts. Intuitions cannot be removed and replaced by better intuitions (they are persistent) but you must realize them and know how to use them. The first step of achieving this competency is to externalise intuitions, that is to find a verbal or visual representation which you can tell other people and discuss it.

3 Overcoming the Barriers

The barriers, which were discussed in the previous section, can be overcome by practising the use of intuition during programming. The Python Visual Sandbox (PVS) is a collection of different online-applications, which I have developed from 2003 to 2005 [16], [17]. It contains approximately 150 animations modelling concepts of the programming language Python. The PVS is both a research tool and a learning environment. Players' actions are recorded and stored in a database for later evaluation. There are three types of games. "Python Visual" shows a couple of animations using different analogies to illustrate the working of a tiny Python script or single statement. Some of them might be definitely wrong. The player chooses the animated model which explains the Python code best. In "Python Quiz" the player analyses a small Python script by assigning appropriate animated models to Python statements. She or he can earn points for correct decisions. "Python Puzzle" is some

kind of visual program editor. The player manipulates pieces of Python code with the mouse and "builds" a function definition for a given task. This section focuses on the question in what way playing in the PVS might help to enrich one's repertoire of intuitive models and train the use of them during program development.

3.1 Deep Tracking of Programme Code - Reflecting Intuitive Models

Perkins et al. [10] observed, that successful problem solvers were able to explain a programme line by line. They call this activity "close tracking of code". The researchers also analysed teachers' scaffolding strategies. The most successful cues were those, which focused on close tracking of code ("Explain your programme!"). While trying to explain the programme in many cases students detected logical errors by themselves.

When people try to tell other persons what they think how a programme works, they externalise their intuitive models of programming concepts. Visualisations seem to be appropriate to express the Gestalt of mental models and are therefore used in textbooks and other professional teaching media. But they are difficult to design. In the early nineties of the last century 604 persons from three European countries were asked to explain with pencil and paper how to search for a text file, display the text on the screen and print it [14]. Only a minority used any visual representations.

So it seems to be a difficult task to create visualizations of internal intuitions. Nevertheless this should be very instructive and the ability to visualize is a very important scientific competency [5]. But on the other hand it affords a lot of experience and is time consuming. Python Quiz is a "quicker" approach. The players are confronted with ready made visual animations, which are supposed to illustrate the execution of one or two lines code. They have to decide whether the model is good or bad. Most of the animations are acceptable but they all have critical aspects, which provoke discussion, and some are definitively inappropriate.

Diversity seems to be important in this context. There are usually different metaphors for the same thing. Chiu observed in the field of mathematics that experts use more different metaphors explaining arithmetic operations in the context of a problem solving process than novices [4].

The advantage of animations in comparison to verbal representations is that they are very clear and explicate details of a metaphor that might be hidden in a verbal representation.

Referring to the former mentioned term "close tracking" the explanation of programme code by visual animations might be called "deep tracking" because it includes a deep reflection of the intuitions that are used to grasp the semantics of a programme. Thus playing Python Quiz is an opportunity to think about limitations of analogies, detect misconceptions and extend the repertoire of visual representations.

3.2 Practising the Use of Intuitive Models

Python Puzzle is designed to practise the use of intuitive models while actively solving small programming tasks. Figure 3 shows a screenshot.

The editor field (entitled "Script") contains the head of a Python function definition and a call of this function in a print statement. The body of the function definition is missing. The player can manipulate code fragments with the mouse on the screen and



Fig. 3. Screenshot from Python Puzzle

“build” the body of the function. He or she can test the program and earns points, if it returns the expected result.

When the player clicks on the button labelled *hint* she or he sees an animation, which explains, what the function is supposed to do (Fig. 4).

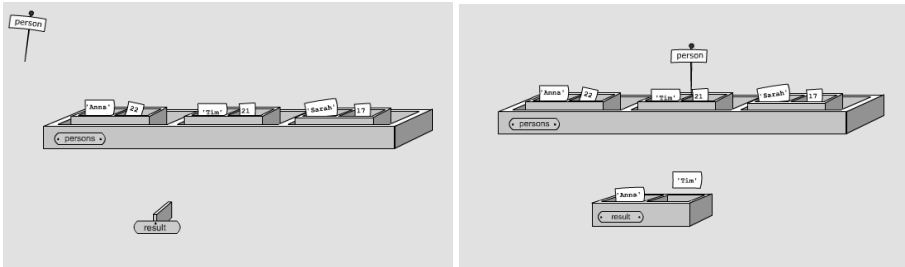


Fig. 4. Animation screenshots, visualizing the execution of a retrieval function

In this example a group of persons is modelled by a list of lists of the form *[name, age]*, e.g. *[['anna', 22], ['Tim', 22], ['Sarah', 17]]*. The function *getNames()* is supposed to return a list with the names of all group members. The correct solution in this example is the following Python script:

```
def getNames(persons):  
    result = []  
    for person in persons:  
        result.append(person[0])  
    return result
```

In the animation (fig. 3) the list, which is used as argument in the function call, is represented by a box with three compartments each containing a smaller box with two compartments. A pin labelled with "person" is hopping from box to box indicating the actual item of the iteration. In each step of the iteration a copy of the first element in the actual box (the name of a person) moves to the list `result`, which is visualized by a "growing" box, and is stored in a new compartment on the right side.

Solving a task the player first has to interpret the animation in order to find out the expected functionality. If a player is not able to construct an appropriate programme text with the given parts, she can watch the animation again. But this time she analyses it more carefully trying to find code fragments that match to subsequences of the animation ("close tracking"). For example at the beginning of the movie an object representing an empty list appears. In this animation lists are visualized through boxes with several compartments, which contain the elements (items) of the list. An empty list in this model is a box with no compartment. This box is "growing" during the execution of the function. Further compartments are added and filled with strings. The programme code corresponding to this first action is the statement `result = []`. So close tracking of the animation leads to the insight that the function definition should start with this statement.

Some Python Puzzles contain many different animations modelling the execution of the same programme. Note that these models are not complete visualisations of a running programme showing every detail of its structure. They are simplified, sometimes focus on special aspects and are more or less abstract. In this respect they differ from "computer models" like in [9] which are supposed to simulate the execution of programmes.

4 Conclusion

CS classes should provide opportunities to discuss and reflect intuitive models of programming concepts. Those who know how to use verbal and nonverbal representations of intuitive models, might get a deeper understanding of informatics concepts, have less difficulty to communicate and cooperate with others during a software developing process and succeed in creating computer programmes on the basis of intuitive procedural knowledge.

References

1. Anderson, John.R. Cognitive Psychology and Its Implications. 6th edition. Worth Publishers New York (2004)
2. Beck, Kent: Extreme Programming Explained. Addison Wesley, Boston San Francisco New York Toronto Montreal London Munich Paris Madrid Capetown Sydney Tokyo Singapore Mexico City (1999)

3. Bell, Tim; Witten, Ian H.; Fellows, Mike: Computer Science Unplugged. Off-line activities and games for all ages. <http://unplugged.canterbury.ac.nz> (1998)
4. Chiu, Ming Ming: Using Metaphors to understand and solve arithmetic problems: Novices and experts working with negative numbers. In: *Mathematical Thinking and Learning*, 3.3 (2001) 93-124
5. diSessa, Andrea A.: *Changing Minds. Computers, Learning and Literacy*. The MIT Press, Cambridge, London (2001)
6. Fischbein, Efraim: *Intuition in Science and Mathematics. An Educational Approach*. D. Reidel Publishing Company, Dordrecht Boston Lancaster Tokyo (1987)
7. Jeffries, Ron: What is Extreme Programming. <http://www.xprogramming.com/xpmag/whatisxp.htm> (2001)
8. Leiser, David: Scattered Naive Theories: Why the human mind is isomorphic to the Internet Web. In: *New Ideas in Psychology* 19 (2001) 175-202
9. Mayer, Richard E.: The Psychology of How Novices Learn Computer Programming. In [12] 129-160
10. Perkins, D.N.; Hancock, Chris; Hobbs, Renee; Martin, Fay: Conditions of learning in novice programmers. Educational Technology Center, Harvard University. In: [12]
11. Smith, John P.; diSessa, Andrea A.; Roschelle, Jeremy: Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. In: *Journal of the Learning Sciences*, Vol. 3 (1993/94) Nr. 2
12. Soloway, Elliot; Spohrer, James C. (ed.): *Studying the Novice Programmer*. Hillsdale, (1989)
13. Spohrer, James C.; Soloway, Elliot; Pope, Edgar: A Goal/Plan Analysis of Buggy Pascal Programs. In: [12] 355 - 399.
14. van der Veer, Gerrit C.: Mental Models of Computer Systems: Visual Languages in the Mind. In: Tauber, M. J.; Mahling, D.E.; Arefi, F. (Hrsg.): *Cognitive Aspects of Visual Languages and Visual Interfaces*. North-Holland, Amsterdam (1994)
15. Weigend, M.: *Objektorientierte Programmierung mit Python*. 2nd Edition. MITP Bonn (2005)
16. Weigend, M.: Intuitive Modelle in der Informatik. In Friedrich, Steffen (ed.): *Unterrichtskonzepte für informatische Bildung. INFOS 2005 Proceedings*, Bonn (2005) 275-284
17. Weigend, M.: The Python Visual Sandbox. <http://www.python-visual-sandbox.de> (2006)

On Novices' Local Views of Algorithmic Characteristics

David Ginat

CS Group, Science Education Department
Te-Aviv University
Tel-Aviv 69978, Israel
ginat@post.tau.ac.il

Abstract. The solution of an algorithmic task requires rigorous insight, based on the task's characteristics. Problem solvers seek insight in diverse ways, from different points of view. Experts usually seek a global, assertional perspective. Such a perspective is not natural to many novices, who often turn to local viewpoints. However, such points of view may yield erroneous outcomes. This study displays three different facets of novices' improper local points of view. The three facets involve local substructures, greedy traps, and unsuitable design patterns. Novices' erroneous solutions to three colorful tasks are described and analyzed, in comparison with the desired solutions, and suggestions are made for elaborating student awareness of the need for a global, rigorous point of view in algorithmic problem solving.

1 Introduction

Consider the following 2-step process. Two equal cups are placed on the table, the first with milk, and the second with coffee (same amounts of liquid). One takes a spoon from the milk cup, pours it into the coffee, and stirs well; then takes the same spoon from the coffee cup, pours it into the milk cup, and stirs well. Is there more coffee in the milk cup, more milk in the coffee cup, or are the amounts of coffee-in-the-milk and milk-in-the-coffee equal?

In our experience, many novices are rather quickly convinced that “there is more milk in the coffee”; and they justify: “in the first transition, the spoon is pure milk, whereas in the second – it is not pure coffee (as it contains a bit of milk)”. The latter justification is derived from a misleading *local point of view* that does not yield the whole picture. When the spoon is used in the second transition, it indeed is not pure coffee, but it is poured into an amount of liquid smaller than the amount into which it was poured in the first transition.

The correct answer is “equal amounts”. Its justification is obtained from a *global point of view*: the amounts of liquid in the cups are equal in the beginning and equal in the end; the total amounts of milk and coffee in the system are equal throughout the process; thus, the amount of milk “missing” in the milk cup is exactly the amount of coffee “missing” in the coffee cup.

The novices' local viewpoint enfold an operational perspective of following the dominant pieces of data in the operations performed, without examining all the relevant data that describe the system's states (specifically, the initial and the final

states). This deficiency of lacking a global perspective of the states of the 2-step algorithmic process yields the erroneous conclusion.

The goal of this paper is to illuminate various facets of the latter novices' deficiency, of a limited, local point of view. The coffee-milk example illustrates a local viewpoint that involves operational reasoning based on partial, insufficient data. Additional examples in this paper demonstrate further facets of reaching erroneous convictions due to a local point of view. All these facets enfold partial, limited perspectives, which lack the notion of an assertional point of view of algorithmic behavior.

Assertions are the logical entities underlying program verification [4,8,10], which involves logical clauses describing algorithmic transitions from an initial to a final state. One particular kind of ordered, logical assertions is that of invariants [4,7,10]. Although program verification is sound and complete, its encapsulation in program design and analysis is debatable, due to its unnatural and formal logical specifications [5,7,9].

Many tutors turn to more natural perspectives, or points of view, upon program design and analysis [5]. Yet, one must be cautious with natural tendencies that may lead to undesirable outcomes. In this paper, we show the unfortunate tendency of viewing algorithmic characteristics through a local perspective, rather than a global one. The illumination of this tendency sheds further light on the essential role of assertional elements, which are the core of the assertional point of view of algorithmic behavior.

Our demonstration is based on an ongoing study we conducted with hundreds of talented high-school students, which competed in our national CS Olympiad competitions, at different levels. The results were collected in the course of the last three years, from written notebooks and personal interviews. In what follows, we display results of student solutions to three colorful and unfamiliar algorithmic tasks, which were posed to them during the competitions. The results reveal student tendencies to examine tasks solely from local points of view, which led them to undesirable, erroneous outcomes.

In each of the next three sections we display an algorithmic task and its student solutions. Each section is named according to the particular facet it reveals of novices' unfortunate tendencies towards local points of view. The novices' erroneous solutions are presented as algorithmic ideas, rather than formal algorithms or computer programs. The erroneous solutions are followed by falsifying examples and by the solutions of the desired global perspective. The paper is then concluded in a discussion section, in which the displayed facets of the local viewpoints are analyzed and discussed, and corresponding suggestions are offered to educators.

2 Local Substructures

Algorithmic tasks often involve computations with various structures of data, such as sequences and graphs. In particular, many algorithmic tasks involve inputs composed of sequences of elements and outputs related to some ordering of these elements. When a task involves absolute ordering requirements, one should be careful with relative ordering. The following tasks illustrates this phenomenon.

Zigzag. Develop an algorithm (computer program) for which the input is a random sequence composed of N 1's and N 2's, and the output is the minimal number of swaps between (not necessarily adjacent) pairs of integers that yield a zigzag ordering (i.e., either the sequence 1 2 1 2 ... 1 2, or 2 1 2 1 ... 2 1).

For example: for the input 1 1 2 1 2 1 2 2 the output will be 1 (the first 1 will be swapped with the last 2), and for the input 1 1 1 2 2 2 2 1 the output will be 2.

The original goal in posing this task to students was to observe whether they can elegantly relate the given sequence state with the desired sequence state, without actually performing the swaps. In particular, we expected the students to seek a global pattern that characterizes the desired zigzag state. Unfortunately, to our surprise, many students approached the task by focusing on local sequences of equal integers. They noticed that when two adjacent integers in the sequence are identical, one of them should be swapped. This characteristic encapsulates a relative, local notion, rather than an absolute, global one. Nevertheless, quite a few students chose to turn to this local, subsequence characteristic for calculating the minimal number of necessary swaps. They offered one of the following arithmetic values as the minimal number of necessary swaps.

- a. *Half the sum of the lengths of subsequences of equal 2's* (where a subsequence is at least two 2's, and "half" means "div 2"). Thus, for 1 1 2 2 1 1 2 2, the total sum of the lengths of consecutive 2's is 4, so the output will be 2 ($=4 \div 2$); for 1 1 1 2 2 2 1 2 1 2, this total sum is 3, so the output will be 1 ($=3 \div 2$).
- b. The total sums of subsequences of 1's and subsequences of 2's may be different, therefore the *smaller* among them should be chosen in part a.
- c. The total sums of subsequences of 1's and subsequences of 2's may be different, therefore the *larger* among them should be chosen in part a.
- d. The total sums of subsequences of 1's and subsequences of 2's may be different, yet not the smaller and not the larger among these values should be chosen, but rather *half the total number of pairs of adjacent integers*.
- e. What matters is only the length of *the maximal subsequence of equal integers*, therefore the desired output is half the length of the maximal sequence.

The above solutions a-e are the main five variants we observed of the students' local points of view. (Several similar variants were also offered by a few additional students.) The students who offered these solutions mentioned that they derived them from various examples, which showed the above patterns. Their main justification arguments for all these approaches were based on the notion that "two equal elements may not be adjacent to one another in the desired zigzag ... therefore one of them should be swapped ... and since a swap may set the final locations for both of the swapped elements ... the total length of equal values should be divided by 2."

Some of those who offered approaches b, c, and d, indicated that they started with approach a, and "fixed" it after realizing that it does not always "work". Did their "fixing" yield the desired result? Not quite. The patched "corrections" yielded

approaches b, c, and d, which still encapsulate the same train of thought of a local viewpoint.

None of the above a-e solutions is correct. The following input example falsifies all these solutions: 1 2 1 2 2 1 2 1, as for this input the output according to each of the solutions should be less than 2 while it actually should be 2.

The proper way to examine the given task is through global lenses, while focusing on the locations of the different integers in the desired zigzag outcome. Three important assertions may be stated:

1. In a zigzag form, the 1's are all in odd locations, or all in even location. The same holds for the 2's.
2. In the initial (input) sequence: for every 1 that is in an odd location there is a corresponding 2 in an even location, and vice-versa. Thus, a single swap between two different values that are not in their desired locations will place both of them in their desired locations.
3. The minimal number of swaps is equal to the minimum among: the number of 1's initially in the even locations and the number of 1's initially in the odd locations.

The computation derived from the above assertions involves a single, $O(N)$ pass over the input, for calculating the relevant data mentioned in the third assertion.

All in all, the illuminating picture was derived from a global point of view. The above assertions are not related to input subsequences. In addition, they easily yield the input sequence 1 2 1 2 2 1 2 1, which falsifies the a-e, local view solutions. Unfortunately, a non-negligible amount of students did not seek a global point of view, but rather turned to a local one. In our impression, they followed their natural tendencies, while being unaware of the potentially erroneous outcomes that may be obtained. We further elaborate on this in the concluding discussion section.

3 Greedy Progression

Many algorithmic tasks are solved by following a greedy approach [3]. However, one must be careful with greed. It may seem completely natural to follow the greedy way in solving many algorithmic tasks, but the notion of “make the best progress you can in the next step” is a local notion, which may yield erroneous global results. In particular, a sequence of local optimal steps does not necessarily yield global optimization. (For example, a greedy approach of calculating in three steps the minimal number of coins for a payment in a coin system of 25, 10, and 1 cent, may yield non-optimal outcomes. If the change is 39, then the greedy approach will yield the minimal amount of coins (one 25, one 10, and four 1's); but if the change is 40, then this approach will not yield the minimum number of four coins.) Novices are often not aware of this “greedy trap”, which encapsulates progression through a series of local decisions. We demonstrate it with the following non-optimization task.

Board Reconstruction. The description of an $N \times N$ board, of Black/White entries, has to be transmitted. Unfortunately, it is impossible to transmit quadratic-size information. Only linear-size information may be transmitted. Thus, the number of Black entries in each row and each column is transmitted rather than the (Black/White) description of each board entry. The receiver of this transmission has to reconstruct a corresponding $N \times N$ board (there may be more than one board that will suite the transmitted information). The task is to develop the receiver's algorithm of reconstructing a suitable board.

Develop an efficient algorithm for which the input is two lines of N elements each – the first line describes the number of Black entries in each of the N rows and the second line describes the number of Black entries in each of the N columns. The output is a suitable Black/White matrix description.

Note 1: The task is described as a board reconstruction task, but it is also relevant as an allocation, or a scheduling task. For example, it may be a task of allocating employees to activities, where each employee may participate in some given number of activities and each activity may require a specified number of employees. Other application may be relevant as well.

Note 2: We assume that the input is legal; that is, it corresponds to some board layout. Obviously, no input value may exceed N , and the sum of the row values should equal the sum of the column values. (Are these sufficient conditions?)

It may be suitable to reconstruct a valid board by constructing its rows one-by-one, from top to bottom. However, the way each row will be constructed is important. One simple greedy approach may yield the construction of a row's columns one-by-one, from left to right, while trying to color Black each constructed entry. This kind of progress enfold a local point of view, of "taking care of the next entry regardless of the values that will be left for future entries". It makes sense and seems natural. It also "works" for many input cases (e.g., the input $\langle 1 \ 2 \ 1 \rangle \ \langle 2 \ 1 \ 1 \rangle$ (i.e., the board is of size 3×3 , the 1st row contains 1 Black entry, the 2nd – 2, and the 3rd – 1; and the 1st column contains 2 Black entries, the 2nd – 1, and the 3rd – 1). However, does it "work" for every legal input? Students offered the latter greedy approach and additional ones. We display below the three main approached we have encountered.

- a. The board entries are colored Black row-by-row, while coloring Black the leftmost entries that can be colored in each row. For example, for the input lines $\langle 1 \ 1 \rangle$ (rows) and $\langle 1 \ 1 \rangle$ (columns), of a 2×2 board, this solution yields the board of Figure 1 below:



Fig. 1. Greedy coloring of the board row-by-row, each row from left to right

- b. The board entries are colored Black diagonal-by-diagonal, each diagonal from left to right, while starting from the main diagonal, and then proceeding with the other diagonals one below and one above, until the board corners are colored. This approach yields the board of Figure 2 below for the input lines $\langle 1 \ 1 \ 2 \rangle \ \langle 1 \ 1 \ 2 \rangle$:

	1	1	2
1			
1			
2			

Fig. 2. Greedy coloring of the board diagonal-by-diagonal, starting from the main diagonal and “diffusing” to both sides

- c. The board entries are colored Black according to the intersections of the largest row and column values; that is – the entry whose indices are the intersection of the largest remaining row value and largest remaining column value is the next entry that is colored Black. This approach yields the board of Figure 3 below for the input $\langle 1 \ 2 \ 1 \rangle \ \langle 1 \ 0 \ 3 \rangle$:

	1	0	3
1			
2			
1			

Fig. 3. Greedy coloring of the board according to the intersections of the largest row and column values (while breaking ties to the top and to the left)

The reader may notice that the input of Figure 2 yields a “dead-end” (incomplete solution) for approach a, and the input of figure 3 yields a ”dead-end” for both of the approaches a and b. These observations were made by students who offered the greedy approaches b and c, after realizing the difficulties with approach a.

Unfortunately, approach c may also yield a “dead-end”. For example, for the input sequences $\langle 1 \ 1 \ 3 \rangle \ \langle 1 \ 1 \ 3 \rangle$ the entry $\langle 3,3 \rangle$ will be colored first, and the board will look as in Figure 4 below:

	1	1	2
1			
1			
2			

Fig. 4. The board state after the first computation step of approach c on the input $\langle 1 \ 1 \ 3 \rangle \ \langle 1 \ 1 \ 3 \rangle$

At this stage, approach c requires a repeated coloring of entry $\langle 3,3 \rangle$, which is already colored Black. Students patched a quick correction: "... in situations like these the algorithm will go for the second largest value among the columns, or the rows ...". Then, the same value may appear more than once. This was solved by "... taking the leftmost among the equal columns ... and the upper one among the rows ..."; i.e., an encapsulation of greedy approach a. Do these patches always "work"? The two patches yield the board of Figure 5 below after three computation steps:

	0	1	1
0			
1			
1			

Fig. 5. The board state after three computation steps of approach c on the input $\langle 1\ 1\ 3 \rangle \langle 1\ 1\ 3 \rangle$

The next step yields the board of Figure 6 below, after resolving the conflict of identical values, according to the approach of "uppermost and leftmost".

	0	0	1
0			
0			
1			

Fig. 6. The board state after four computation steps of approach c on the input $\langle 1\ 1\ 3 \rangle \langle 1\ 1\ 3 \rangle$

The above result is a "dead end". Upon realizing states such as the above some students started rethinking of their solution approach. Yet, even at this stage, many still sought further, local patches.

A local point of view is unsuitable here. A different perspective is required, based on an illuminating characteristic. It may still be possible to color the board row-by-row, without coloring each row in a greedy way from left to right; yet a "safe", successfully-ending coloring is required. A "safe" coloring of the rows one-by-one must keep invariant valid intersections between the values of the remaining rows and the remaining columns. In particular, after coloring a row, the remaining column values should coincide with the next row value. This may be achieved by keeping the following invariant during the computation progress:

Coloring Invariant. *In coloring the rows one-by-one, after a row is colored, each of the column values does not exceed the number of the remaining rows, and each of the row values does not exceed the number of the remaining columns (which are not yet zero).*

The natural way for keeping the invariant is by selecting the columns with the largest values for coloring a row. This guarantees that a maximal number of non-zero columns will remain after each row coloring. Thus, for the input $\langle 1\ 1\ 3 \rangle \langle 1\ 1\ 3 \rangle$:

first, the entry 1,3 will be colored; then – 2,3; and finally – all the three entries in the bottom row, as shown in Figure 7 below:

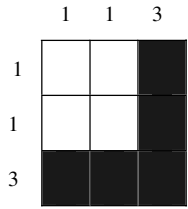


Fig. 7. Coloring the board row-by-row, while turning to the columns with the maximal values upon coloring each row

This approach will never yield a “dead-end”, as the invariant guarantees valid column values throughout the row-by-row progress. How costly is this algorithm? The list of column values should be initially sorted; and this requires $O(N\log N)$ time. Reordering of this list after coloring a row (and decrementing 1 from the larger list values) may be performed in $O(N)$ time. Thus, the total time complexity is $O(N^2)$, which is the lower bound for coloring an $N \times N$ board.

All in all, the illuminating picture was obtained from the above global invariant, which captured the core of the coloring progress. The resulting scheme partially involved a greedy component of progressing row-by-row, but the key point in coloring a row is not greed, or locality. It involves the global theme of coloring the entries that belong to the columns with the highest remaining values. Unfortunately, many students did not seek a global perspective, but tried and retried variants of local, greedy schemes, which encapsulated unfounded heuristic progress.

4 Unsuitable Design Patterns

Design patterns (or templates) are essential in algorithm and software design [1,6]. Programmers and algorithm designers repeatedly utilize them, at all levels. At the basic levels, they are iterators of counting, summation, maximum computations, searching, sorting, and the like. When an algorithm (program) designer selects a solution scheme for a given algorithmic task, she often invokes familiar design patterns that seem suitable. However, one should be careful with suitability. A particular pattern may only seem suitable, in cases where sufficient insight is lacking, possibly due to improper, “local” task analysis. This, unfortunately, is sometimes the case with novices, as can be seen in the following task.

Ordered Permutations. Design an algorithm for which the input is a positive integer N and the output is all the permutations of the integers $1..N$, such that in each permutation each integer is larger than all the integers to its left or smaller than all the integers to its left (e.g., for the input 3, the output will be the four permutations: 1 2 3; 2 1 3; 2 3 1; 3 2 1; notice that 3 1 2 is an invalid permutation, since the task condition does not hold for the rightmost integer 2).

The above is an enumeration task, which requires some ordered generation of elements (permutations). It makes sense to seek a link to some familiar design pattern that may be utilized as an enumeration scheme. Yet, such a link should be founded on sufficient insight, which yields the generation of all the valid elements. Unfortunately, some students only gained limited insight, through partial, local viewpoints, and offered a solution scheme based on a hasty design-pattern association.

These students looked at the example of the task definition, and noticed a seemingly relevant characteristic, which yielded a seemingly suitable design pattern. The students looked at the permutations as encapsulating a “rolling” scheme of values from left to right – first the rolling of 1, then the rolling of 2, etc. This notion is illustrated for $N=3$ in Figure 8 below:

1	2	3
2	1	3
2	3	1
3	2	1

Fig. 8. “Rolling” of 1, and then 2, in generating the valid permutations for $N=3$

The above rolling observation led these students to the familiar design pattern of Bubble-Sort [3]. Their idea was to “... reverse the order of the initial list $1 \dots N$ in a series of rolling swaps ... which will yield all the desired permutations ...”. In looking through the local lenses of a particular swap, one notices that this scheme indeed generates valid permutations, as each number is rolled through larger numbers until a smaller number is met. However, does this scheme yield all the valid permutations? Not quite. In examining the case of $N=4$, notice that the permutation $3 \ 2 \ 1 \ 4$ will not be generated by the above scheme, since in this scheme the rolling of the 2 starts only after the completion of the rolling of the 1.

A different, more insightful perspective is required. In carefully examining from a global viewpoint all the valid permutations for a given N , one may notice that *the rightmost element may only be 1 or N* (since if some K , $1 < K < N$, is the rightmost element, then both 1 (which is smaller than K) and N (which is larger than K) are to its left). If N is the rightmost element, then the second from right may only be 1 or $N-1$; and if 1 is rightmost, then the second from right may only be 2 or N ; and so on. Thus, one may set a permutation's values from right to left, while making one of two choices for the next generated element in the permutation. This observation implies a recursive scheme that yields all the 2^{N-1} valid permutations.

All in all, the illuminating picture was gained from a global point of view, which captured the important characteristic of all the valid permutations – their rightmost value, second-from-right value, etc. The local point of view, which was followed by the students, focused on the limited, misleading characteristic of “rolling”. This scheme, which was associated by some students with the design pattern of Bubble-Sort, yields only part of the valid permutations. The number of permutations it yields is by far smaller ($O(N^2)$) from the total number of valid permutations. Yet, students

who followed the latter local point of view did not examine the number of permutations. They were “caught up” with their seemingly suitable design pattern.

5 Discussion and Conclusion

The last three sections demonstrated different facets of novice tendencies towards local, limited-insight points of view, which yielded undesirable, erroneous outcomes. The different facets involved local substructures, greedy traps, and unsuitable design patterns. The novice tendencies seem to evolve from some natural inclination to “go for” the elements that “pop up” as relevant in the beginning of the problem solving process. In addition, it seems that the students lack awareness of the dangers of their natural tendencies.

The facet of turning to local substructures appears with algorithmic tasks that involve input structures such as lists and graphs. The example in section 3 illustrated the case of a list, where students tended to naively examine simple sub-list characteristics that encapsulated initial insight, which was insufficient for the task at hand. In our experience, novices show similar tendencies with graph tasks, where they sometimes examine simple sub-graph cases that lead them to erroneous characteristics.

The facet of greedy traps occurs with tasks of different kinds, in particular those that involve optimization. It is a natural tendency to construct a sequence of steps by locally simplifying, or optimizing each of the sequence steps. This approach often “works” in real-life situations. Yet, one should be careful. The example of section 4 illustrated various greedy approaches offered by novices, which all led to “dead ends”.

It is interesting to observe that in both examples, of sections 3 and 4, some of the students noticed the erroneous characteristics yielded by their own initial local approaches. Yet, instead of turning to a global, solid approach, they offered various local patching schemes, which were only seemingly helpful. Patching may be useful in cases where small, local details are improperly handled, but it cannot solve inherently incorrect designs.

The facet of turning to unsuitable design patterns is typical of novice behaviors in tasks that may seem at first glance similar to familiar standard schemes. This was also the case in the example of section 5. Superficial analysis, based on a small, limited set of examples, yielded associations to a familiar sorting scheme, which shared some particular local characteristics with the task at hand. This led novices to a hasty conclusion of the suitability of that scheme as the basis for the task solution. A global view shows very clearly the partial, limited correspondence between the scheme and the task. Yet, the students did not seek such a point of view.

In examining the novice behaviors according Schoenfeld’s cognitive model of problem solving [11], we notice improper students’ control and beliefs. In the sense of control, students showed hasty convictions of unfounded characteristics, based on partial local data. In addition, they neither orderly sought falsifying evidence for their convictions nor did they seek rigorous justifications of these convictions. In the few occasions where they realized some falsifying evidence, they patched local corrections that did not really help. In the sense of beliefs, it seemed that students felt

natural with their tendencies for local points of view, and were unaware of potentially misleading allies. In addition, they seemed to be unaware of the important role of the alternative global perspective.

An important lesson of this study for educators is the need to educate novices to carefully examine their convictions and seek rigor. One may turn to a local point of view, and obtain task characteristics from this perspective. Yet, one should be careful from hasty convictions, and seek justification through assertional arguments, which are derived from a global point of view.

Students may sometimes be more convinced of the need for rigor by realizing the fallacies of wrong, unfounded ways [2]. Thus, tutors may use in class examples like those above, in order to demonstrate the unfortunate outcomes of following local, erroneous allies. In this paper we displayed three facets of erroneous allies, related to novice tendencies for local points of view. We believe that there are additional facets of these tendencies, which can shed further light on unsuitable ways in which novices approach algorithmic tasks.

References

1. Astrachan, O., Berry, G., Cox, L., & Mitchener, G.: Design Patterns: an Essential Component of CS Curricula. Proceedings of the 28-th SIGCSE Symposium. ACM Press (1998) 153-160
2. Borasi R.: Reconceiving Mathematics Instruction: A Focus on Errors. Ablex Pub (1996)
3. Cormen T. H., Leiserson, C. E., and Rivest, R. L.: Introduction to Algorithms, MIT Press, Massachusetts (1991)
4. Dijkstra, E. W.: A Discipline of Programming. Prentice-Hall (1976)
5. Dijkstra E. W. et al.: A Debate on Teaching Computing Science. Communications of the ACM 32 (1989) 1397-1414
6. Gama, E., Helm, R., Johnson, R., & Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
7. Ginat, D.: Loop invariants, Exploration of Regularities, and Mathematical Games. Int. J. of Mathematical Education in Science and Technology 32 (2001)
8. Ginat, D.: Embedding Instructive Assertions in Program Design. Proceedings of the 9-th ITiCSE Conference. ACM Press (2004) 62-66
9. Ginat, D.: Mathematical Operators and Ways of Reasoning. The Mathematical Gazette (2005) 7-14
10. Gries, D.: The Science of Programming. Springer-Verlag. (1981)
11. Schoenfeld, A.: Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics. In: Grouws, D. A. (ed.): Handbook of Research on Mathematics Teaching and Learning. Macmillan (1992) 334-370

Learning Computer Programming with Autonomous Robots

Shuji Kurebayashi¹, Toshiyuki Kamada², and Susumu Kanemune³

¹ Shizuoka University

`eskureb@ipc.shizuoka.ac.jp`

² Aichi University of Education

`tkamada@aecc.aichi-edu.ac.jp`

³ Hitotsubashi University

`kanemune@acm.org`

Abstract. This paper reports on a programming lesson using autonomous robots in junior high school. First, the design of the low cost circuit board for the lesson is described. The structure of a general programming language “Dolittle” which controls a robot is also explained. Then, we introduce lessons of manufacturing and controlling robots in “Information and Computer” area of “Technology and Home Economics” subject for students (from 14 to 15 years old). From the result of the lessons we found that (1) learning programming is “hard fun” for students and (2) robot programming is effective for students those who have difficulties in learning. We propose introduction of learning programming with autonomous robots to IT education of junior high school.

1 Introduction

In Japan, IT education at the junior high school level is performed in “Information and Computer” area of “Technology and Home Economics” subject. The primary goal of the area is that all of the students have the ability to operate computers. However, because there sometimes occur severe problems in our daily life caused by flaws in computer software, teaching only how to use computer is not sufficient. It is impossible to foster citizens with IT literacy who can think of the potential risk of the highly information-oriented society. The reason comes from the fact that learning only how to operate computers cannot give students the substantial knowledge of fundamental structures and mechanism of computers. Thus, we think it is crucial to teach computer programming in IT education and the most important point is to show that every computer program is created by humans [1,2]. Normally, general programming languages are designed for IT specialists. Such language is too difficult and not suitable for children. Therefore, we have chosen the programming language “Dolittle” [3] for our educational practice because “Dolittle” is simple but has enough capacity to provide computer programs for children. The characteristic features of “Dolittle” are that (1) it has simple language syntax and that (2) it can use local language such as Japanese, Korean and English for instructions and identifiers including variable

names. We also developed a low cost autonomous robot as teaching material and the course of IT education using this robot. Then, we implemented the course. The reason of choosing a robot is that we found learning through controlling tangible objects is more effective than learning with operating virtual objects on the screen. Consequently, we have realized our teaching material can enhance the motivation of students to learn programming and assist them in learning that a lot of electrical products in our society are controlled by software.

In this paper, the reasons of introducing robot manufacture to our lessons are considered. Then the autonomous robots that we developed are presented. Finally, the performed lessons of “Technology and Home Economics” subject in junior high school are reported.

2 Meaning of Learning Programming in Elementary and Secondary Education

2.1 Learning Programming Is Learning “Manufacturing”

Hardware and software are both necessary for running computers. Software is constructed by humans as well as hardware.

2.2 The Effect of Robot Programming

Learning of computer science is essential for educating computer specialists. However, it's also important to teach both hardware and software to common students.

Instructions on computer hardware to common students are not easy. Therefore we thought that robot programming was effective for students. By experiencing robot control, students can learn programming from the viewpoints of both sides of hardware and software. Controlling real robots, students can see and understand each steps of the program. This provides a realistic feeling and motivation to students.

Thanks to robot programming experience, students start to figure out the motion of robots, think of algorithms, complete their programs and evaluate them. The skill of actual program development cycle is thus acquired. Students can learn hardware and software are both important.

2.3 Autonomous Robots Are Desirable for Lessons

In order to raise educational effects, autonomous robots are desired. It is because many electric appliances whose structure we want students to understand are nowadays equipped with microcomputers to enable autonomous operation.

Most of current computer software is downloaded from a computer network before being installed. Because autonomous robots need to download their software from the host computer before operation, they are quite suitable for providing concepts of computer communication with network and embedded system. Furthermore, from the property that software is transferable, students can easily conceive that software is an entity independent of computer hardware. Given

this perspective, we think autonomous robots are a very good teaching material to explain “programming is also a kind of making things” to students with persuasiveness.

2.4 The Educational Effect of Robot Manufacturing Experience

Students can see and look into the circuit board while building robots. They can observe the connection between sensors, motor drivers and a microcomputer. Students may not be able to understand the exact mechanism of the microcomputer or meaning of the circuit only by observing. Through handling electrical parts during wiring work, they come to understand the function of each part explained by a teacher; e.g. signal voltage for driving motors is output to signal pin of motor drivers; electric signals are sent from sensors and they are input to microcomputers etc. Then, they would understand the computer system deeper than before. They become aware that the most important part is CPU and that the computer system is not composed of only visible devices such as keyboards, mice or displays. Once this knowledge is obtained, students feel a sense of accomplishment that they completed the robot work all by themselves. We think this will encourage the interest and willingness to learn programming and increase teaching effectiveness.

3 Our Autonomous Robotic Car

We developed a new robotic car which can move autonomously. Students built their robots and wrote their programs to control them. The circuit and the controller board were designed by an expert. We adopted the PIC microcontroller which consists of Central Processing Unit (CPU), Random Access Memory (RAM), Read Only Memory (ROM) and Input and Output (I/O). ROM includes a monitor program (firmware) to interpret commands transferred from computers. RAM can store a program of up to 39 steps and two subroutines of up to 7 steps. I/O can control 2 motors which enable moving the robot forward, backward and turning it. This board can connect a switch as a sensor. The Program is transmitted from computers by means of an infrared ray interface. Fig. 1 is the controller board and an infrared ray interface.

4 Dolittle

4.1 Programming Language “Dolittle”

In Japan, teachers and students speak Japanese at school. Kanji and Kana characters which are unique to Japanese are used to express the language. Because students learn English in junior high schools, children are not familiar with the english/roman alphabet in elementary schools. That is why we adopted the Dolittle programming language. Dolittle is an object-oriented language designed for school education. Using Dolittle, students can write programs in Japanese, Korean and English. Fig. 2 is a sample of turning pentagon animation.

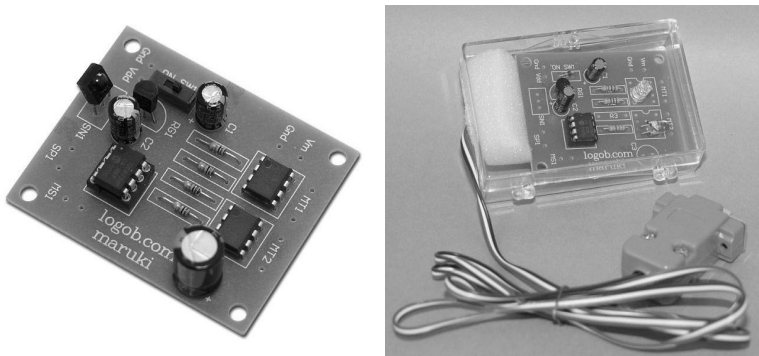


Fig. 1. The controller board and the infrared ray interface

```
kameta = turtle ! create.
```

In this statement, the “turtle” object receives a “create” message, then its “create” method creates a new object named “kameta”. The basic syntax of Dolittle is like “obj ! arg...msg.”. object “obj” receives message “msg” with some arguments.

```
pentagon = [kameta ! 100 forward 72 leftturn] ! 5 repeat makefigure.
```

“[...]” is a block. It makes an object that includes pieces of program code. By sending a “repeat” message to a block, we can repeat the code within the block. As a result, a turtle moves on the display, then a pentagon is drawn.

A figure drawn by the turtle graphics is only lines. But we can make a figure object from lines by executing the “makefigure” message.

```
clock = timer ! create.
```

A timer object runs program code periodically. In the above sentence, a timer object named clock is created.

```
clock ! [pentagon ! 10 0 moveby] execute.
```

“clock” moves the “pentagon” periodically on the display. Students can make animation programs using timer.

Students can make more diverse programs using buttons. They are very interested in programming because they can use buttons they made. Fig.3 shows an example of a program made by students.

```
button1 = button ! "turn" create.
```

This makes a button object named “button1”.

```
button1:click = [kameta ! 90 rightturn].
```

This defines a method which will be executed when “button1” is pressed. When the “button1” is pushed, the statement “kameta ! 90 rightturn” is executed, “kameta” turns 90 degrees.

4.2 Control Program

Dolittle can control external devices by accessing to external ports. As students can write programs in Dolittle syntax, they don’t need to write bytecode data which is transferred to robots. Fig.4 is a sample program of controlling

```

kameta = turtle ! create.
pentagon = [kameta ! 100 forward 72 leftturn]
! 5 repeat makefigure.
clock = timer ! create.
clock ! [pentagon ! 10 0 moveby] execute.

```

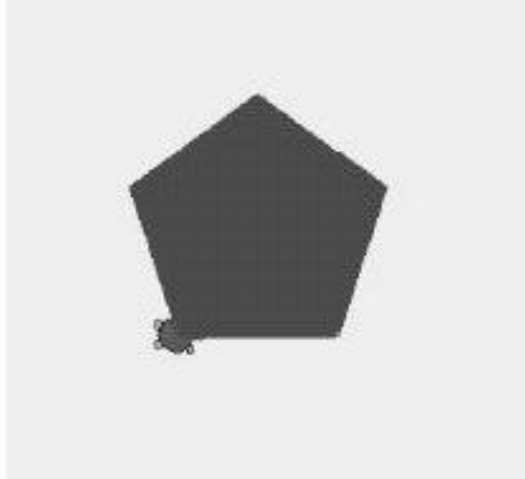


Fig. 2. Sample program of turning figure

robotic cars using Dolittle. This program makes a communication object named “aRobot”, then defines the method to send commands to the robotic cars. This method will be executed after opening the communication port. By this method, robotic cars move according to the following steps.

- Start when the switch is pushed.
- Move forward.
- When it collides with something, go back then turn left.
- Move forward again.
- When it collides with something, go back then turn right.

5 Lessons at a Junior High School

5.1 Curriculum

We conducted lessons at Nishimashizu Junior High School in Shizuoka prefecture. All of 135 students in the 2nd grade (14 years old) attended the lessons. Kurebayashi took charge of these lessons.

Table 1 is the curriculum of these lessons. First of all, student prepared programs in Dolittle language. They made animation programs using timer objects. Then they manufactured robots. They also wrote their own programs to control robot movement. Finally they produced game programs of Dolittle.

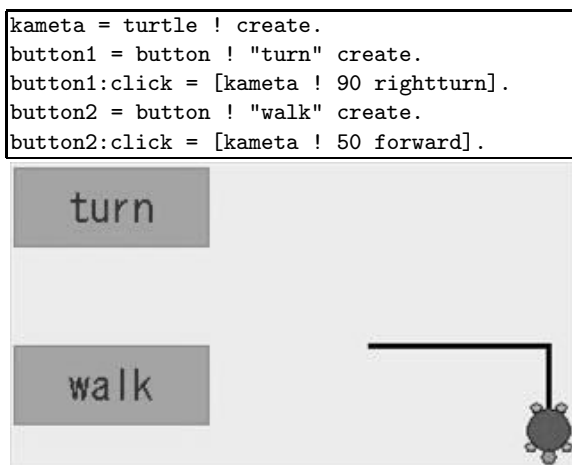


Fig. 3. Sample program using buttons

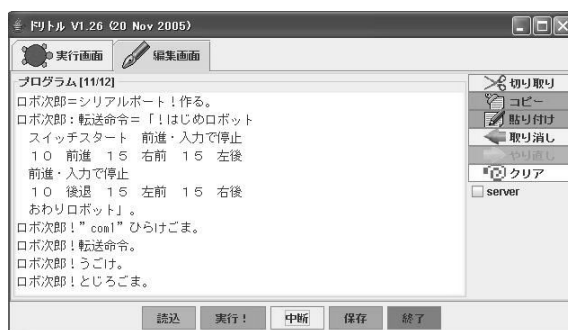


Fig. 4. Sample of controlling robotic cars (written on the edit screen of Dolittle)

5.2 Results

(1) Robot Construction. Students developed their robotic cars that have own shapes in 10 lessons. The body of the robots was constructed using the parts of a vehicle chassis kit for kids. Half of the students were beginners who have never used solder and screwdrivers. But all of them strived to assemble their own robotic cars seriously. Thus, all students were able to complete robot building. Fig. 5 is a robot made by one of the students.

(2) Controlling Robots. Students wrote control programs to run in maze courses in 8 lessons. Fig. 6 is a photo of the maze course and students.

Table 2 shows the results of a questionnaire for evaluating this class. More than 90% of students answered that it was fun to learn about robots. Most students were satisfied with learning of controlling robots. However, more than 70%

Table 1. Curriculum of lessons

Contents	Lesson hours
Let's make programs	8
Let's make robots	10
Let's control robots	8
Let's make programs	8

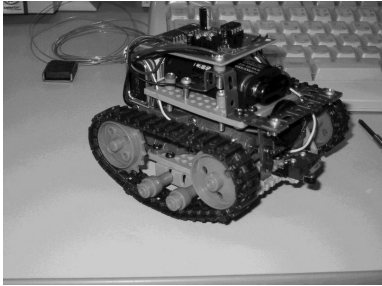


Fig. 5. Student's robot

students answered that it was difficult to control the robots by programs. Many students found it “hard fun”. Fig. 7 presents the comments of some students.

(3) Programming of Dolittle. In the first and the last lessons, students enjoyed Dolittle programming. They tried to develop game programs using button objects. They utilized their original idea. Fig. 8 is a screen-shot of the game and the student who programmed it.



Fig. 6. Maze and students

Table 2. Results of questionnaire (%)

question	Yes \longleftrightarrow No			
Programming of robots is fun	44	51	5	0
Programming of robots is easy	4	23	47	26

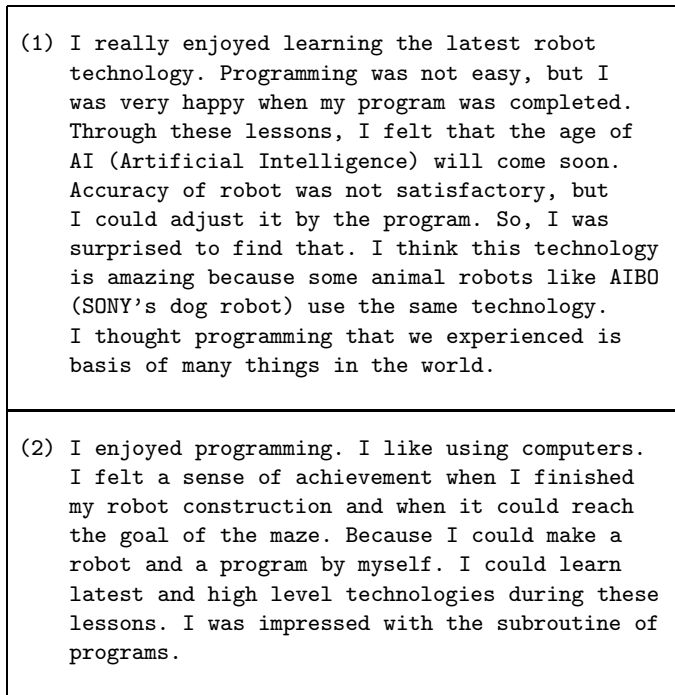
- 
- (1) I really enjoyed learning the latest robot technology. Programming was not easy, but I was very happy when my program was completed. Through these lessons, I felt that the age of AI (Artificial Intelligence) will come soon. Accuracy of robot was not satisfactory, but I could adjust it by the program. So, I was surprised to find that. I think this technology is amazing because some animal robots like AIBO (SONY's dog robot) use the same technology. I thought programming that we experienced is basis of many things in the world.
- (2) I enjoyed programming. I like using computers. I felt a sense of achievement when I finished my robot construction and when it could reach the goal of the maze. Because I could make a robot and a program by myself. I could learn latest and high level technologies during these lessons. I was impressed with the subroutine of programs.

Fig. 7. Comments of students

6 Discussion

The student in front of Fig. 7 said, “Accuracy of robot was not satisfactory but I could adjust it by the program. So, I was surprised to find that.” This comment shows he solved the mechanical problems by a program. His robot couldn’t move exactly straight when “forward” movement was commanded. By the experiences of controlling robotic cars, students could learn about inside and outside of the computer and linking the program implementation with the actual movement.

The second student shown in Fig. 7 said, “I felt a sense of achievement when I finished my robot construction and it could get to the goal of the maze. Because I made the robot and the program all by myself.” This comment indicates that students felt a sense of fulfillment through making robots and writing programs to control them. Students can feel deep achievement and motivation by developing their own robots.

In this class, students experienced two types of Dolittle programming: Dolittle programming and robot control programming. Fig. 3 shows the results of the respective questionnaire.

We correlated the above result with the students’ scholastic achievements in 5 major subjects (Japanese, Social Studies, Mathematics, Science, and English). The result indicated that low-achieving students tend to answer “Robot is more fun”, “Dolittle is more difficult” and “Robots made me interested in computers”.

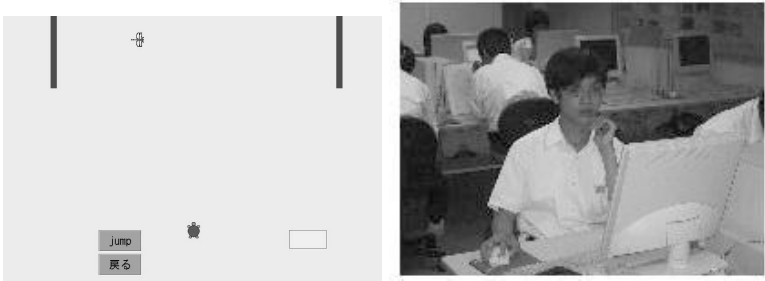


Fig. 8. Student and the programmed game

Table 3. Results of questionnaire (%)

question	Dolittle	Robot
Which is more fun?	57	43
Which is more difficult?	56	44
What make you interested in computers?	58	42

Because the style of robot control programming is a “trial and error” approach, students who have difficulties in learning can also understand easily. On the other hand, high-achieving students found robot programming boring. As this programming is not so complicated, the students can estimate the result of their programs before execution. Thus they tend to find intellectual interest in Dolittle programming. Therefore, we concluded that robot control programming is suitable and effective material for the introductory stage of programming.

7 Future Work

7.1 From Biaxial to Triaxial Robot as Teaching Material

Use of robots as a teaching material enabled us to conduct lessons which combine programming and robotic control. In these lessons, robots are biaxial and they only move around the field. Although the lessons were very successful, we consider that tasks are too simple to apply in the field of education. We need more extensible teaching material with more teachers can conduct lessons according to their original ideas. Thus we are planning to implement a triaxial robot control lesson. In this lesson, students establish a goal to accomplish certain work such as carrying some objects from one point to another by using a robot arm with the third motor. We also think robots need to equip more sensors such as light sensor etc. If more sensors are provided with the robot, more flexible and complicated work can be attained. To realize this, we requested another expert to design and develop another controller board which can control three DC motors and install up to four sensors. Fig. 9 shows this controller board and a trial model of triaxial robotic car.

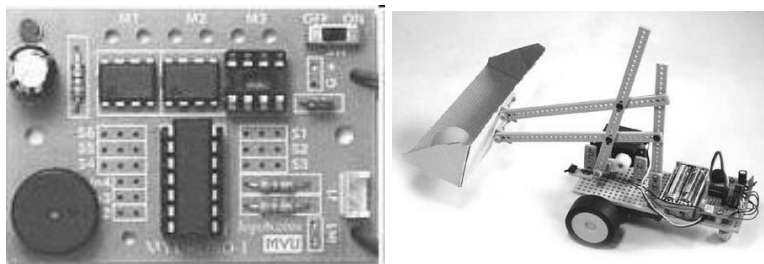


Fig. 9. Triaxial controller board and an example robot

By using this controller board, students can work on more practical problems than conventional biaxial robotic car control. Furthermore, triaxial robots have room to exercise ingenious mechanism of the arm. This enables school teachers to run robot contests to compete for superiority of both programming skill and sophistication of the mechanism. In most of the popular robot contests for junior high school students in Japan, robots are controlled by hand with a wired remote controller (switch board). In this type of contest, the effect of control skill is sometimes superior to that of the design of the robot. However, in creating an autonomous robot, student must consider the motion of the robot beforehand. This makes students think how their robots capture the information of the surrounding environment and process it. We think students will be able to enjoy solving problems throughout this process. Currently the experimental lessons of triaxial robot control are in progress. We are going to evaluate and analyze these lessons after finishing them.

7.2 Coherency Between Screen Objects and Real-World Robots

Two robots mentioned in this paper i.e. biaxial and triaxial robots run with programs downloaded from a PC to the on-board CPUs. These CPUs are not designed to execute complicated and long programs. The programs must be written as procedural, not object-oriented; this means the language syntax of robot control code is different from that of the Dolittle programming language. Consequently, there is a semantic gap between Dolittle and robot control code.

We are planning to improve both Dolittle and the firmware of the controller board so that consistency in language syntax can be achieved between screen objects like turtles and robots in the real-world. With this, we hope students will be able to learn the concept of object-oriented programming not only from Dolittle but also robot control. We also hope this will be an unprecedented teaching material for learning programming.

8 Related Works

The endeavors to utilize autonomous robots for school education have been studied since 1990s[4]. The most innovative work in this area is Programmable

Bricks[5]. It was developed from experiences of the autonomous robot contest at MIT. It is presented as a large LEGO block, and it is combined with the body and other mechanisms built with conventional LEGO parts to make an autonomous robot. It was commercialized as LEGO MindStorms by the LEGO Group and has been used widely in schools. In LEGO MindStorms, a unit compatible with Programmable Brick is called RCX. It is equipped with 3 inputs for sensors, 3 outputs for motors, simple LCD displays and an infrared ray interface. RCX programming is conducted in RCX Code or ROBOLAB. RCX Code is a visual programming environment which was improved from LogoBlocks[6]. ROBOLAB is another visual programming environment based on LabView.

In lessons using LEGO MindStorms, students can construct robots easily with LEGO blocks. However, it is difficult to construct small and light weight robots. Furthermore, because RCX is a sealed package, students would not see or touch the electronic circuits or IC chips inside of it.

Several years later, Cricket[7] was developed by MIT. Cricket has two inputs for sensors, two outputs for motors, an infrared ray interface and two general bus ports. Programs can be written in Logo languages using Cricket Logo (text language) or LogoBlocks (visual language).

Cricket is also used for constructing autonomous robots. However, as the capacity of its battery is small and it has only two motors by default (without expansion with general bus devices), it is not suitable for complex robots.

RCX and Cricket have in common with our controller boards that they are used for building robots, but there is one notable difference. Most of RCX educators are interested in the type of robots they create. Cricket is often used in a wider range of use of education such as science. Our objective is that students understand computer through constructing and programming of robots. We adopted the Dolittle programming language and students write program by text and input. Because the syntax of Dolittle is simple and it allows to express programs in local languages, students can prepare programs without difficulty. Additionally, because our robots are simple, even controller boards can be constructed by soldering. We designed robots that are simple and not expensive so that students can buy and bring them to their home.

9 Conclusion

Learning programming by using Dolittle and robot control has been introduced. Ideal learning which students think “it is hard but fun” can be realized by incorporating robot control into learning programming. Furthermore, learning that links the program implemented inside the computer with the real motion of a robot was made possible. Self-produced robots induced a sense of accomplishment in students learning programming.

In addition, students good at learning enjoyed learning programming with Dolittle and tended to have interest in a computer. Low-performing students found pleasure of learning programming with robot control and tended to have interest in a computer. Thus we are convinced of the effect to strengthen motivation by introducing robots to learning programming.

Acknowledgement

We would like to thank Mr. Masayoshi Okada and Mr. Shuji Inoue for their kind cooperation to develop controller boards for autonomous robots we use. We also deeply appreciate the contribution of the teachers and the students of Nishimashizu Junior High School.

References

1. Kanemune, S. and Kuno, Y. : Dolittle : An Object-Oriented Language for K12 Education. *EuroLogo2005*, Warszawa, Poland, pp. 144–153, 2005.
2. Kanemune, S., Nakatani, T., Mitarai, R., Fukui, S. and Kuno, Y. : Dolittle — Experiences in Teaching Programming at K12 Schools. *Proc. of the Second International Conference on Creating, Connecting and Collaborating through Computing (C5)*, IEEE, pp. 177–184, Kyoto, Japan, 2004.
3. Dolittle Programming Language. <http://dolittle.eplang.jp/>
4. Resnick, M. : Behavior Construction Kits. *Communications of the ACM*, Vol. 36, No. 7, pp. 64–71, 1993.
5. Resnick, M., Martin, F., Sargent, R. and Silverman, B. : Programmable Bricks : Toys to think with. *IBM Systems Journal*, Vol. 35, No. 3–4, pp. 443–452, 1996.
6. Begel, A. : LogoBlocks : A Graphical Programming Language for Interacting with the World. Advanced Undergraduate Project, MIT, 1996.
7. Martin, F., Mikhak B., Resnick, M., Silverman B. and Berg, R. : To Mindstorms and Beyond : Evolution of a Construction Kit for Magical Machines. *Robots for Kids*, Morgan Kaufman, 2000.

A Master Class Software Engineering for Secondary Education

Tom Verhoeff

Technische Universiteit Eindhoven,
Department of Mathematics and Computer Science,
Den Dolech 2, 5612 AZ, Eindhoven, The Netherlands
`T.Verhoeff@TUE.NL`

Abstract. We explain why it is interesting and useful to organize master classes in software engineering for pupils from secondary education. We then describe the format and technical content of our master class. Finally, we present our experiences from organizing several such master classes and conclude with some recommendations.

1 Introduction

Informatics is still not a mature topic in the secondary education of most countries. Informatics curricula, teacher training, textbooks, and exam requirements have not stabilized, are the subject of extensive discussions, and are in need of further development (see e.g. [9]). The situation is certainly improving, as witnessed by the founding of the Computer Science Teachers Association [5] and their K–12 curriculum model [1].

Besides the incorporation of informatics education in the regular curriculum, there have been numerous other efforts to provide pupils in secondary education with an opportunity to become acquainted with (some parts of) informatics. Among these are competitions like the International Olympiad in Informatics (IOI) [8] since 1989 and [2] since 2004 (see also [6]) at national, regional, and international levels. Many enthusiastic teachers and universities have organized special projects for secondary education involving informatics. At the Technische Universiteit Eindhoven (TU/e, the Netherlands), we have organized for several years a master class in software engineering for interested pupils from secondary education.

1.1 Overview

In Section 2, we give our motivation for organizing the master classes. Section 3 explains the format and technical content of the master classes. In Section 4, we summarize our experiences from organizing several such master classes. Section 5 concludes with some recommendations.

2 Motivation

The IOI has stimulated the creation of many national olympiads and has thereby offered numerous pupils an opportunity to delve into the algorithmic aspects of

informatics. Over the years, the level of the IOI has steadily risen. Currently, the tasks posed at the IOI are even challenging for graduate students at universities (see [7] for an example problem and its analysis).

As a consequence, instead of being an attractive low-threshold introduction to informatics, the IOI has become an awe-inspiring competition that is accessible to a relatively small group only. This also has meant that national olympiads need to be quite selective. Furthermore, these olympiads offer a narrow view of informatics because they focus exclusively on individual solving of quite hard algorithmic problems.

In 2001, the author participated in an experimental software architecture training activity for industry. One of the cases involved control software for a simple (simulated) elevator system. There, we got the idea that this could make an interesting topic for high school students, because the control software hardly has any algorithmic depth. By working from a predefined architecture, it is possible to create the control software with a group of ten to twelve inexperienced programmers in a day or so.

Thus was born the idea of a master class software engineering, which would focus on

- informatics from an engineering point of view,
- constructing software with a team,
- architecture rather than algorithms.

It turns out that the prerequisite knowledge for developing the control software is rather limited. The participants should have elementary knowledge of a programming language¹. In particular, it is important to have an understanding of global versus local variables and routines with parameters, how to use (call) and how to implement such routines.

3 Master Class: Developing Elevator Control Software

The concept of a master class has been around for a long time. It is, for instance, popular in the Arts (music, painting, and drama). In a master class, a small group of participants receives a lesson from an expert. The aim is to teach the participants the ‘tricks of the trade’, helping them advance on the professional ladder. The feedback and advice is given in front of the entire class, but is personalized based on individual performance.

The *Master Class Software Engineering* that TU/e offers has the following format:

- **Day 0** Half a day preparation at school, to ensure that the basic programming knowledge is adequate.
- **Day 1** Half a day at the university, to present some theoretical background information, to discuss the requirements and architecture of the case, and to form subgroups.

¹ In the industrial training, this was C, but we switched to Pascal under Borland Delphi.

- **Day 2** One full day at the university, to design and implement the modules in the subgroups, to verify and integrate them, to do a final acceptance test, and to conclude the project.

Our master classes have had from 10 to 12 students, who are accompanied by a few teachers (see also Section 4). We can accommodate up to 24 students. Below, we give some more details of the topics that are addressed in the master class itself. The introduction at school reviews their knowledge of an imperative programming language.

The introduction is done interactively with the group. Typically, with some help they can formulate a definition of Software Engineering along the following lines:

- Deliver a software product (i.e. not for personal use only)
- of specified (high) quality
- with limited resources (time, money, developers)
- according to accepted engineering principles.

We then pay attention to the notion of software quality, not only the obvious external qualities of functional correctness and efficiency, but also dealing with internal qualities as verifiability, maintainability, and reusability. They typically can state the major phases in the life cycle of a (software) product, from initial idea and problem statement, via requirements engineering, design, coding, testing, to deployment, operation, maintenance, and retirement.

Project management is touched, but only lightly. In fact, we do most of the *planning* in advance, and we drive the preparation of the role distribution and work-breakdown schedule. *Configuration management* is important, and we give them a simple scheme for dealing with multiple revisions. *Verification* is integrated in the workflow, and the accompanying teachers have a role in this as well. *Monitoring* of the process is again done by us, without bothering them too much with the details. It is emphasized that the quality of their working process is an important ingredient for the quality of the final product. When deficiencies in a product are discovered, we try to trace them all the way to their (or our) working process.

The main focus of our master class is on

- constructing quality software with a team (working in parallel),
- software architecture (in particular, its role and use),
- verification (through reviews and systematic testing),
- maintainability (e.g. well-structured code, coding standard).

Our master class is based on the case of developing control software for a simple elevator system (see Fig. 1). This system was initially designed and implemented by Björn Bon at Alert Automation Services to illustrate the classical notions of software architecture; see also [3]. We ported it from C to Delphi Pascal.

The problem context is shown in Fig. 2. The product to be developed is the elevator control software in the double-framed box. It drives the (simulated)

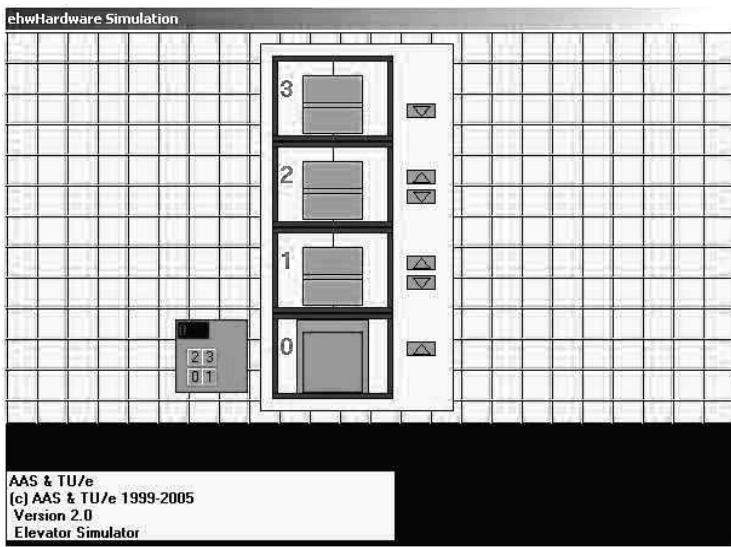


Fig. 1. Screenshot of simulated elevator system

elevator hardware, which in turn interacts with the elevator passengers. Note that the elevator passengers do not directly interact with the software. The ‘behavior rules’ represent the constraints on the hardware that must be realized by the control software.

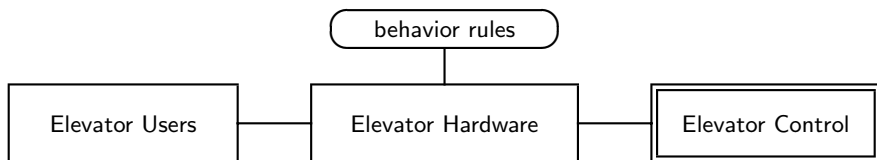


Fig. 2. Context diagram for elevator system

The (simulated) elevator hardware consists of

- a single *cage* in a vertical shaft with 4 floors,
- *doors* on each floor and in the cage (separately controlled),
- *buttons with lights* on each floor (up/down) and in the cage (0 through 3),
- a 3-character *display* in the cage to show the position.

In the initial state, the cage is stationary at floor 0, with all doors closed, all button lights off, and an empty display.

The hardware–software interface (EHW) is available in the form of a few constants (number of floors, etc.), type names, and routines for various operations:

- Buttons: detect a request, switch light on/off
- Doors: get current position, set speed open/close, stop

- Cage: get current position, set speed up/down, stop
- Display: set text
- Debug: print debug line, emergency stop with message

The elevator control software (ECT) is ‘clocked’, that is, every clock tick (20 times per second) the procedure `ectProcess` is called. It should be noted that the hardware is completely ‘unprotected’. The software can do whatever it pleases, including actions that could be unsafe for passengers or the hardware. Of course, good control software must not misbehave.

Day 1

During Day 1, the participants are encouraged to experiment with the interface. For instance, to make the cage go through the roof with open doors, etc. Next, we interactively clarify the requirements, such as:

- all passenger requests must be handled appropriately,
- all actions must be safe for passengers and hardware (e.g., doors open only on a floor with a stationary cage),
- user friendly (e.g. gentle acceleration), fair, fast, ...

Of course, further details are specified. These provide the ingredients for the acceptance test.

It soon becomes clear to them, that it will be quite hard to fulfill all requirements with software that has the same form as their earlier ad-hoc experimental software. The technique of *divide and conquer* must be applied to bridle the complexity. We briefly discuss the notion of *software architecture* as a description of the structure of a software solution (components and their relationships), and various views of such an architecture (abstract static, abstract dynamic, code files, code execution). The main advantages of software architecture are that it facilitates

- concurrent development and verification of components in isolation;
- stepwise integration of components;
- piecewise improvement by changing one or two components at a time.

Each component of the elevator control software is placed in a separate Pascal **unit**, with a specified **interface** and specified **uses**-relations to other components. The architecture has a great impact on internal qualities such as verifiability, maintainability, and reusability. We look at such things as the number of components, their size, coupling and cohesion, interface complexity, fan-in/out. The students are challenged to propose their own architecture for the elevator control software. This turns out to be harder for them than they expect and it offers interesting topics for discussion.

One of the key decisions in our master class concerns the architecture. Even if the group comes up with a good proposal for an architecture, we still choose to use our own architecture (see Fig. 3). This has several advantages:

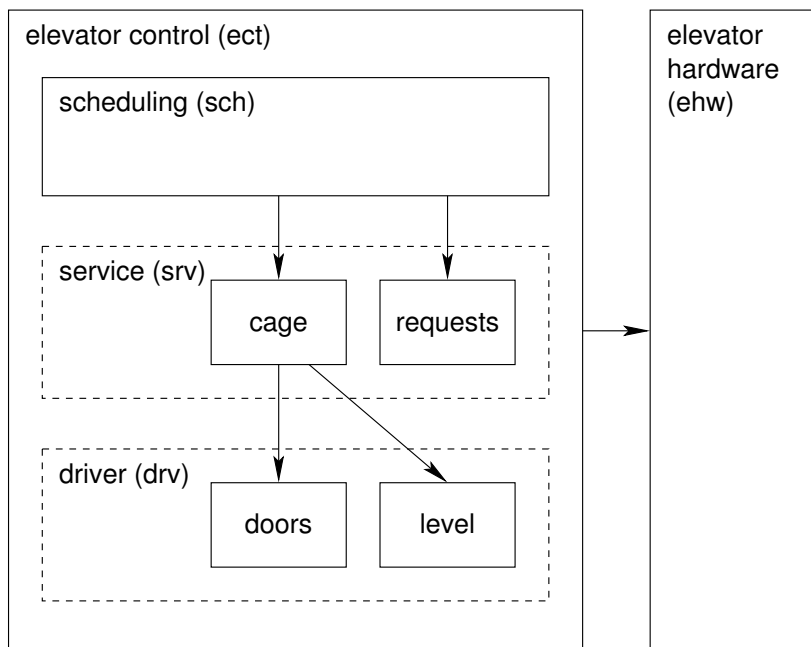


Fig. 3. Software architecture for elevator control software

- we know the limitations² and challenges of this architecture quite well;
- we have empty code frames for each component;
- we have a reference implementation for each component;
- we have a testing frame for each component.

Our architecture consists of three layers. At the *lowest level* are two independent driver components, one for the doors and one for the vertical cage position (level). The *middle layer* has two service components, one for coordinating the door and cage movement, and the other for recording button requests and controlling the button lights. At the *top* is a single scheduling component, which uses the requests component for ‘input’, and the cage component for ‘output’.

The participants get a complete specification of the interfaces of these components on Day 1 (space limitation prohibit the presentation of more details here). At the end of Day 1, we discuss the issues surrounding (human) errors: their inevitability, their cost, how to avoid them, or at least discover and eliminate them early.

We finish Day 1 by forming five groups, one for each component. Typically, a group consists of two students, but the scheduler and, if necessary, the requests component could use an extra member. Thus, we have work for 10 to 12 participants.

² Some limitations are included on purpose.

Day 2

On Day 2, each group makes a detailed design of their assigned component, reviews the design (with a teacher), produces the corresponding code, reviews the code (with a teacher), prepares some test cases with related drivers and stubs (where necessary), unit tests the code, delivers the code. The component designs can be expressed in terms of *state diagrams* with conditional state transitions. Assistants monitor the work and provide instant feedback on the practices of each group.

We go through two *iterations*. The first iteration is aimed at a simple and safe solution, possibly omitting some less important requirements. For instance, the `scheduler` could simply oscillate between floor 0 and floor 3 stopping at every floor, and the `level` component could move the cage at the slowest speed only (not worrying about a speed profile). The second iteration is intended to improve the initial solution by taking the remaining requirements into account.

Once the unit code is delivered, we subject it to our own tests, and report any problems encountered. This often results in some rework, and a discussion of how the problem could have been avoided, or how they could have found it by themselves.

In the afternoon, we integrate the components. We first do this in a *big bang* to illustrate the dangers of this. In all master classes, we ended up with a situation where the system fails and one group blames another without any effective way to resolve the issue. Stepwise integration provides more insight. For example, one could first integrate the components `cage`, `doors`, and `level`.

Finally, an acceptance test is done on the fully integrated system. In conclusion, we look back at the project and the lessons learned. We emphasize

- that the specific methods applied in the master class are not the only ones available; in fact, there are many alternatives;
- that the methods are especially suitable for larger projects; there is a danger that application in a small project creates the (false) impression that the methods are ‘overkill’.

4 Experiences

We have organized this master class six times: in 2001, 2003 (twice), 2004 (twice), and 2005. The first time was with pupils from the upper level of a single high school. Later, they came from two or three different high schools. On all occasions, we had no more than 12 participants per master class. We could have handled up to 24, by duplicating some or all components. That would have been interesting, because there would be competition and an opportunity to compare solutions. More than about 24 participants is unrealistic for a master class, which requires closer interaction.

The master classes were quite successful, in the sense that the final products were of better quality than we had initially expected, and also that the participants enjoyed the project very much and indicated that it helped them to form a better opinion about informatics.

To our surprise, the prior programming knowledge of participants was not so important. About half of the participants did not have a working experience with the programming language Pascal, though they had used (Visual) Basic, C, C++, or Java. A few participants did not have any programming experience to speak of and they were able to pick this up rapidly. We also had a few participants who had competed in the informatics olympiad, and who could be said to be very experienced. However, their work within the group often led to problems (either social, or because of hard to find defects in overly complex code).

We have deliberately chosen to avoid any *object-oriented* programming, because of the conceptual complexity. The features of the programming language Pascal that are relevant in the master class can be summarized on a single page. The algorithmic content is minimal (only the **requests** component requires the use of an array and simple loops, and the **scheduler** requires some algorithmic reasoning). The main difficulty to be overcome is the clock driven nature of the control software and how to reason about the resulting parallelism, because it is somewhat counterintuitive.

It has been useful to give teachers a role in the master class. They often feel uncomfortable because they cannot oversee the project from the start, and judge themselves incapable of developing the code. A role in the verification trajectory (design and code reviews) is much less unsettling, and allows them to reflect on their pupils work.

The software development process employed in the master class has certain limitations. Consequently, the end products are not of high quality. In order to improve the quality, we feel it would be necessary to take design and testing even more seriously. However, this requires extra work and adds complexity (e.g. in the form of more advanced drivers and stubs), which is not feasible in the available time.

From an organizational point of view, it would be nicer to offer a one-day master class (as for instance our mathematics department does). But we felt that the reductions necessary to accomplish this would not allow us to cover enough of software engineering and still provide them with the satisfaction of having produced a significant piece of software.

5 Conclusion

We have described in some detail the organization and contents of a Master Class Software Engineering that TU/e offers to senior pupils in high school. In the master class, a group of fairly unexperienced programmers designs and implements the control software for a (simulated) elevator system. None of the participants by themselves would have been able to produce the software in time, but as a team they succeed.

This master class contrasts sharply with the kind of informatics that plays a role in the IOI (International Olympiad in Informatics) and its national and regional variants. The success of the master class shows that it fulfills a need.

One of the big dilemmas we encounter is aptly expressed by [4]:

Marco Polo describes a bridge, stone by stone.

“But which is the stone that supports the bridge?” Kublai Khan asks.

“The bridge is not supported by one stone or another,” Marco answers, “but by the line of the arch that they form.”

Kublai Khan remains silent, reflecting. Then he adds: “Why do you speak to me of the stones? It is only the arch that matters to me.”

Polo answers: “Without stones there is no arch.”

The “stones” are the many details that are important for running a software engineering project, including the syntax and semantics of a programming language, how to operate programming tools, how to divide a large program into components, how to express a design, how to deal with errors. But none of them alone can carry the full weight. Only a balanced combination (“arch”) leads to success. The master class is long enough to touch the stones yet short enough to allow one to maintain an overview of the arch.

In our opinion, even if informatics becomes a regular topic in the curriculum of secondary education, it would be good to offer such master classes.

Acknowledgment

The author would like to thank Ria van Ouwerkerk, Desirée Meijers, Marcel Koonen, and Erik Scheffers for their contributions in preparing and executing the master classes described in this article.

References

1. ACM K–12 Task Force Curriculum Committee. *A Model Curriculum for K–12 Computer Science: Final Report*. October 2003.
<http://www1.acm.org/education/k12/k12final1022.pdf>
2. Beaver IT Contest
<http://www.bebras.lt/> (accessed March 2006).
3. Bon, B. “Software Architecture: Standing on the Shoulders of Giants”, XOOTIC Magazine, Post-Masters Programme in Technological Design, Technische Universiteit Eindhoven, The Netherlands, pp. 5–11, March 2000.
<http://www.win.tue.nl/xootic/magazine/mar-2000/bjornbon.pdf> (accessed March 2006).
4. Calvino, I. *Invisible Cities*. Harcourt Brace Jovanovich, 1974.
5. Computer Science Teachers Association. <http://csta.acm.org/> (accessed March 2006).
6. Dagiene, V. “Information Technology Contests: Introduction to Computer Science in an Attractive Way”. *Informatics in Education*, vol. 5, no. 1, 37–46, (2006).
7. Horváth, G. and Verhoeff, T. “Finding the Median under IOI Conditions”, *Informatics in Education*, 1(1):73–92 (2002).
8. IOI, *International Olympiad in Informatics*, Internet WWW-site
<http://www.IOInformatics.org/> (accessed February 2006).
9. Szlavi, P. and Zsakó, L. “Delusions in informatics education”, *Teaching Mathematics and Computer Science*, University of Debrecen, Hungary. 2(1):151–161 (2004).

Algorithmic Thinking: The Key for Understanding Computer Science

Gerald Futschek

Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstrasse 9, 1040 Vienna, Austria
futschek@ifs.tuwien.ac.at

Abstract. We show that algorithmic thinking is a key ability in informatics that can be developed independently from learning programming. For this purpose we use problems that are not easy to solve but have an easily understandable problem definition. A proper visualization of these problems can help to understand the basic concepts connected with algorithms: correctness, termination, efficiency, determinism, parallelism, etc. The presented examples were used by the author in a pre-university course, they may also be used in secondary schools to help understanding some concepts of computer science.

1 Introduction

In autumn 2005 the Faculty of Informatics at the Vienna University of Technology started to offer a pre-university course [1] (propaedeutic course), similar to other universities, e.g. [2], for all applicants who intended to start one of the bachelor studies in Informatics. This course addresses all beginners and has many-fold reasons:

1. Lack of pre-knowledge, what is Informatics after all
2. Lack of pre-knowledge, how computers work
3. Lack of pre-knowledge about algorithms
4. Lack of pre-knowledge about programming
5. Lack of sufficient knowledge in mathematics

These facts were observed by our lecturers and were confirmed by a survey of our beginners. Although these topics are part of the secondary school curriculum and should be known by all students passing secondary school, most of our beginners have not enough skills and pre-knowledge that is necessary to start a university study in Computer Science. The consequences were a very high drop out rate during the first study year and a low success rate in the topics Programming and Algorithms & Data Structures. The pre-university course should overcome the lack of usual pre-knowledge. Therefore the contents of this propaedeutic course are:

- What is informatics?
- How do computers work?
- Algorithmic thinking
- First steps in programming
- Basics in mathematics

In this contribution we focus on the part on algorithmic thinking and try to answer the question: How can students that cannot program, learn basic facts about algorithms in a very short period of time?

2 What Is Algorithmic Thinking?

Algorithms are defined differently in literature, but for our purpose the following definition is sufficient: “An Algorithm is a method to solve a problem that consists of exactly defined instructions”. Algorithmic thinking is a term that is used very often as one of the most important competences that can be achieved by education in Informatics [3]. Algorithmic thinking is somehow a pool of abilities that are connected to constructing and understanding algorithms:

- the ability to analyze given problems
- the ability to specify a problem precisely
- the ability to find the basic actions that are adequate to the given problem
- the ability to construct a correct algorithm to a given problem using the basic actions
- the ability to think about all possible special and normal cases of a problem
- the ability to improve the efficiency of an algorithm

For me, algorithmic thinking has a strong creative aspect: the construction of new algorithms that solve given problems. If someone wants to do this he needs the ability of algorithmic thinking.

3 How to Teach Algorithmic Thinking?

This question is as hard to answer as “How to teach creativity?” A practical answer can be: try to solve many problems. Especially for beginners in Informatics these problems should be chosen very carefully. In my opinion these problems should be solvable independent from a specific programming language. Especially beginners have many problems to understand the underlying programming language properly, so that they cannot concentrate additionally on the design of a new algorithm. The language to describe the algorithm should be high-level and problem-oriented, e.g. pseudo code fulfills these criteria.

The problems to be solved should be not too simple, but the problem statement should be easily understandable. More complex problems give more space to creativity and to the students individual, sometimes unusual solutions. As is shown in the following examples pre-knowledge of a programming language is not necessary.

Very useful and powerful tools are visualizations of algorithms, e.g. [4]. These tools help to understand algorithms. The possibility to construct easily different input values allows to play with different variations of inputs and allows to study normal and extreme cases. It gives also a feeling why an algorithm works and how an algorithm may be improved. We used in our course a tool, called Theseus, see [1], written by the student Marian Kogler. This tool is able to produce and manipulate mazes and it allows to apply different path-searching algorithms.

4 Example: Paths in Mazes

We want to show with this example that it is possible to gain first insight in problem analysis, algorithm design and effort analysis without prior knowledge of computer programming.

4.1 Problem Analysis

Finding a path in a maze (labyrinth) is a classical task and is not trivial. Usual tasks are:

*Find a path out of a maze,
Find a path through a maze, or
Find a path to a specific position inside a maze.*

What is to do seems easily understandable, but it is not evident how to do it. A first step in solving the tasks is the analysis of the problem. We can find out that the mentioned tasks have a common generalization:

Find a path from position A to position B

It is also important to know what we know about the maze while we search for position B. In the simplest scenario we know nothing about size and structure of the maze. We can walk on corridors and we can see until the next crossing. At a crossing we can see all the different corridors starting from this crossing. We can recognize the goal B, but we do not know the direction to the goal B. If we reach a crossing that looks like a crossing already visited, we cannot decide if it is the same crossing or a new one.

While walking in a maze, we have to decide at each crossing which corridor to go. Possible strategies are:

*Follow the leftmost corridor
Follow a random corridor
Follow systematically all corridors, beginning with the leftmost*

We now discuss these three strategies in detail.

4.2 A First Solution Strategy

A well-known strategy to find a path through a maze is:

`follow the left wall`

It is not necessarily clear that we find a way through a maze with this strategy. In the case that the starting point A is an entrance of the maze it can be shown that this strategy finds an exit. Reasoning about this could be a good exercise in algorithmic thinking. It should also be considered the case that the maze has no exit at all.

Mazes can differ in size, form, and length of the corridors and crossings, mazes can be planar or even 3-dimensional. An appropriate model of a maze shows only its relevant characteristics.

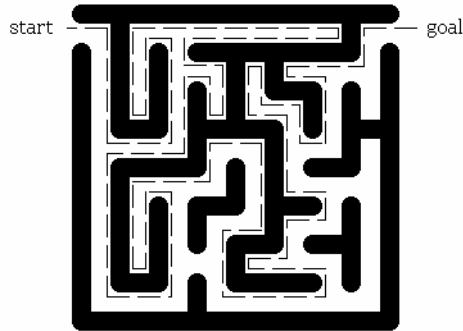


Fig. 1. The Left Wall Algorithm. After entering the maze we follow the left wall. This left wall is connected to (a part of) the surrounding wall. Therefore we will reach an exit of the maze using this strategy. If we exit at the entrance then we know that there exists no other exit in the maze.

4.3 A Model of a Maze

Modeling is abstraction. From a maze we can abstract the length and form of the corridors. The only thing we have to know is: which crossings are connected with which other crossings. An adequate model of a maze is a graph, its nodes are the crossings and its edges are the corridors.

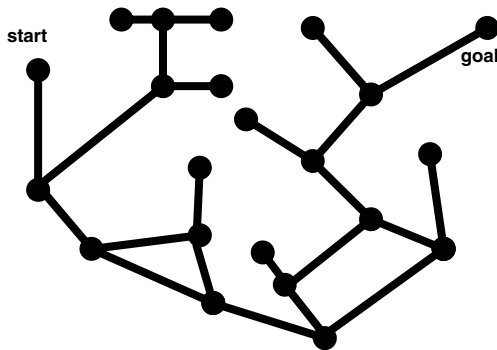


Fig. 2. The Maze of Fig. 1 represented as graph consisting of nodes (crossings) and edges (corridors). The length, form, and direction of the corridors are abstracted, they are not relevant in the graph. A graph just represents the neighborhood relation of crossings. Which pairs of crossings are connected by a corridor (an edge exists) and which pairs of crossings are not connected (no edge exists).

In the left wall algorithm we need to know for each node that can be reached which one of the edges around this node is the leftmost edge. In order to be able to define the leftmost edge of a node, we can define a circular ordering of the edges around each node. Each edge i around a node has in this ordering a succeeding edge $\text{succ}(i)$. When

reaching a node on an edge e , then the leftmost edge is $\text{succ}(e)$ according to this circular ordering. In a planar maze there exists a natural circular ordering of the edges according to their direction in the plane. Please note that the leftmost edge of a node is not a fixed edge, the leftmost edge depends on the edge e that reaches the node.

Since a normal graph as an abstract model of a maze does not represent a direction or ordering of edges we have to explicitly and additionally define the circular ordering of the edges around each node. A graph with such a circular ordering of edges is also known as “embedded graph”.

4.4 Basic Actions of a Maze-Algorithm

To describe algorithms we need to know the actions that are the elementary instructions that can be used. In this maze example we define the basic actions at a high level (much higher than that of programming languages). So we hope that the semantics of the algorithm will be easier to understand.

We define 4 basic actions that can be performed when a node (crossing) is reached by an edge e :

- action 1. Query: Is the goal reached?
- action 2. Determine the number of edges (corridors) that leave this node.
(If this number is 0, then a dead end is reached)
- action 3. Follow the i^{th} edge and go to the connected node (crossing).
(The 1^{st} edge is called the leftmost edge)
- action 4. (Turn around 180 degrees and) go back to previous node,
following edge e .

Using these 4 basic actions we will formulate the algorithms to find paths in mazes.

“Follow the i^{th} edge” in action 3 means: after entering a node on edge e the edge $\text{succ}(\dots \text{succ}(\text{succ}(e))\dots)$ has to be followed, where the circular ordering succ is applied i times. The 1^{st} edge is defined as the leftmost edge and can be determined by $\text{succ}(e)$. The number of basic actions can be reduced by one since basic action 4 can be expressed by basic action 3: Follow the 0^{th} edge, which is in fact edge e . We think that this special case of action 3 is easier understood in an extra basic action.

4.5 A First Algorithm

Using the 4 basic actions we can formulate the left wall strategy in the following pseudo code algorithm:

```

while goal not reached                                (action 1)
  if dead end                                           (action 2)
  then turn around (180 degrees) and
    return to previous node                            (action 4)
  else follow the leftmost edge                        (action 3)

```

For this algorithm we need to give a precondition that guarantees termination and the existence of the leftmost edge:

precondition: the goal is reachable (to guarantee termination) and there exists a starting edge (to be able to determine the leftmost edge at the first node)

This simple left wall algorithm shows already the algorithmic concept of *backtracking* while returning in dead ends. This algorithm is very simple because there are no additional tools necessary and there are also no visited nodes to remember during execution. While trying out this algorithm with several mazes and with several start- and end-positions, we find out that this algorithm sometimes does not terminate.

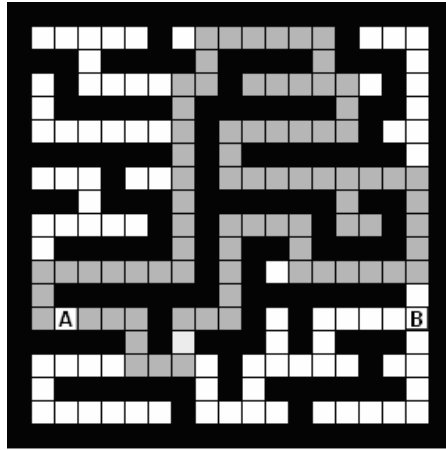


Fig. 3. If the maze contains cycles the algorithm may be trapped in a cyclic path forever, while following the leftmost path. The algorithm does not terminate in these cases.

The left wall strategy always terminates when it is applied to mazes without cycles and may not terminate when it is applied to mazes with cycles.

4.6 Random Paths

How walking in cycles can be avoided? One possible observation is: It seems that the left wall algorithm runs endless because at each crossing always the leftmost path is selected to follow. Why not choose randomly at all crossings a path to go? Now a tool to perform this random decision is needed, a dice or a random number generator. Using this tool at each crossing, an arbitrary path can be selected. Every time the algorithm starts from the same starting point again possibly other paths are used to find the goal. This is a property of non-deterministic algorithms. Sometimes the goal is reached with a small number of steps, sometimes it needs many steps to reach the goal. The question arises: Does this algorithm always terminate? Is it possible that by random algorithm always those paths are chosen that don't reach the goal? All experiments with different mazes and different start- and end-positions show that the algorithm always terminates when the goal is reachable at all. Is this a proof that the random path algorithm always terminates? No. These observations are not a proof. A proof needs some scientific insight into probability theory. But, anyway, it is important to raise such questions to motivate students to study the necessary theories.

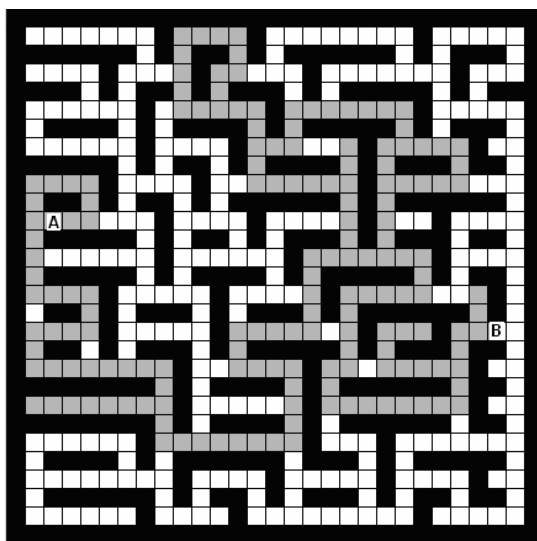


Fig. 4. Random Path Algorithm. A random walk does not search systematically. Each start of the algorithm generates another path. This algorithm is not deterministic. Each reachable position of the maze can be reached in a finite number of steps. Already visited positions may be visited many times again and again. There is no upper bound of the number of steps needed to reach an end-position.

4.7 Using Ariadne's Thread

Another classical tool to avoid cycles is the famous “Ariadne's Thread” that was used by Theseus to find and kill Minotaur in his maze at Knossos. It is the eldest known back-tracking algorithm. The thread shows the path from the start-position to the current position. The basic actions 3 and 4 have to be modified: while walking on new paths the thread is unwound, while back-tracking an unsuccessful path the thread is wound up again. A basic action to find Ariadne's thread has to be added. Every time Theseus finds his thread again he is sure that he found a cycle and returns to the last crossing in the same way as he does at dead ends. If Theseus chooses the leftmost path first, the pseudocode of Ariadne's Algorithm can be formulated very similar to the Left Wall Algorithm:

```

while goal not reached
  if (dead end) or (Ariadne's thread is found)
    then turn around (180 degrees) and return to previous node
  else follow the leftmost edge

```

Notice that the leftmost edge depends on the edge one is coming from. To find the goal the edges of a node are tried out in the sequence of the circular ordering. When returning from a not successful edge, the leftmost edge is the next edge in the circular ordering to be explored. While performing Ariadne's Thread Algorithm with several mazes and several start- and end-positions we can observe that this algorithm finds sometimes the end-position very fast. But sometimes it takes very long and sometimes

it takes extremely long (many hours). It can be observed that this algorithm tries very systematically all possible combination of edges. It can be discussed how many combinations of edges are necessary to find all paths from A to B. A rough estimation shows a combinatorial explosion depending on the degree of the nodes and the distance from the starting point. This fact is a good motivation to improve this algorithm ...

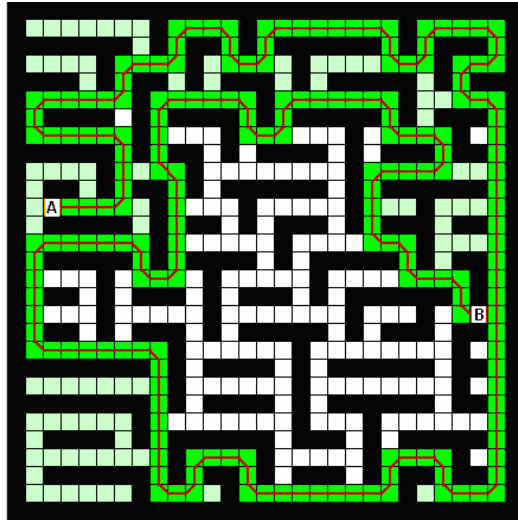


Fig. 5. Ariadne's Thread Algorithm. The thread avoids walking in cycles. The Algorithm produces systematically all possible acyclic paths from the start-position. It stops when the end-position is reached.

There are still many questions that can be discussed:

- What is the necessary length of the thread?
- What is the effect of the algorithm if the goal is not reachable?
- How we can improve this algorithm?
- How can we find the shortest paths?
- etc.

The maze-problem is an interesting starting point to discuss many variants of maze algorithms and is an ideal tool to give first insight in different aspects of algorithms.

5 Example: Parallel Sorting

Sorting of a series of values can be played by the students themselves. To be an active part of an algorithm is very motivating. Each student holds a value that can be seen by all the other students. At the beginning the students stand in a row at arbitrary positions. The task of finding a sorting algorithm consists of defining precise instructions to the students what they have to do. The students are told just to follow

the instructions of the algorithm and they are not allowed to do something else. The first approximation to a sorting algorithm is very often the following instruction to all students in the row:

```
repeat forever:
    exchange your position with a neighbor, whose
    value is not in correct order compared to your value
```

This first algorithm shows the idea of the algorithms very well, but it is not precise enough to be executed properly. There can raise the following problem: both neighbors of a student want to change with him. What should he do? Also the following question cannot be answered exactly at any time: Who is a neighbor? The values change their positions very often. If a neighbor starts changing his position with his other neighbor. Is he still a neighbor? Which one of the two exchanging values is the neighbor?

A more precise definition is necessary. We can solve the questions with *synchronization*: the two exchanging persons shake their right hands while exchanging their positions. The students are free for exchange if they have their right hand free. An improved algorithm with synchronization may look like this:

```
repeat forever:
    look at your left neighbor.
    if (he is free) and (his value is less than your value)
    then synchronize and exchange your position
        with this neighbor
```

This play of a sorting algorithm works very well with a group of students. This algorithm is also very robust: Even if a value is changed from outside or an additional value is inserted, this value will be rearranged until it is on its final position.

An analysis of this algorithm is also very easy: the maximum number of exchanges of a specific value is $n-1$. Because all persons act independently at their own personal speed, it is not possible to tell the number of steps of this parallel program. This is a perfect program to show the concept of *software agents*. But there is an unsolved point: as the parallel agents will act forever, it is not clear at what time the row is sorted!

We can observe that all odd positions can exchange at the same time or all even positions can do this at the same time. The odd and the even exchanges can be clocked so that they take place at the same time. Furthermore we need at most n parallel exchanges. With this observation we can formulate a terminating algorithm.

Parallel exchanges at odd and even positions take place alternately:

```
repeat  $n/2$  times:
    all even positions perform in parallel:
        look at your left neighbor
        if his value is less than your value
        then exchange your position with this neighbor
    all odd positions perform in parallel:
        look at your left neighbor
        if his value is less than your value
        then exchange your position with this neighbor
```

We know that we need at most n parallel actions, so the order of this sorting algorithm is linear. In fact this is the most efficient algorithm with the restriction that

all persons exchange with their neighbors. Studying computer science will show that there exist much faster parallel sorting algorithms, but all of them exchange positions over longer distance.

6 Summary

Our experience with students shows that a first introduction to algorithms can be learned at a problem oriented level independent from any programming language. A wide series of informatics relevant topics can be addressed and learned by solving interesting and encouraging problems. These problems should be well-chosen. A visualization of the algorithms in form of a program-tool or of an activity play performed by the students themselves can be very helpful. Beginners like to play with algorithms while feeding them with different input values to get an impression how the algorithms work and what problems arise, e.g. non-termination or combinatorial explosion. To improve or modify algorithms abstract models and precise definitions of properties are necessary.

References

1. Propädeutikum in Informatik (prolog), Lessons held at Vienna University of Technology: www.informatik.tuwien.ac.at/prolog
2. Loidl, S., Mühlbacher, J., Schauer, H.: Preparatory Knowledge: Propaedeutic in Informatics. In Mittermeir, R.T. (ed.): From Computer Literacy to Informatics Fundamentals. Lecture Notes in Computer Science, Vol. 3422. Springer-Verlag, Berlin Heidelberg New York (2005) 104-115
3. Snyder, L. Interview by F. Olsen "Computer Scientist Says all Students Should Learn to Think 'Algorithmically'," *The Chronicle of Higher Education*, May 5, 2000: <http://chronicle.com/free/2000/03/2000032201t.html/>
4. Mudner, T., Shakshuki, E.: A new Approach to Learning Algorithms. In Proceedings of International Conference on Information Technology: Coding and Computing. (2004) 141-145

Issues of Selecting a Programming Environment for a Programming Curriculum in General Education

Rimgaudas Laucius

Institute of Mathematics and Informatics
Akademijos str. 4, LT-08663 Vilnius, Lithuania
rimga@ktl.mii.lt

Abstract. The programming environment has an essential role in the curriculum of programming. This paper presents the main features determining an environment's methodical suitability for teaching programming. It is based on our recent experience of selection of environments when we had decided to replace the obsolete Turbo Pascal system in our schools. Our solution – to adapt the Free Pascal compiler for teaching purposes according to prearranged requirements may be treated as success story. The paper presents the main features of Free Pascal and the main tasks necessary for its adaptation: the development of an integrated development environment and the localization of compiler as well.

1 Origin of Issue

Our investigation is based on the practice of teaching programming in Lithuanian schools. There are two courses of programming in Lithuanian's general education curriculum: compulsory and optional (advanced). There are 16 hours assigned for a compulsory course that takes place in 9-10 grades. During this course students have to acquire basic programming concepts, data types and control structures. The main aim of this course is to develop students' understanding of what programming is, and help them to self-determine for continuation of studies in this field [3, 4, 7]. Students in the 11 and the 12 grades may choose advanced programming course (up to 34 hours) [8]. The content of this course is not clearly defined, but it is proposed to teach object oriented programming [2].

Essential parts of the environment which is necessary for teaching programming is a compiler and an editor. Our schools have been using the Turbo Pascal system for this purpose for a long time. But it has not been updated since 1992 and has out dated already. Its main deficiencies became clear:

- It is developed for DOS. It uses DOS keyboard drivers, code pages that cause trouble with Lithuanian character input and output. Textual user interface of DOS programs is very distinct from graphical user interface of Windows programs that already became usual for students. So they have to wait more time acquiring it;
- It is not localized and almost impossible to do that. This concerns also the programming language;
- It poorly implements the concepts of object oriented programming;

- It uses a 16-bit architecture. Hence, the size of data structures and of the program itself are too much limited (limited to 64 KB). That is relevant for solving problems designed for programming competitions.

Because of these and many other defects we decided to replace Turbo Pascal with a newer and better environment. Even though, there was no obviously best alternative at that time we considered two possible approaches of problem solution:

- Replacement with the newer Pascal environment;
- Replacement with the contemporary environment of another language.

After some discussions a solution was adopted to adapt for programming curricula the Free Pascal compiler, that was not well known at that time but promising. In the rest of this paper we will review criteria that determined the choice and highlighting the main features of the Free Pascal compiler and the tasks carried out for its adaptation.

2 Criteria of Selection of Programming Environment

Discussions concerning programming environment selection still emerge. So it is worthwhile to review the main criteria that determined our choice in more detail. That is:

- Methodical suitability;
- Cost.

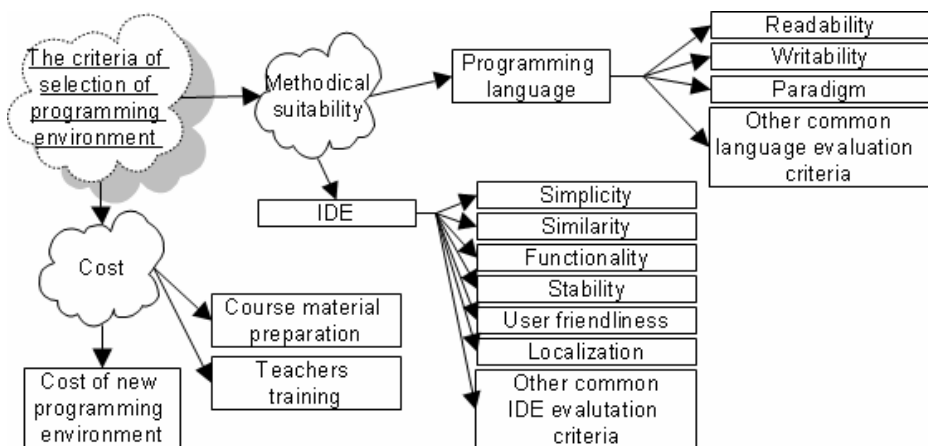


Fig. 1. The criteria of selection of a programming environment

2.1 Methodical Suitability

Readability. The main factors determining methodical suitability of a programming language are common programming language evaluation criteria [9, 15, 17, 18]. Main attention has to be paid to readability and writing convenience of programs.

Readability of programs is closely related to the form of its constructions (syntax). There are three aspects of the syntax that we want to emphasize: 1) the form of identifiers 2) the meaningfulness, consistency and convenience of reserved words 3) clear relation between syntax and semantics. The closer reserved words, identifiers and other programming language constructs are to natural language, the more comprehensible and readable are programs, and hence a programming language itself is simpler and easier to acquire.

A programming language is hard to learn when it is complex. Hence, a language that is not overloaded with constructs that are reasonable and have less syntactic homonyms and synonyms is simpler and better. **Simplicity** of a programming language is also influenced by **orthogonality**. Orthogonality provides the language with features of creation of the new language constructions combining existing constructions in all possible ways. The more orthogonal the language is the fewer exceptions it has. Hence it is simpler. Looking from the other side, too much orthogonality can give too much freedom for creation of new constructions. Hence, orthogonality has to be reasonably limited seeking to prohibit creation of fuzzy and even confusing constructions.

Writing convenience. Program writing convenience also depends on a programming language's simplicity and orthogonality. It depends on its features of **abstraction** and **expressivity**. Abstraction allows managing of complex data or control structures without going deep into their insides or details of implementation. Expressive language has to cover all essential semantic and syntax constructs.

An important feature of programming language that influences readability and writing convenience of programs is **internationalization**. The first level of internationalization of a programming language allows the usage of national characters in identifiers [12]. Some of contemporary programming systems already have implemented such possibility (Visual Studio 2003, Delphi 2005). The specifications for implementation of internationalized identifiers described in Unicode standard annex (UAX #31: Identifier and Pattern Syntax). The second level of internationalization of programming languages allows localization of lexical elements (reserved words, identifiers of standard functions and data types and etc.). Localized languages are more natural and easier to learn. Lexical elements written in vernacular are more comprehensible and mnemonic.

Programming paradigm. The two most popular programming paradigms currently taught over the world are structured and object oriented programming [6].

Object oriented programming has been established as the main ideology and technology of software development for the last couple of years. Distinctly from structured programming object oriented programming treats programs as composition of relatively independent parts – objects. These objects can be easily reused in other programs. Hence they allow for a wide library of objects, one does not need to write program from the start, but can combine them from existing objects. That is one of the most important and effectively exploited advantages of the object oriented programming for creation of large programs. But it is hard to reveal these advantages during rather short lessons in school. Lecturers face this even in higher level of education institutions [9].

Most of the troubles arise when tools purposed for professional programming that are not suited for teaching at all are nevertheless used for that purpose. In order to

adapt these tools for teaching purposes they have to be at least supplemented with methodically developed libraries of simple, visual, interactive and attractive objects. For example these objects could be designed for modeling of phenomena of real world in virtual environment [1, 5].

Many agree that teaching of object oriented programming is necessary. The question is how to teach it. Some recommend starting from structured programming at first, others to start straight from the objects [11, 13, 14]. The structure and aims of programming curriculum in Lithuanian schools determines that it is mostly reasonable to teach structured programming in compulsory course and object oriented programming in optional course. Hence, seeking to keep consistency between these courses, hybrid paradigm fit best to our needs.

Programming environment. The methodical suitability of a programming environment is determined by these most important criteria: **simplicity** – it has to be simple to use and acquire, not overloaded with features that are not useful in a programming course; **consistency** – it has to have graphical user interface, it has to be familiar and intuitive for students that have already used similar programs; **functional** – it has to have all functional features necessary for the programming courses including step-by-step debugging, watching values of expressions and etc; **stability** – it has to be completed and have no defects; **user friendliness** – it has to have a clear, easily accessible help system, notice errors made by users, to prevent erroneous actions and possible loss of data; **localization** – it has to be localized or internationalized so that it may be easily localized.

2.2 Cost

The amount of charges mostly depends on fact: are we switching to a newer environment of the Pascal language or to another language? Bigger changes require bigger investment. It includes preparation of methodical material (textbooks, online material and etc.) and training of teachers. It is necessary to assess not only charges but also other factors related to implementation of novelties too – e.g. how teachers and students will meet novelties.

There is a wide assortment of software that can be applied for teaching programming. It is worth to consider freeware tools too, e.g. open source software. Research on the use of open source software in education [14] accomplished in 2004 in Lithuania revealed that the use of open source software brings not only economical profit – to localize, propagate and support open source software in many cases is cheaper than to buy commercial software – but also pedagogical benefits. Maintenance of open source software adaptation and localization projects are able to ensure high quality and methodical suitability. This is in contrast to commercial software where support is under developers' decision.

3 Features of Free Pascal

Free Pascal is an open source compiler available for different processors (Intel x86, Amd64/x86 64, PowerPC, Sparc) and operating systems (Linux, FreeBSD, Mac OS Win32, OS/2). The Free Pascal language syntax is compatible with Turbo Pascal 7.0 as well as with most versions of Delphi.

Many authors comparing Pascal with other languages note that it poorly represents ideas of object oriented programming. This is true as long as one references to obsolete Pascal language standards (ISO 7185, ISO 10206). Meanwhile contemporary Pascal language dialects (including Free Pascal) are oriented into Delphi Object Pascal dialect that considering its features does not lose to such well known languages as C#, C++ or Java.

Free Pascal is modern and perfectly represents ideas of object oriented programming. It has interfaces, exceptions, classes, functions' overloading and many other features appropriate to contemporary languages. Hence, after supplementation with libraries of objects methodically suitable for teaching programming it may be successfully used not only in compulsory but in optional programming course as well.

Specialists state that Pascal because of its simplicity, readability, clearness, consistency and many other positive characteristics is more suitable for teaching than many other popular languages used in education (Java, C++, Visual Basic [6])[18]. Pascal has a well-balanced set of data types, their construction instrumentality and control structures. Because of that it perfectly suites for programming calculation and data processing tasks. Very little percent of students selects programmer's specialty after school but ability to solve algorithmically this kind of tasks can be applied in a wide range.

Free Pascal language is compatible with Turbo Pascal. Hence its application for teaching programming in compulsory course will cost minimum investment, because existing textbooks and other material almost do not need to be changed. Neither does teachers' training require changes.

Free Pascal has already become widely used at this time. It has been used in International World and Baltic Olympiads in Informatics for a couple last years. This compiler can be used not only for teaching but also for professional programming. That is illustrated by programs developed using Free Pascal: Pixel – image editing program which beat even programs like GIMP, PaintShop Pro; Lazarus – class libraries and IDE that emulate Delphi.

Even though the Free Pascal compiler does not meet some of the criteria mentioned it has to be additionally adapted before using it in school curricula. The main tasks are the following:

- to develop an integrated development environment for Free Pascal compiler;
- to localize the Free Pascal compiler.

4 Integrated Development Environment for Free Pascal

The main problem that we had faced assessing Free Pascal was the absence of an integrated development environment (IDE) that could be suitable for our programming curriculum. FP IDE distributed with compiler run using textual mode and was not stable. So we had decided to develop the IDE that would meet all requirements by ourselves.

We have developed a graphical (Fig. 2) programming environment and integrated it with additionally adapted Free Pascal compiler into one package named FPS. Because of that it is easier to install and to use.

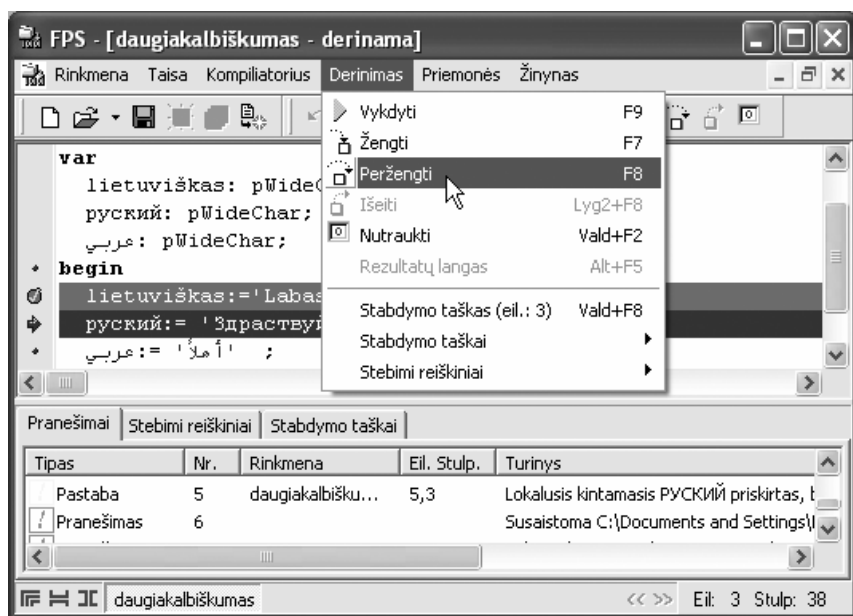


Fig. 2. FPS main window

Though not all criteria have been totally satisfied yet the new environment has already been applied in schools' programming curriculum.

Main features of FPS:

- It is freeware. It may be downloaded from <http://ims.mii.lt/fps>. More information about it may be found there too;
- It is easy to install and work may start immediately after installation. It does not require any additional setup;
- It has a graphical user interface that is similar to other Windows programs and easy to acquire for new users who already have practice with them;
- It is simple and easy to learn. At the same time it has all main functionality, including step-by-step debugging, watching of expressions values change while debugging;
- It is internationalized. User interface uses Unicode controls and data processing is done using Unicode encoding. It is easy to localize to any language, because all localizable resources can be extracted and localized using standard localization tools;
- It uses an extended version of the Free Pascal compiler which supports Unicode encoded programs, allows the use of Unicode identifiers; and has implemented many other features that makes it more suitable for teaching programming.

5 Localization of Free Pascal Compiler

Localization of software that is used for teaching is essential for various reasons. These are the most important: 1) the uphold of the inheritance of Lithuanian language and culture implementing the information technologies 2) Lithuanian law on the state language states “The State shall guarantee the residents of the Republic of Lithuania the right to acquire general, vocational, higher post-school and university education in the state language”.

There are three Free Pascal compiler localization tasks:

- Translation of compiler messages;
- Translation of programming language documentation;
- Localization of the programming language itself.

We already have translated compiler messages. That was not a big task but it required exceptional thoroughness because neat compiler messages have to reflect exact meaning and be comprehensible not only for professional programmers but also for students that do not have enough experience yet.

We do not see difficulties with translation of programming language documentation. It is delivered in HTML format, but we plan to convert it into the more convenient HTLMHelp format. Documentation is an important means of consulting about programming language constructs and cases of their use.

There is a slightly more complicated situation with localizing programming language. Compiler developers usually do not provide possibilities of localization, yet. One of the most widely known languages in the world that is usual to localize is LOGO. In Lithuania we use it for teaching algorithms in younger grades.

It needs to internationalize the Free Pascal compiler prior to make possible localization of the language [12]. We have planed two stages of internationalization:

- Implementation of support of multilingual source code;
- Implementation of support of multilingual lexical elements and separation of them from the compiler’s source code.

Free Pascal internationalization tasks were prearranged and bowl along. We have implemented support for multilingual source code already. Unicode encoding method UTF-8 has been used for that (e.g. that method is used by Delphi 2005, Visual Studio 2003). This method allows making syntax extensions step-by-step and it is easier because encoding of ACSII characters remains the same. Unicode is used not only for source code encoding but for internal data too. Obviously, the implementation of internal data structures and processing of internal data requires most of the extensions and is the most complex of all tasks. For internal data encoding Unicode method UTF-16 has been used. This method is used by most of the operating systems. Because of that it is possible to use operating system’s API in a more efficient way. It does not require additional encoding of data. Unfortunately older Windows systems – Win9x does not support Unicode. To maintain consistency between all Windows systems, Microsoft Layer for Unicode has been used in that case.

We have extended standard text input and output functions in order to support Unicode too. Not all of the text streams support Unicode (e.g. Windows console). Text is encoded using the default code page of system in these cases.

Support for multilingual lexical elements is not very complex. Mostly it requires extensions of lexical scanner. Support has to cover all lexical elements: comments, strings, identifiers, reserved words, directives, operations, numbers, etc. The internationalized programming language has to allow for the use of mathematic signs: assignment “←”, ordinal division “÷”, multiplication “×”, etc; numbers: Latin “579”, Arabic “٥٧٩”, Thai “๕๗๙”, etc.

Programming language semantic requires many extensions too and they are more complex. Let’s consider the following example: in some cases compilers have to treat string constants as single-byte strings (first line) in other cases as multi-byte (second line).

```
const abc : array [1..4] of char = ('a', 'a', 'b', 'c');  
const a = 'a';
```

The syntax of programming language depends on selected locale. E.g. decimal separator: USA “.”, most of European countries “,”; format of numbers: Lithuanian “1 234 567 890,12” Swiss “1'234'567'890,12”, Japanese “12億3456万7890.12”, etc. [12]

It is easy to describe lexical elements formally, so it is worth to include them into locale data after separating them from source code. That will help to solve the compatibility problem between different locales. The best way to implement locale data is to link it dynamically in compiler runtime (Fig. 3). By that way support for older programs may be done including pseudo-locale (it is named as en-ST it Fig. 3) that implements lexical elements corresponding to standard compiler syntax.

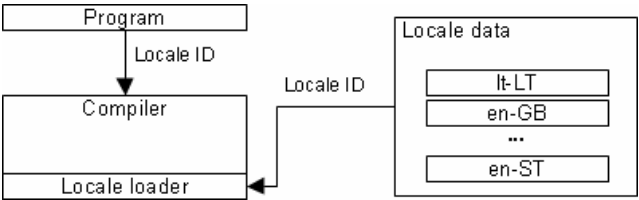


Fig. 3. Compiler locale data

Fig. 3 describes the compilation process. Compiler selects appropriate locale according to system locale or according to supplied compiler switches or directives. After that it loads appropriate locale from its own locale database. Initial locale data for this database may be collected using publicly available locale data from Unicode’s Common Locale Data Repository (UTS #35: Locale Data Markup Language).

6 Conclusions and Challenges

Free Pascal language suits for teaching programming very well, but the Free Pascal compiler does not meet all proposed criteria. However we were looking forward when choosing it, because it may be fully adapted to suit all of them.

Most of the tasks proposed for adaptation of the Free Pascal compiler for our programming curriculum have been accomplished and already have been successfully applied for that purpose.

Though Free Pascal language is modern and represents ideas of object oriented programming well, it is still necessary to supplement it with methodically developed libraries of simple, visual, interactive and attractive objects if we want to apply it for teaching object oriented programming in optional course.

References

1. Bruderer, R. Object-Oriented Framework for Teaching Introductory Programming. Master thesis. Zurich: Swiss Federal Institute of Technology, 2005.
2. Dagienė, V., Blonskis, J. Teaching of programming in advanced course of informatics. Lietuvos matematikos rinkinys, No 42., Vilnius, 2002, pp. 229–234 (in Lithuanian)
3. Dagiene V. Teaching Information Technology in General Education: Challenges and Perspectives. Lecture Notes in Computer Science. R. T. Mittermeir (Ed.). Springer-Verlag, Berlin, Heidelberg, vol. 3422, 2005, 53–64.
4. Dagiene, V. The Model of Teaching Informatics in Lithuanian Comprehensive Schools. Journal of Research on Computing in Education, Vol. 35, No 2 (2002-2003), pp. 176-185.
5. Price, C. B. An Integrated Programming Environment for Undergraduate Computing Courses. *2nd International Conference "Information Technology: Research and Education"*. T. Boyle, P. Oriogun, A. Pakstas (Eds.), London, 2004, pp. 141–144.
6. De Raadt, M., Watson, R., & Toleman, M. Language Trends in Introductory Programming Courses. Informing Science 2002, Ireland.
7. General Curriculum and Education Standards: Pre-school, Primary, and Basic Education. Vilnius, Ministry of Education and Science, 2003 (in Lithuanian)
8. General Curriculum for General Education School in Lithuania and General Education Standards for Grades 11-12. Vilnius, Ministry of Education and Science, 2002 (in Lithuanian)
9. Gupta, D. What is a good first programming language?. ACM Crossroads Computer Science Education Issue 10.4. 2004
10. Haden, P.& Mann, S., The Trouble with Teaching Programming, Proceedings of the NACCQ, pp. 63-70, July 2003, Hamilton, New Zealand.
11. Kölling, M. and Rosenberg, J. (2001). Guidelines for Teaching Object Orientation with Java. Proceedings of the 6 th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, pp. 33-36.
12. Laucius, R. Free Pascal compiler internationalization. Information Sciences, Vol. 34, 2005, p. 302–306. (in Lithuanian)
13. Lee, P. A. Phillips, C. An Assessment of C++ as an Introductory Teaching Language. Technical report series- University of Newcastle upon Tyne computing science, University of Newcastle, 2002.
14. Machanick, P. Teaching Programming Backwards, Proc. Southern African Computer Lecturers' Association Conference, Golden Gate, June 1999, pp 69-73
15. McIver, L & Conway, D.M., Seven Deadly Sins of Introductory Programming Language Design, Proc. Software Engineering: Education and Practice (SE:E&P'96), University of Otago, Dunedin, NZ, pp.309-316, IEEE Computer Society, 1996.

16. Open Source in Education. Vilnius, Ministry of Education and Science, Centre of Information Technologies of Education, Institute of Mathematics and Informatics, 2004 (in Lithuanian)
17. Sebesta, R. W. Concepts of Programming Languages, Sixth Edition. Addison Wesley, Boston, 2003.
18. Wirth, N. The Programming Language Pascal. Acta Informatica, 1, 1971, pp. 35-63

Object-Oriented Programming at Upper Secondary School for Advanced Students

Lubomir Salanci

Department of Informatics Education
Faculty of Mathematics, Physics and Informatics
Comenius University, 842 48 Bratislava, Slovak Republic
salanci@fmph.uniba.sk
www.edi.fmph.uniba.sk/salanci

Abstract. There are powerful computers and new operating systems at secondary schools but we observe that many teachers teach programming in old programming environments such as Borland Pascal for DOS. Imagine Logo or Delphi are relatively new programming environments in comparison to DOS based environments. They bring nice programming paradigms as object-oriented programming, event-driven programming and the idea of constructing application by direct manipulation with components. We believe that more advanced students would like to learn and use these new features. Therefore we run educational research and tried to answer questions like “is it possible to teach basics of programming in object-oriented programming”, “are students of secondary school prepared and willing to learn object-oriented programming” or “can object-oriented programming be useful for them”?

1 Introduction

We have observed that students and teachers want to use modern programming environments such as Delphi. New programming environments offer object-oriented programming languages, for example Delphi offers Object Pascal. But we often meet teachers who do not know how to teach basics of programming in object-oriented environments. When we asked them “where are problems?” their usual answers were that “languages and environments are too object-oriented”, “I cannot use traditional methodical materials in object-oriented environments”, “I cannot transfer my programs, my examples, into new programming language or programming environment”.

Therefore, teachers keep teaching Borland Pascal for DOS or FreePascal although students work in operating system Windows. We understand teachers approach because good school books are missing. There are nearly no methodical materials and training courses. Therefore, we tried to find:

- Which topics could be taught in object-oriented environments?
- What kind of projects we may carry out at upper secondary school?
- Are students able to learn object-oriented programming?

2 The Content

Our schools have very good experience with Logo and Pascal as programming languages. Therefore, we decided to use Delphi in our research. We were teaching students of the 2nd, 3rd and 4th grades (age 15–18) at upper secondary school.

Students of the 2nd grade had a small introduction to programming before and they used Borland Pascal for MS-DOS. We continued in teaching the basics of programming but we used Delphi as a programming environment. Here is a brief list of topics that we have taught:

- Introduction to the Delphi environment,
- Variables, loops, conditions,
- Working with graphics, colours, pictures, working with the mouse,
- Animations – working with bitmaps, controlling characters via the keyboard,
- Records, arrays and text files,
- Sorting,
- Turtle graphics – building our own units,
- Recursion.

We were teaching algorithmic problem solving and we made an introduction to object-oriented programming and design for students of the 3rd grade:

- Backtracking,
- Fractals,
- Introduction to object-oriented programming, inheritance and polymorphism,
- Working with dynamic arrays and streams,
- Simulation of behaviour,
- Parallel processes.

We were teaching students of the 4th to use advanced algorithms, data structures and programming techniques:

- Solving equations using approximate algorithms,
- Binary trees,
- Introduction into OpenGL,
- Programming for the internet using TCP/IP,
- Spline curves,
- Multimedia.

We taught many traditional topics but we have developed a lot of new ones. We would like to show, describe or explain:

- How we have been teaching basic of programming in Delphi,
- How we have implemented user's input – how to replace `ReadLn`,
- What is easy to do in traditional programs but is not easy to transform to event-driven applications.

2.1 Teaching Basics of Programming in Delphi

We made a small introduction into the Delphi programming environment as we started teaching students of the 2nd grade. We decided to use two components: `Image1` as an instance of `TImage` and `Button1` as an instance of `TButton`. We had the following reasons for choosing them:

- `TImage` is an area where we draw some shapes and write texts; it serves as an output to program (we did not use the `WriteLn` command),
- `TImage` can “remember” a drawing; therefore, students do not need to care about re-painting the main form,
- `TButton` “executes” commands that we programmed. Students are programming the body of the `OnClick` event-handler.

We can teach basics of programming using these two components, only. We can teach: “how are commands executed”, “how to display simple greetings”, “how to change font or colour”... For example, we can display a simple message in this way:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Image1.Canvas.TextOut(0, 0, 'Hello!');
end;
```

We teach to use simple graphics as soon as possible. We do not talk about objects, here. Using graphics, we were able to teach other basics of programming, like:

- Simple variables,
- The **for** loop, nested loops,
- The **if ... then** command.

For example, using the **for** loop we draw lines instead of listing texts:

```
for I:=0 to 100 do begin
    Image1.Canvas.MoveTo(0, 0);
    Image1.Canvas.LineTo(I*2, 200);
end;
```

We had a bigger possibility to choose nice examples or assignments when we used graphics environment and graphics commands. It was much more interesting for students to draw, to use colours, to play with pictures instead of displaying texts on the black console screen by the `WriteLn` command.

2.2 How to Implement User’s Input

There are plenty of components that provide input. Let’s look at this example which is often used in the traditional programs written in Pascal:

```
ReadLn(Name);
WriteLn('Hi ', Name);
```

We can implement the same algorithm using the `TEdit` component:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Image1.Canvas.TextOut(0, 0, 'Hi');
    Image1.Canvas.TextOut(20, 0, Edit1.Text);
end;
```

Moreover, we can use better components to acquire different types of input values. For example, we can use scrollbars to get integer value:

```
procedure TForm1.ScrollBarChange(Sender: TObject);
begin
    Image1.Canvas.Brush.Color:=RGB(
        ScrollBar1.Position,
        ScrollBar2.Position,
        ScrollBar3.Position);
    Image1.Canvas.Rectangle(0, 0, 400, 300);
end;
```

Remark: Delphi offers a large number of components. They allow us to obtain values of different types as numbers, texts, list or tables. We must be very careful in introducing new components. We want to teach algorithmic thinking and programming – we do not want to teach components... Therefore, we introduce and explain a new component to our students only when we need it.

2.3 What Is Simple to Implement in Traditional Programs But Is Not Easy to Transform to Event-Driven Applications?

We shouldn't use some traditional techniques in an event driven applications as infinite loops or delays. There is an example of such program for MS DOS:

```
while True do begin
    Time:=Time+1;
    WriteLn(Time);
    Delay(1000);
end;
```

Infinite cycles are not recommended in an event-driven application because application looks like “frozen” (unless we use threads). We can use the `TTimer` component to implement this algorithm. Timer executes commands in periodic intervals:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Time:=Time+1;
    Label1.Caption:=IntToStr(Time);
end;
```

A traditional program could consist of a one monolithic block of commands. Let's look at this fragment of code – it was often used to animate a character:

```
Initialize_Graphics;
repeat
    if KeyPressed then begin
```



```

c:=ReadKey;
if c=#0 then begin
  c:=ReadKey;
  if c=#77 then X:=X-10;
  if c=#79 then X:=X+10;
end;
end;
Clear_Screen;
Paint_Character;
delay(50);
until False;

```

We usually split it into several event-handlers in an event-driven application:

```

procedure TForm1.FormKeyDown( ... var Key: Word ... );
begin
  if Key=VK_LEFT then X:=X-10;
  if Key=VK_RIGHT then X:=X+10;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Clear_Canvas;
  Paint_Character;
end;

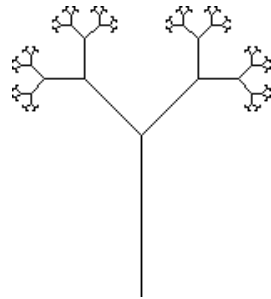
```

It was easy to transform the previous traditional program into a new event-driven application. But, there are situations when similar straight transformation is not so easy. Let's look at the following procedure. It draws a fractal using recursion and turtle graphics:

```

procedure Tree(Length: Real);
begin
  if Length<0.5 then Exit;
  Delay(100);
  Forward(+Length);
  Left(45);
  Tree(Length/2);
  Right(90);
  Tree(Length/2);
  Left(45);
  Forward(-Length);
end;

```



The `Delay(100)` slows down the program to enable students seeing the process of recursive drawing. It is hard to convert this program so that a new program uses the timer. Probably, we should use a stack to simulate recursion call and to store the state of the program between two events of the timer. From a didactics point of view, this solution hides the recursive call and complicates the previous simple program. A small trick will be very useful. We can simulate the `Delay` procedure in Delphi:

```

procedure Delay(Time: Integer);
begin
  Form1.Update; // update content of form
  Sleep(Time);  // sleep application for a while
end;

```

This trick is useful for debugging and demonstration purposes. It is not appropriate for real applications. We avoid students using these tricks.

3 Different Levels of Handling of Objects

We do not want to create “console applications” but we want to create programs for Windows. We saw in the previous sections that Delphi brought several new ideas and techniques to design and create programs. Delphi allows us to construct nice application interfaces, to write and debug programs. Therefore, we had to use objects in our programs and we had to explain the object-oriented paradigm to our students. But, is it possible to teach objects? How is it possible to explain objects to the programming-beginners?

We know that we must teach simple and easy things at the beginning. When students are familiar with the basics of programming then they can go forward and they are able to solve problems and to discover new things.

We did not want to explain “how to define a class of objects” to programming-beginners. If he or she does not know “what is a variable”, “what is a function” then he or she will not understand such abstract thing as a definition of a class. Therefore, we started to distinguish different levels of object-oriented programming:

- Using of objects and components,
- Designing our own objects, hierarchy of classes and programming our own classes of objects.

We will now explain what each level means and what kind of activities or topics we are able to carry out at each level.

3.1 Using of Objects

Let’s imagine that some advanced programmer prepares good objects and we use them in our programs. We do not create any class. Instead of this, we are simply using objects. We can find examples of such an approach in different programming environments:

- Turtles in the Imagine Logo environment,
- Components such as button or image in the Delphi environment.

It is very easy to use objects if they are well designed. A programmer-beginner does not need to know that he or she uses an object. For example, there is a command called `forward` in Imagine Logo. It moves the turtle forward by some distance. Command `forward 100` causes that the turtle moves forward by 100 steps. The beginner will understand this explanation. Beginner will be able to draw simple lines on a computer screen using this command. But he or she does not need to know that command `forward` is a call of a method of an object of the class `Turtle`.

We can find similar examples in the Delphi environment; although, objects are more visible in Delphi than in Imagine Logo. While students are designing an appearance of application they put some components and controls into form. But, they need not to know about objects. They use components as Lego bricks. When students use Delphi’s Object Inspector to change properties, they manipulate properties of

objects in very intuitive way. Objects become much more visible as soon as students start writing their first programs.

We observed that students are able to create small and simple programs, although they do not understand “what exactly object is” and “how does it work”. At this time, we hide objects behind metaphors. For example, we say to students:

Use this command to draw a rectangle into a graphics area:

```
Image1.Canvas.Rectangle( ... )
```

Use this command to display a text:

```
Image1.Canvas.TextOut( ... )
```

It is much more important for students to understand that commands perform certain actions or commands may have parameters. We hide many “technical” facts about objects, at this time. Instead of talking about “objects and methods” we used the simpler term “command”.

It was useful to apply the principle “do not tell everything” at the beginning because we were concentrated on explaining of programming basics. But, it would be a disadvantage to continue in this strategy later. We are risking that we would obscure information and we would make things unclear. Therefore, we explain more facts during the next lessons, such as:

- Component `Image1` is containing the graphics area `Canvas`,
- The graphics area has commands to draw basic geometric shapes, like `Rectangle`, or it has a command to display text the `TextOut`,
- The graphics area has some settings that allow us to change the colour of a line, colour of filling or colour of letters.

Still, we do not talk about methods.

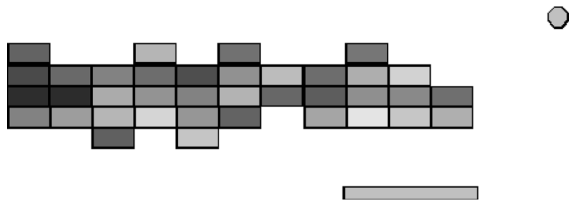
Remark: As we are designing the appearance of our application, Delphi generates a class of application automatically. Students risk meeting with inheritance of classes in time when they are not prepared to understand this idea. Here, students do not use inheritance intentionally, therefore we are still talking about using objects. We explain the idea of inheritance only latter, when students move to a higher level, the level of designing and programming classes.

3.2 Programming Our Own Classes

Designing, creating and programming objects and classes is a higher and more advanced level of programming. At this time, students have to know what is a variable and what is a function to create a class of objects. Moreover, they need certain level of analytic and abstract thinking to be able to design a class. Designing requires imagination and thinking ahead “what will be the attributes of object”, “how the object will behave”, “how the object will interact with other objects”...

We created a simple version of the *Arcanoid* game when we introduced object-oriented programming. We started with designing a simple class that represents a ball. We did not want to complicate task at the beginning. Therefore we did not program collisions testing as bouncing ball from bricks and paddle or destroying bricks. Together with students we discussed about the ball object:

What are properties of the ball? ... position and direction,
What actions do we expect from the ball? ... initialization, movement, painting.



Then we defined the object's class and we programmed methods. We tested one object whether it works. If it moves and displays itself on the screen. Later, we designed classes for bricks and paddle, we accomplished detection of collisions, bouncing and destroying bricks. We created whole game step-by-step together with students.

Remark: It is important to choose a simple project with simple objects and rules. We do not want to solve hard problems. Then we can concentrate on designing classes and object-oriented programming. We do not want to cover crystallization of new knowledge by solving a hard problem.

We helped students with designing of object and classes at the beginning. It was hard to design a class while they did not have enough experience. Later, students were able to design their own classes and they were able to use inheritance and polymorphism to derive new classes in their own projects.

4 Evaluation of the Research

We asked different people to write their opinions about object-oriented programming:

- We asked students who we were teaching at upper secondary school.
- Moreover, we asked future teacher of Informatics at our faculty.

4.1 Questionnaire for Students

We designed a questionnaire for our students at the upper secondary level at the end of their study in the 4th grade. We asked them

- to compare programming in Delphi with programming in Pascal for DOS,
- to tell us their opinions about object-oriented programming.

a) Do you prefer programming in Delphi for Windows or Pascal for MS-DOS?

I prefer Delphi	It does not matter	I prefer Pascal
56%	22%	22%

b) Do you think that using objects in your programs is helpful?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
100%	0%	0%	0%	0%

c) Do you think that designing objects and classes is useful for you?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
78%	22%	0%	0%	0%

We discovered that

- students prefer programming in Delphi.
- object-oriented programming is meaningful for students.

4.2 Questionnaire with Future Teachers

We prepared another questionnaire for our future teachers of the subject informatics. They came from very different secondary schools. They completed a course on programming at our faculty but they haven't taken lectures on teaching programming so far. They were not influenced by our opinions about teaching programming. Therefore, we were interested in their opinions about object-oriented programming in secondary schools.

a) Do you think that using objects is helpful?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
77%	15%	8%	0%	0%

b) Do you think that using objects in your programs is useful?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
82%	0%	0%	8%	0%

c) Do you think that students at upper secondary schools should learn basics of object-oriented programming in higher classes?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
46%	46%	8%	0%	0%

d) Would you teach basics of object-oriented programming at upper secondary schools in higher classes?

Definitely yes	Maybe yes	I don't know	Maybe no	Definitely not
67%	25%	0%	8%	0%

We discovered that

- object-oriented programming is meaningful for future teachers,
- future teachers think that basics of object-oriented programming belong at upper secondary schools for students in higher classes.

We can see similar opinions on object-oriented programming in both questionnaires. Both students and future teachers are thinking that object-oriented programming is helpful and useful.

5 Conclusion

We have explored the possibility of teaching programming in an object-oriented environment at upper secondary schools.

The combination of Delphi and Windows allowed us to use graphics from the very beginning. We skipped problems with complicated initialization of graphics and we did not solve limitations that exist in Borland Pascal. Also, it was easy to use mouse, keyboard and to program animations. We found many interesting topics like using threads to program a race of sorting algorithms, programming for Internet, creating 3d graphics applications using OpenGL.

We think that it is important to distinguish between using objects and designing classes. Students were programming in object-oriented environment and they had to use objects in their programs from the very beginning. We discovered that if we introduce objects in an appropriate way, students have no problems to work with objects.

We designed and administered a questionnaire about object-oriented programming. Both results show that object-oriented programming is very popular among young programmers.

Acknowledgement

This research was funded by ESF project *MISS 21*  **Európsky sociálny fond**.

References

1. Blaho, A., Kalas, I.: Object Metaphor Helps Create Simple Logo Projects, EuroLogo 2001, Österreichische Computer Gesellschaft
2. Blaho, A.: <http://www.delphi.input.sk>
3. Salanci, L.: <http://user.edi.fmph.uniba.sk/salanci>

Informatics Education at Austria's Lower Secondary Schools Between Autonomy and Standards

Peter Micheuz

University Klagenfurt, A-9020 Klagenfurt, Austria
peter.micheuz@uni-klu.ac.at

Abstract. The present state of informatics education at lower secondary schools in Austria is the result of many single autonomous decisions which have been made locally at schools. The Austrian Ministry of Education has left it to schools to define their profiles and to decide about the amount of imparting informatics to their pupils who are aged between 10 and 14 years. This paper gives a short historical overview and some insight into the present dialectic process between autonomy and the upcoming issue of educational standards. A recently conducted evaluation project in Carinthia/Austria, which can be considered as representative for related current studies in Austria, is presented. Empirical research is important. It perfectly fits the ongoing shift from input orientation to output measuring. Whereas other traditional subjects as (foreign) languages or mathematics are based on solid, historically developed fundamentals, the comparatively new field of informatics at schools is still looking for stable frameworks. Considering the newest developments, we can be optimistic to achieve a reasonable solution for improving informatics education at lower secondary schools in Austria. Hopefully the announced educational standards in informatics will help to decrease the digital divide among our pupils.

1 Introduction

The lower secondary schools in Austria educate pupils at the age-groups ranging from 10 up to 14 years. About 70% of all primary school leavers attend secondary general schools (Hauptschule/HS), and the cognitively better gifted pupils attend secondary academic schools (Gymnasium/AHS). Compulsory education in Austria ends after 9 years of school attendance. After that about 40% are attending higher secondary schools. This paper focuses on the present state of IT/informatics in lower secondary education.

Informatics education at the lower secondary level has its roots in the late 80-ties of the last century. An informative nationwide study in [9] shows explicitly the efforts which have been made to provide (some, not all) pupils at an early age with basic computer and programming skills.

At that time there were three approaches to establish informatics in lower secondary schools. The first was the non obligatory subject informatics offered at the 7th and 8th level of education. This was attended by a minor part of all the pupils. The second one was a compulsory project week where all 13 and 14-year-old pupils

should get computer supported lessons. The third approach was an attempt to integrate the computer in some other subjects by ministerial enactment. The study reveals that the output was insufficient, except for the non-obligatory informatics lessons to the extent of two hours a week which were based on a central curriculum. The reasons were lack of appropriate standardized software, insufficient hardware, and therefore rather poor acceptance of the teachers involved.

In the meantime the basic conditions for imparting IT/informatics at schools (should) have noticeably changed.

In the year 1995 a fundamental school reform was initialized in Austria. The ministry of education granted all schools autonomy and thus empowered schools to develop their own characteristic profiles. In this context IT/informatics played and still plays an important role. In many schools informatics lessons have been established as an obligatory subject.

Since 1999 there has been a new curriculum for the lower secondary academic and for the lower secondary general schools which consists of core subjects and areas of extension.

As a matter of fact policy makers for education neglected to anchor informatics in the core curriculum. IT/informatics has been abandoned completely to autonomic forces at the schools. The only curricular hints can be summarized as follows:

"Innovative technologies are gaining more importance in our lives (...). Within the scope of education, ICT has to take this development into account. Moreover the didactic potential (...) has to be utilized.[1]"

Additionally ICT can be found as an educational principle, a recommendation with more or less obligation, expressed in the single sentence: "In the educational process new technologies should be applied." Furthermore some remarks of curricula for core subjects mention "IT should be used". Unfortunately we lack information to what extent these recommendations and enactments have been executed.

Another important step in the process of enhancing quality in our educational system is the prescribed change from input orientation to standardized learning objectives and output measuring. Presently there are already task forces establishing educational standards for the fourth grade of primary and lower secondary level. A recent report from Austria's Ministry of Education confirms five developments.

- The autonomy of schools to alter timetables and introduce new subjects,
- new curricula,
- change from input orientation to output measuring,
- establishment of educational standards,
- offering and supporting the ECDL (and other IT-certificates).

These general recommendations described in [2] have important implications for IT/informatics especially for the lower secondary level.

2 Results of Autonomy in the Field of IT and Informatics

Although informatics does not explicitly occur in the canon of obligatory subjects, there have been many initiatives in establishing informatics at an early age group. Today 10 to 14 year-old pupils face a wide range of offers in informatics education

which varies from school to school. In this spectrum we find courses for typewriting, word processing, non-obligatory tutorials, and even not so few (in some federal states about 50%) obligatory lessons in informatics which have been established at the expense of other subjects. Furthermore, there are some models to integrate the computer and informatics methods in mathematics, foreign languages or other established subjects.

About 20% of all the lower secondary academic schools are already involved in the eLSA-project [5] which aims at introducing e-learning and new technologies in schools. The interdependency of IT literacy and effective e-learning is obvious and a worthwhile field of research. This does not only apply for pupils but also for teachers.

In many schools autonomy has yielded astonishing efforts and initiatives as for installing IT and informatics lessons at Austria's lower secondary level. Competition among school types, where HS and AHS are attracting pupils by means of widely offering IT/informatics education, can still be regarded as a promoter of the ongoing IT-related development of schools.

Some schools perform excellently for reasons of an active school administration and due to highly motivated teachers. The result is not only a growing digital divide among schools but also among pupils which is not really satisfactory. Policy makers would be well advised to be aware of this fact and to take appropriate countermeasures.

3 Some Research Findings About a Carinthian Initiative

Since the beginning of the school year 2002/2003 almost all secondary academic schools in Carinthia/Austria offer the obligatory subject informatics in the first two years at the expense of other subjects. Carinthia is the most southern country of Austria with about 600.000 inhabitants, that is about 7,5% of the Austrian population.

This initiative is unique in Austria and due to the farsighted decision of the former presidency of the Carinthian school authority. Two years later an evaluation and standardization project was launched. Some core statements and research findings about this still ongoing project, which was published in [13] can be summarized as follows.

- Almost all (ca. 90%) lower secondary academic schools in Carinthia take part in this project based on autonomous decisions.
- The amount of informatics lessons is one hour per week with varying organizational characteristics. Almost half of the schools have decided to offer double hours every two weeks and the reduction of lessons has taken place in different subjects such as German, crafts or drawing.
- The involved schools agreed upon a common standard as shown in Fig. 1.
- A collection of many assignments covering this standard has been developed and is available for all schools.
- All teachers and pupils involved in this project appreciate this initiative.
- The schools involved are inhomogeneously equipped with computers due to autonomy. The ratio between number of pupils and the number of computers ranges from 1:15 up to 1:6.

- Practically 100% of the pupils have at least one computer at home and about 70% have access to the internet. This applies to AHS. There is no study about HS.
- The pupils like this subject very much regardless of gender. Boys declared to be a little more fascinated than girls.
- There is a significant loss of fascination from the 5th to the 6th grade.
- Most of the pupils performed - with few exceptions – very well.
- About one third of the pupils at primary level is already experienced in handling computers by informal instruction.

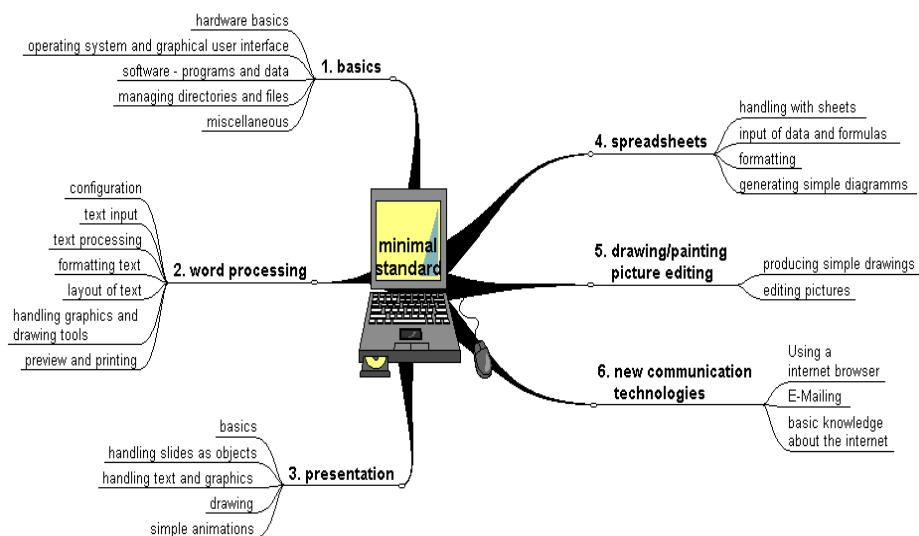


Fig. 1. Standardized learning objectives for 12-year-old pupils in Carinthia/Austria

Especially at the transition from primary to secondary schools an attempt for standardization makes sense in order to avoid a further digital divide among pupils at an early age.

The collection of the objectives can obviously be seen as an adopted subset of the syllabus of the ECDL (European Computer Driving License) [6]. This very pragmatic attempt to standardize the basic IT-skills and IT-knowledge of pupils at an early age was the common denominator of all schools involved.

Defining standards and producing objectives is just one side of the coin. Output measuring and the results of two years and about 60 hours formal informatics lessons with clear guidelines is even more important.

A second empirical, unpublished study was conducted in 2005 by a group of students at the University of Klagenfurt in form of an online research that used the following instruments:

- an online-questionnaire for teachers and pupils,
- an online-test for pupils with a dichotomous answering format covering all fields of the standard,

- practical assignments for the modules text processing, spreadsheets and presentation.

The target group consisted of a random sample of all teachers and pupils at the end of the 6th grade and had a reasonable rate of return (about 40%) in consideration of the fact that the survey was carried out in June at the end of the term. Some major findings after evaluating the questionnaires for teachers were:

- The lessons take place all the time in computer labs where there is one computer per pupil. Universally, there is a high satisfaction with the technical equipment.
- Most teachers find typewriting very important although about 75% do not favor it very much in their lessons. In general, the teachers are poor in typewriting.
- The standard is still considered very helpful and the majority of teachers find that the level of the standard is compatible with the amount of available informatics lessons. One hour informatics lesson per week is considered sufficient.
- On average the classes consist of 13 pupils.
- There is beamer support almost all the time.
- Almost all teachers find that grading/marking pupils makes sense.
- Two thirds find informatics lessons stressful.
- About 65% of the teachers give (sometimes) home works.

The following statements result from the pupils' questionnaire:

- About 75% want to have informatics also in the 7th grade.
- More than 60% want to have more informatics lessons
- 70% use the computer also in other subjects and do (sometimes) their homework using the computer.
- 75% sometimes play on the computer with permission of the teacher during the lessons.
- Practicing with the computer for school at home is not very popular (about 50% are practicing very little or nothing).
- Almost 50% are admitting that they play on the computer during lessons without the teachers taking any notice of this fact.
- About 80% like informatics lessons, the others definitively don't like them.
- About 35% of the pupils admitted that they do not like to go to school (!).
- More than 80% are allowed (by the parents) to use the computer at home all the time.
- 80% have access to the internet at home.
- About 75% indicate that they work for school with the computer less than 1 hour.
- About 35% are spending more than 3 hours with the computer at home.
- About 10% are spending more than 5 hours on the internet.
- 75% of the parents use the computer in profession.

A proverb from Pythagoras is: "The oldest, shortest words - "yes" and "no" - are those which require the most reflection". Accordingly, our online-test consisted of representative choice of true-false items and assessed the pupils' ability to recognize the accuracy of declarative IT-relevant statements in the context of the Carinthian standard. This method of true-false questions turned out as a well-suited means for quickly and collectively measuring knowledge and comprehensive understanding. As

the usefulness and effectiveness of true-false items in higher education is still debated, we carefully considered the advantages and disadvantages of true-false assessments in relation to their specific learning objectives.

The result of this online-test with 80 statements covering the 6 modules of the Carinthian standard had to be marked true, false or skipped. Over 200 tests were completed and more than 16000 answers do impressively give an insight into the strengths and weaknesses of the pupil's IT-knowledge at the age of 12 years. Let us illustrate this by means of some too "extreme" statements.

More than 75% of the pupils answered wrongly:

- A valid address in a web browser always begins with `http://www`.
- A good presentation should contain as many fonts and effects as possible.
- The Internet is part of the WWW.
- Data in the RAM are deleted when the computer is switched off.
- A document which contains text has necessarily the file extension "txt" or "doc".

As one can imagine, these statements have been well done by about 90%:

- A password should not be given to other persons.
- The Austrian internet addresses end with "at" in most cases.
- Google is a well known search engine.
- You can insert pictures into a word-document.
- One directory can contain more directories.

These were the statements which had been skipped by more than 25% of the pupils:

- By drag and drop you can push objects.
- BCC means that the receiver only gets the attachment.
- Text files cannot be changed by an editor.
- With the insert key you can change from the insert modus to the overwrite modus.
- Gif- and jpg-files are text files.

The order of the statements during the test was randomized for every pupil. So it can be excluded that the pupils had insufficient time for parts of the test and did not answer because of lack of time.

Finally all the pupils had been tested practically by concrete assignments, containing tasks such as text processing, producing spreadsheets and presentations, and using the internet. The evaluation of these tasks revealed that the presentation assignment had been solved best, followed by the text processing task. A simple calculation example (summing up a column and producing a diagram) was a big problem for most pupils. It should be mentioned that we discovered at least in one school massive cheating. It was up to the particular informatics teacher to provide a reliable test situation...

The next step in evaluating the Carinthian project will be a research focussing on investigating the objective "sustainability". The first results of this study will be available in summer 2006. The target group of this research will be the pupils of the 7th grade and it should show how much IT- competence has remained from the two years of formal informatics lessons. Approximately 20% of the age-group were not in the project and did not take any lessons in informatics in the 5th and 6th grade. This group is included in our research for reasons of comparison and in order to show that informatics knowledge and practical skills cannot be learned "by the way" without formal instruction.

Informatics lessons are rather expensive and the investor – that is the tax payer – for reasons of split groups has the right for a sufficient output. Magenheimer states in [12] that “due to the economic importance of a nation's educational system the focus on education has partly shifted from its content and methods to the outcomes of learning processes and the assessment of their cost-value ration”.

It may be expected that the experimental group, consisting of pupils in Carinthia with two years formal lessons in informatics, performs significantly better than the control group. Otherwise there is a severe problem! The research group at the University of Klagenfurt looks forward to the outcome of this study.

4 The Way to Educational Standards in Informatics

Informatics education needs a solid and consistent development with generous stages of exercising and consolidation. This is only ensured if a framework of appropriate objectives for each level goes along with a sufficient offer of formal informatics instruction. As reported in the chapter above, Carinthia has already done the first steps in the appropriate direction. It is advisable to start informatics in the 5th grade because some pupils have already IT-related pre-knowledge from primary schools.

In this context optimism is indicated, because the disappointing results of the worldwide PISA study (Programme for International Student Assessment) caused in Austria severe debates about the efficiency of our school system. The reasons were quickly assumed to be an overemphasis of input orientation and the disregard of output measurements. As a consequence the magic word “standards” flooded the whole educational system not only in Austria but also in Germany.

“Educational standards (...) aim at general educational goals. They specify the competencies that schools must impart to their students in order to achieve certain key educational goals, and the competencies that children or teenagers are expected to have acquired by a particular age-group. These competencies are described in such specific terms that they can be translated into particular tasks and, in principle, assessed by tests.” [10]

Presently there are some institutions and task forces occupied with research and work on these standards. This applies especially to the PISA-relevant subjects German, English and Mathematics at the transition from lower secondary to upper secondary level.

There is no doubt that setting up also IT/informatics objectives for the lower level secondary education in form of realistic educational standards is absolutely necessary. But what can be measured in the field of informatics when there are not even common objectives and central curricula for the lower secondary level?

It is improbable that the Austrian Ministry of Education will discontinue autonomy and switch to central regulations in form of a nationwide mandatory curriculum. On the other hand there is obviously – as a corrective - a strong demand for evaluation and output measurements. As a matter of fact many – but by far not all – of Austria's lower secondary schools have developed an impressive agility and dynamism due to autonomy.

Now the time for reflecting these developments and their outcomes has come. Reasonable preparatory work has been done in that field till now by informal working

groups. Dörning and Micheuz describe the Austrian progress of educational standards in IT/informatics in [3][16][17].

In March 2006 the Austrian Ministry of Education has decided to install additional task forces for developing national standards in the specific fields of natural science and IT/informatics. It seems that a concern expressed already for a long time has preliminarily come to a good end. The foundations for building a widely accepted framework of IT/informatics competencies in Austria are laid.

5 Standardizing the Educational Standards

At the lower level of Austria's secondary education the syllabus of the European Computer Driving License (ECDL) [6] can be regarded already as a quasi standard. Since its introduction with prominent recommendation from the aforesaid Ministry in 1999 this certificate developed rapidly. Recently we can state an increasing penetration of this certificate into the lower secondary level.

Doubtlessly the ECDL can serve as a useful orientation in building IT/informatics standards. But it is not only the product oriented syllabus which has to be looked at, but also the excellent organizational structure for assessments and quality assurance which has been established in Austria.

In its core version promoting only computer literacy and basic skills the ECDL is seen as a didactically undesirable limitation.

But in Austrian schools it is not possible to disregard the ECDL when thinking about standards. The spread of this certificate especially at the lower level of secondary education has reached already a critical mass. However, considering that the ECDL originally was designed for adults and for the economic world, there are also critical comments on adopting it unchanged for schools. The ECDL foundation and the Austrian Computer Society (OCG) already reacted to that in inventing additional certificates and adjustments.

This might be rather counterproductive not only because of the loss of overview and orientation but also because of the profane reason that pupils and parents are not willing to pay for each certificate.

An informal Austrian working group has developed an informatics standard since 2002 [16][17]. In our hitherto existing work we find that pupils/students should be able to

- cope with informatics systems and understand them as well as related security and social aspects,
- use application software properly,
- apply computers for problem solving,
- take advantage of learning management systems and to gain more self-competence in using IT for learning.

This leads us to the first dimension of our proposed framework. Its content dimension can be briefly categorized as follows [14]:

- informatics systems and social aspects,
- applications and publishing,
- problem solving and modeling
- e-learning.

In our considerations the syllabus of the EDCL can be integrated in this schema very well. Regarding the dimension of content there is much accordance with the ECDL. However it is obvious that this first draft of a standard is much more comprehensive.

Regarding the second dimension of process handling we adopted Blooms taxonomy in form of knowledge and comprehension, application, analysis and synthesis, evaluation. In order to build a framework for all pupils at all age-groups presently three levels of complexity are planned.

Looking over the narrow national borders reveals the European and worldwide approaches in building frameworks and standards. So the question arises why not to abbreviate the long and difficult way for a national solution by looking for a suitable product from abroad and adopting it? In fact, a number of approaches can be found and the choice is already wide.

The FITness program is based on three knowledge levels which have been recognized as important for coping with IT [7].

- Intellectual capabilities: having the ability to solve problems by reasoning, test possible solutions, anticipate and adapt to change, and troubleshoot.
- Fundamental concepts: knowing about computers and information systems, being aware of how they work and how they impact society.
- Contemporary skills: being able to manage a personal computer and use common software applications such as e-mail, word processing, spreadsheets, and databases.

Other approaches are the IFIP curriculum, the ACM framework and last but not least the approach in Germany where the attempt is made to adapt the NCTM standards in mathematics for the realm of IT/informatics. Many informatics didacts around the informal Königstein group (Friedrich S. in [11]) are very active. Their approach is elaborating standards on the basis of the content bands:

- information and data
- algorithms
- informatics and social aspects
- informatics systems – design and functionality
- (programming) languages and automatons.

6 Conclusions

As already mentioned above it is one side of the coin to build highly sophisticated and consistent curriculum frameworks and standards. But this is not the main and most difficult task! A very important process takes place in the classroom and in the pupils' brains, a process which cannot be even controlled by the best curriculum or standard. A new standard is no more than dead paper unless it is widely accepted and followed by teachers and students alike.

Therefore, according to the shift of paradigm from input orientation to output measurement we do not have to focus solely on defining objectives. The more important step in this context is to think of how we can measure the output of an educational system. This is by far the bigger challenge and very demanding also in the field of IT and/or informatics.

The observant reader might have noticed the permanent (artificial) separation of IT and informatics. This is indeed a major semantic problem which needs to be solved before developing educational standards.

References

1. BMBWK, Lehrpläne etc. <http://www.gemeinsamlernen.at> (15.3.2006)
2. Development of Education, Ministry of Education, Austria 2004, <http://www.bmbwk.gv.at>
3. Dorninger C., Educational Standards in School Informatics in Austria, p. 65-69, in From Computer Literacy to Informatics Fundamentals, Mittermeir R. Springer. 2005
4. Eurydice, Key Data on ICT in Schools in Europe, 2004 Edition, <http://www.eurydice.org>
5. <http://elsa.schule.at> (15.3.2005)
6. <http://www.ecdl.at> (15.3.2005)
7. FITness: Being Fluent with IT, <http://www.big6.com/showenewsarticle.php?id=298>
8. Friedrich S., Informatische Fachkonzepte im Unterricht, INFOS 2003, GI-Ed., p.133f
9. Haider G., Schule und Computer, Studienverlag, Innsbruck, 1994
10. Klieme, E. et al., The Development of National Educational Standards, 2004, p.15 http://pisa.ipn.uni-kiel.de/pisa2006/bildungsstandards_eng.html (15.3.2006)
11. Königstein group: <http://koenigstein.inf.tu-dresden.de> (15.3.2006)
12. Magenheimer J., Towards a Competence Model for Educational Standards of Informatics In: WCCE 2005 - Proceedings of the 8th IFIP World Conference on Computers in Education, University of Stellenbosch, Cape Town (SA), 4-7. Juli 2005.
13. Micheuz P., Informatics and Standards at an Early Stage, Informatics and Student Assessment, GI Lecture Notes, Dagstuhl, 2004
14. Micheuz P., Auf dem Weg zu Standards, in LOGIN Nr. 135, Berlin, 2005
15. OCG, Homepage of the ECDL in Austria, <http://www.ecdl.at>
16. Reiter A. ed., Standards in der Schulinformatik, CDA-Verlag, Linz/Perg, 2004
17. Reiter A. ed., Informatische Bildung in der Sekundarstufe I, CDA-Verlag, Linz/Berg, 2005

Development of an Integrated Informatics Curriculum for K-12 in Korea

SeungWook Yoo¹, YongChul Yeum¹, Yong Kim², SeungEun Cha¹,
JongHye Kim¹, HyeSun Jang¹, SookKyong Choi¹, HwanCheol Lee¹,
DaiYoung Kwon¹, HeeSeop Han¹, EunMi Shin¹, JaeShin Song²,
JongEun Park³, and WonGyu Lee⁴

¹ Department of Computer Science Education, Graduate School, Korea University
Anam-Dong Sungbuk-Gu, Seoul, 136-701, Korea

{yoosw0810, yycok, lyndon, elener, micro38, luisy81, wink1011, dykwon,
anemone, sempoby}@comedu.korea.ac.kr

² Korea Education & Research Information Service
KERIS Building, 22-1, Ssangnim-Dong, Jung-Gu, Seoul, 100-400, Korea
{dragon, song}@keris.or.kr

³ Ministry of Education & Human Resources Development
77-6 Sejong-No, Jongno-Gu, Seoul, 110-760, Republic of Korea
psn67@moe.go.kr

⁴ Department of Computer Science Education, College of Education,
Korea University
Anam-Dong Sungbuk-Gu, Seoul, 136-701, Korea
lee@comedu.korea.ac.kr

Abstract. The current informatics education in Korea is based on the guidelines for ICT education announced in Aug. 2000, which was very timely considering the need to utilize the infrastructure that had been constructed. Still, the essential question regarding informatics education as a tool for enhancing the learning effects and solving problems in daily living has been persistently raised. This study resolved the problems related to the informatics education implemented in Korean primary and secondary schools and considered various conditions for implementing the integrated informatics education. An integrated informatics curriculum designed to enhance problem solving capacity was also proposed. Finally, the new guidelines for ICT education announced in Dec. 2005 were introduced.

1 Introduction

Currently, the 6-3-3 system-based 7th Curriculum is implemented as the Korean curriculum for primary and secondary education. The national common basic curriculum consisting of 10 subjects, independent activity, and special activity applies to grades 1-10, and the elective curriculum consisting of elective subjects and special activity, to grades 11 and 12 [1].

In the national basic curriculum, the goal of fostering basic knowledge as required in the information society was set for informatics education, and the

educational contents, expanded to the practical use of the PC as tool in daily life beyond the concept of information processing technology such as programming, etc. The use of ICT was also encouraged in learning activities for each subject based on the ‘Guidelines for ICT Education in Primary and Secondary Schools’ announced in Aug. 2000.

Considering the need for and importance of ICT education in the knowledge and information society, the guidelines sought to promote ICT in informatics and other subjects and to enhance the educational effect in the process. Nonetheless, it only revealed problems such as overlapping and omissions in the educational contents and operations caused by implementing informatics education under various systems as well as contents that focus mainly on learning how to use application software.

Informatics education are carried out on many countries as a regular citizen education for nurturing competitive human resources for the future society. We also have carried out the informatics education from 2000 year. However, many researchers in Korea believe that current informatics education is insufficient for nurturing competitive human resources.

2 Informatics Education and ICT Education in Korea

In Korea, three curricula for informatics education shown in Fig. 1 are currently being implemented: First, informatics education is an independent subject such as ‘Computer’ in middle school, ‘Information Society and Computer’ in high school. Second, informatics-related subjects refer to informatics education in other subjects, i.e. ‘Practical Arts’ in primary school and ‘Technology and Home Economics’ in middle school. Finally, ICT education is the ‘Guidelines for ICT Education in Primary and Secondary Schools,’ which influence the educational contents of the first and second curricula.

2.1 Implementing Informatics Education

Figure 1 shows the composition of informatics education-related subjects. ICT education is practiced until grade 10. Still, informatics subjects or informatics-related contents in other subjects have priority. In the case of grades 1–4 or 10, which have no informatics-related subjects in the curriculum, independent activities or special activities are used in lieu of ICT education. Therefore, informatics education in Korea can either be implemented every school year for grades 1–12. Otherwise, courses for 2–3 school years can be integrated into one year on a class hour basis.

Based on the national common basic curriculum, ‘Practical Arts’ (Technology and Home Economics) is taught in grades 5–8. In the 5th grade Informatics-related contents include the composition of PC, using the keyboard, and typing. Making pictures using PC and using PC communications is taught in the 6th grade, structure and principle of PC and production, storage, and distribution of information in the 7th grade, and practical use of software and using the Internet in the 8th grade. In middle school, ‘Computer’ is either taught for 3 hours a week

School Level Subject	Elementary School						Middle School			High School		
	1	2	3	4	5	6	7	8	9	10	11	12
ICT education	Guidelines for ICT education											
Informatics-related subject					Practical arts		Technology and home economics					
Informatics subject							Computer				Information society and computer	

Fig. 1. Informatics Education-related Subjects in the 7th Curriculum

during one of the three school years or included among the elective subjects for all three years. As of 2005, 2228 out of 2888 middle schools adopted the subject (77.1%), whose contents include men and computer, computer basics, word processor, PC communications and Internet, and multimedia.

In the elective curriculum for general high schools, ‘Information Society and Computer’ can be selected as one of the general elective subjects by 11th and 12th graders. If necessary, 4–6 units can be assigned to the subject as an independent activity performed in the 10th grade. As of 2005, 1689 out of 2080 high schools adopted the subject (81.2%), whose contents include social development and computer, operating a PC, word processor, spreadsheet, computer communications network, and multimedia.

2.2 Implementing ICT Education

ICT education was practiced until the 10th grade, consisting of independent activity in grades 1–4. It was also practiced as either an independent activity, special activity, or ‘Practical Arts’ in grades 5–6. ICT was taught for at least 34 hours per school year, but only for 30 hours in grade 1. In middle school, ICT education can be carried out from the 7th to the 9th grade using informatics-related contents included in ‘Technology and Home Economics’ and additional subject-related independent activities when the subject ‘Computer’ is adopted. In the 10th grade, the first year in high school, subject-related independent activities can be used. Such ICT education started in school year 2001 for 1st and 2nd graders, expanding annually to include 3rd and 4th graders in 2002 and 5th and 6th graders in 2003.

The content framework of the ‘Guidelines for ICT Education in Primary and Secondary Schools’ is categorized into five areas, each of which is further divided into five steps. Step 1 applies to grades 1 and 2, whereas steps 2 and 3 target 3rd and 4th graders and 5th and 6th graders, respectively. Steps 4 and 5 correspond to middle school and high school, respectively. Depending on the learners’ cognitive development level or school-specific conditions, however, the steps can be adjusted. Below are the major contents in each area:

- ‘Understanding information and ethics’ deals with understanding information devices, the concept of information, information and copyright, sharing information in the information society, etc.

- ‘Computer basics’ covers the components and operation of PCs, operating system, computer viruses, utility programs, programming basics, etc.
- ‘Practical use of software’ includes educational application software, multimedia, word processor, presentation, spreadsheet, and database.
- ‘Computer communications’ deals with internet use, email and network configurations, cyber space activities, and search and use of information.
- ‘Comprehensive activity’ covers the collection and use of data via PC communications, cooperative project-based learning, conversion between data types, constructing homepages, publishing internet learning newspapers, and managing homepages.

2.3 Problems in Informatics Education

For the three types and contents of informatics education described in this paper, several problems were noted. For one, informatics education through independent subjects and related subjects and that in independent activities and special activities lack systematic relationship. For example, ‘Technology and Home Economics’ as a compulsory subject among 7th and 8th graders as well as the elective ‘Computer’ are taught at the same time, or ‘Computer’ is not included in the elective subjects for 9th graders. These cases are often seen in Korean middle schools, demonstrating a lack of coherence in the curriculum. Second, educational contents overlap. In particular, 7th and 8th graders learn almost the same contents repeatedly; thus causing them to lose interest in informatics. Third, educational contents focus too much on the acquisition of functional skills for using application software and simple operation of a PC. This could cause the learners to associate information literacy with the ability to use application software, which diverges from the concept of information literacy, i.e., developing the basic abilities to produce, process, analyze, and search information for learning and solving problems in daily living using ICT.

A report by the OECD revealed that students aged 15 (corresponding to 10th graders in Korea) in OECD countries use the PC primarily for electronic communications, followed by internet search, PC games, downloading music files, and document preparation [3]. The use of educational application software is at the bottom of the list, followed by computer programming and use of spreadsheet programs. At least 49% of the students answered that they never used educational application software; 28% of the respondents said they do not use the PC for learning purposes. Korea is no exception when it comes such phenomenon, which says a lot about informatics education, particularly ICT education in the country.

3 Integrated Informatics Curriculum

In the 7th Curriculum wherein an effort was made to integrate ICT into the learning activities through the 5th and 6th Curricula, which emphasized the principles of informatics, the informatics curriculum and educational contents

in Korea focused on the use of application software or internet. Still, the effect turned out to be negative, based on the OECD report in 2004. Thus, a new integrated informatics curriculum supporting learning activities and problem solving in daily living was proposed. The integrated informatics curriculum prepared in this study is expected to set the standards for informatics education as well as ICT education in Korea.

3.1 What Should an Integrated Informatics Curriculum Be Like?

Human resources with proper ICT literacy should be able to solve newly emerging problems based on the basic ability to operate a PC. This is considered the 'basic knowledge required of every citizen living in the ICT society' and the educational contents that should be provided based on the current national common basic curriculum.

To develop the 'basic knowledge required of every citizen living in the ICT society,' the curriculum should focus on information literacy education to deliver the basic ICT concepts and develop elementary ICT usage capabilities for lower classmen in primary school. As the academic level increases, more emphasis should be placed on computer science education for IT application, followed by the understanding of the fundamental concepts and principles of informatics. Overall, the curriculum should be balanced between ICT literacy education and education on the practical use of ICT.

Information processing involves a cycle of collection, analysis, design, processing, creation, and sharing of information. Linking this cycle with an informatics curriculum is not that simple. Figure 2 is an attempt at schematization. In particular, the information processing cycle is believed to be composed of the practical use of application software, principles of information science, understanding the operation principles of ICT-based systems, capabilities for solving ethical and legal issues, and their complementary relationship. Through these interactive processes, capacities for learning and solving problems in daily living are fostered.

The integrated informatics curriculum has been reorganized by adding information science factors for fostering problem solving capacity to the basic content framework of the 'Guidelines for ICT Education in Primary and Secondary Schools' announced in Aug. 2000. Various curricula proposed, particularly the ACM Report(2003), ACM K-12 Level 2 Objectives and Outlines, CC2001, and Japan's High School Teaching Guide (Informatics), were selected as models. The ICT curriculum from the UK and the curricula from Israel, India, and China were also very helpful [4], [5], [6], [7], [8], [9], [10].

3.2 Goals of the Integrated Informatics Curriculum

The integrated informatics curriculum set the goals including understanding informatics based on a proper attitude toward living in the information society and nurturing good human resources befitting the knowledge and information society and possessing the capability to solve problems creatively through the

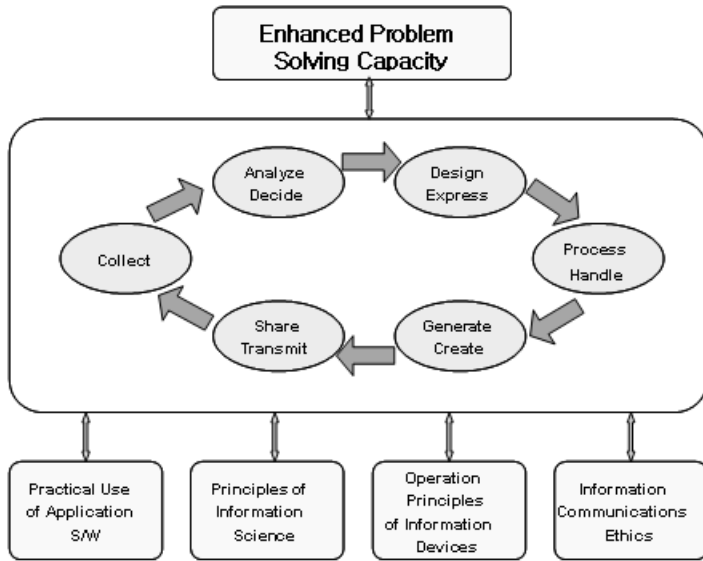


Fig. 2. Information-based Problem Solving

active application of ICT. The following concrete goals were formulated in the curriculum:

- fostering a sound information culture by understanding information communications ethics and communications security.
- developing an ability to model and solve real-life problems by programming procedural thinking.
- using and organizing information devices to solve real-life problems by understanding the composition and operating principles of PC and networks.
- understanding various data expressions and structures to apply them on a problem-specific basis.
- understanding the uses of various information devices and application software and knowing how to use them in daily life and learning the subject.

3.3 Content Framework

Based on the defined goals and on the cyclic model for information processing shown in Fig. 2, detailed objectives were established to complete the content framework. Step 1 applies to grades 1–3, steps 2, to grades 4–6, step 3, to grades 7–9, and step 4, to grades 10–12 (see Figure 3).

The contents are further divided into a total of 76 cells including all contents that should be provided by informatics education in Korean primary and secondary schools. The goals and educational contents regarding the detailed, area-specific content framework are provided to serve as guidelines for developing educational materials and managing classes.

	Step 1	Step 2	Step 3	Step 4
Information society and life	Understanding the changes in the information society and developing a basic attitude toward the practical use of ICT and applying it to daily life	Understanding the practical use of computers in the job and establishing proper moral standards for information protection and sharing	Understanding IT influence on the modern society and using it properly	Fostering appropriate morality as a true member of the information society and applying the necessary measures for information protection
Problem solving methods and processes	Determining data characteristics and planning problem solving procedures	Processing data with a PC and using effective algorithms	Creating simple programs for solving problems in daily life	Understanding problem solving procedures and solving problems creatively through programming
Composition and operation of the PC	Understanding the roles of the PC components and basic process of PC communications	Understanding the functions of information devices and network phenomenon	Understanding the principles of computer systems and explaining the principles of networking	Understanding the principles of O/S through the process of implementing commands and understanding the network structure
Understanding data structures	Understanding how data are expressed digitally and explaining the need for and characteristics of data structure	Understanding data structures and algorithms for expressing an optimal structure	Expressing basic data structures and data structures and classifying types of multimedia data	Expressing data using propositions and logic and understanding data structures and databases
Application of information technology	Understanding the interface and performing various tasks by using application software	Understanding the files to be managed/edited and editing/creating web documents considering their characteristics	Cooperative works to create various presentations by searching the necessary data for learning	Creating/managing homepages by searching and processing data

Fig. 3. Objectives by Areas and Steps

3.4 Basic Strategy for Teaching-Learning

The entire content framework of the integrated informatics curriculum as proposed in this paper was classified by the teaching-learning method and type of knowledge as shown in Figure 4 below [2]. The horizontal axis represents the type of teaching-learning; 1 is the learning activities led only by the teacher's guide; 4, the learner-leading learning, and 2 and 3 are compromise methods. The vertical axis represents the type of educational contents; A includes contents related to thinking, investigating, and deciding, whereas D is the contents to be simply memorized. B and C are compromise types.

Results showed that the contents from those requiring simple delivery of knowledge in the area-specific and step-specific content framework to those enabling meaningful learning through the discovery and inquiry processes can be linked in a systematic distribution. In terms of steps, more meaningful learning

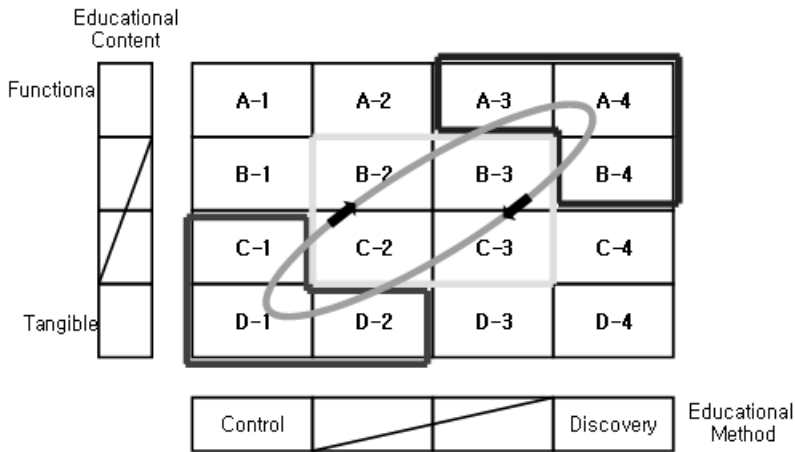


Fig. 4. Relationship Between the Teaching-Learning Method and Knowledge Type

of high-level knowledge through inquiry was identified as the step went up from 1 to 4.

This means that teachers are required to deliver correct concepts effectively through application to daily life when they share elementary knowledge of fundamental concepts and principles. On the other hand, they need to supervise teaching-learning activities through the discovery and inquiry processes to enable learners to acquire higher-level concepts and principles by applying basic concepts.

4 New ‘Guidelines for ICT Education’

The current ‘Guidelines for ICT Education in Primary and Secondary Schools’ have been contributing to: the development of students’ basic ICT literacy; improved teaching-learning methods using ICT in subject learning, and; practical use of ICT in real life. Still, the increasingly common use of the PC and the Internet, contents behind the changing learning environments, higher expectations from the state and society, etc., necessitate the revision and supplementation of the step-specific contents.

Therefore, the new ‘Guidelines for ICT Education in Primary and Secondary Schools’ were reorganized into the following five areas, and the content framework was revised extensively.

- ‘Life in the information society’ encourages learners to behave appropriately as members of the information society by practicing information communications ethics in daily living based on their understanding of the proper use of information and information protection and expression methods.
- ‘Understanding information devices’ involves students’ acquisition of basic ICT literacy required for daily life and school learning activities by understanding

the operating principles and methods of a variety of information devices including the PC and organization of the cyber space environment.

- ‘Understanding information processing’ fosters the ability to recognize various types of information and determines efficient problem solving methods. This area also reinforces the algorithmic thinking that enables the application of ICT as well as programming capabilities.
- ‘Handling and sharing of information’ enables the creation and management of cyber space based on the ability to use the PC and understanding of the transmission and exchange of information in cyber space and creation of data expressed in cyber space and their limits.
- ‘Comprehensive activity’ fosters higher-level thinking such as creativity, problem solving capacity, and logical thinking through self-directed tasks or team projects that enable understanding the principles of information communications technology in daily life and learning activities, application of information communications technology, and participation in the information society.

The new ‘Guidelines for ICT Education in Primary and Secondary Schools,’ were announced in Dec. 2005 after they were developed based on these goals as recommended by the Ministry of Education and Human Resource Development. Based on each step-specific content framework, textbooks shall be developed, and teacher training and infrastructure will be provided. The targeted full-scaled implementation is scheduled for 2007 once teacher training and textbook development are completed.

5 Conclusions and Future Works

In Korea, the establishment of the system and educational contents for informatics education is still in progress. Identifying and seeking a solution for the problems related to informatics education and attributed to its implementation under various systems and too much emphasis on ‘ICT application education’ were prompted by the question. Can ICT education help to solve problems in learning and in daily life even without education on the principles of informatics, programming education, and algorithmic thinking? This study tried to determine the answer and proposed the result. Fortunately, the new ‘Guidelines for ICT Education in Primary and Secondary Schools’ was released in Dec. 2005, reinforcing the problem solving capacity. Still, the issue related to the system for implementing informatics education persists. The relatively short history of informatics subjects makes classifying them into independent subjects (e.g., English and mathematics) very difficult; ditto for securing a certain number of class hours for them. Moreover, the relevant policy makers neither agree to, nor oppose the matters at this time. Therefore, more research on informatics education for primary and secondary school students should be carried out to justify the need for teaching IT or ICT principles as a useful tool for solving problems and promote informatics subjects.

References

1. National Curriculum of Korea, <http://english.moe.go.kr/html/education/?menu=03>.
2. Mizkoshi, A. Study on discovery learning. Meiji Book Pub. Co., 1975.
3. Educational Policy Analysis-2004 Edition, OECD Publishing, 2004.
4. Department of Secondary & Higher Education, Government of India http://education.nic.in/ncert_it_curri_guide/it_curri_content.asp
5. A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee, ACM, 2003. <http://csta.acm.org/Curriculum/sub/k12final1022.pdf>.
6. A Model Curriculum for K-12 Computer Science Level 2 Objectives and Outlines. ACM, CSTA Curriculum Committee, 2005. http://csta.acm.org/Curriculum/sub/Level_2_Objectives_Outlin
7. Computing Curricula 2001 for Computer Science, IEEE Computer Society/ACM Task Force, <http://www.computer.org/education/cc2001/final/index.htm>.
8. England National Curriculum in Action, <http://www.ncaction.org.uk>.
9. England National Curriculum, <http://www.nc.uk.net/>.
10. NC Standard Course of Study, <http://www.ncpublicschools.org/curriculum/computerskills/>

Contribution of Informatics Education to Mathematics Education in Schools

Maciej M. Sysło¹ and Anna Beata Kwiatkowska²

¹Institute of Computer Science, University of Wrocław
F. Joliot-Curie str. 15, 50-383 Wrocław, Poland
syslo@ii.uni.wroc.pl

²Faculty of Mathematics and Informatics, Nicolaus Copernicus University
Chopin str. 12/18, 87-100 Toruń, Poland
aba@mat.uni.torun.pl

Abstract. In this paper we discuss a number of topics which are usually considered as a part of informatics education and we show how they can enhance substantially mathematics education. The presented problems may be used as a bridge between both school subjects which can integrate them and help to better understand mathematics and informatics and the relations between both disciplines.

The informatics topics discussed here belong to discrete mathematics which plays an important role in the development of efficient computer algorithms. Some of these mathematical topics are already included in some informatics curricula, however they are still absent in mathematics education. One may expect some changes when, according to the model for ICT development, school informatics and school mathematics will reach the fourth stage (ICT specialization).

1 Computer Science Topics with Mathematical Flavour

In this section we present a number of topics, which are usually considered as a part of informatics education in schools and we show how they can contribute to mathematics education. All the topics described in this paper are included in the textbook for informatics [2], addressed to high school students in Poland. However they are still absent in mathematics textbooks used by our students.

The description of each topic begins with motivations and ends with conclusions.

1.1 A Difference Between Mathematics and Computer Solution of a Problem – An Example

Motivation. A computer introduces a new dimension to problem solving – a student develops a solution which is suitable to be implemented on a computer and has good properties, such as for example low complexity and high efficiency.

We begin with an example of a problem which could be used to illustrate a difference between two its solutions, one given in mathematics class and another developed during informatics lesson.

Problem. Let set A contain a large amount of integers, e.g. 10 millions. Verify if each triple of numbers from A can be the lengths of sides of a triangle [4].

It is obvious that one has to test if each triple of numbers a, b, c satisfies the *triangle condition*, i.e. three inequalities:

$$a + b > c; \quad a + c > b; \quad b + c > a$$

Solution I (by a mathematician). The solution is very “simple”, Students suggest to test all possible triples of elements from A if they satisfy the triangle condition. If the set A contains n integers, then $c \cdot n^3$ operations (additions and comparisons), where c is a constant, are performed in this case.

Solution II (a computer solution). We assume now, that an informatics solution to a problem is a computer program which solves the problem. Students first observe that if no order is imposed on elements of A , then for a given triple of elements, each inequality has to be tested and each triple has to be checked. Then we ask them to verify that if $a \leq b \leq c$ then the triple a, b, c satisfies the triangle condition if and only if $a + b > c$. Hence, students quickly conclude that if the triangle condition must hold for all triples in A , then we have to test this condition for the two smallest integers, \min_1 and \min_2 , and the largest integer \max in A . After a short discussion students are also convinced that in fact we do not need to sort all numbers in A to find these three numbers – the following statements are sufficient:

```

 $\min_1 := \infty; \min_2 := \infty; \max := -\infty;$ 
while there is new data  $x$  in  $A$  do begin
    if  $x < \min_2$  then
        if  $x < \min_1$  then begin  $\min_2 := \min_1; \min_1 := x$  end
        else  $\min_2 := x$ 
    else if  $\max < x$  then  $\max := x$ 
end

```

Finally only one inequality $\min_1 + \min_2 > \max$ must be verified. If it is satisfied then the triangle condition is also satisfied for all triples of elements from A .

The computer solution is very efficient, at most two comparisons and two assignments are performed in each step. Thus, the total complexity is bounded by $c' \cdot n$ operations, where c' is a constant and n is the number of elements in A . Hence, Solution II is linear in the size of the input and moreover it may be applied to data set A of any size. It is not only very efficient solution but also it has a very simple implementation – the program is short and the whole solution is elegant.

Conclusion. Developing a computer solution to a mathematical problem may lead to an elegant, short, and efficient solution, not only from computational point of view but also as a mathematical solution.

1.2 Approximate Solutions – A Square Root

Motivation. Almost all real-world problems deal with real numbers (here, non-integers) and their solutions are usually approximations to the exact solutions.

A method which approximates a solution usually consists of a sequence of steps which brings an approximate solution closer to the exact solution. In school mathematics, the calculation of a square root is quite often used to demonstrate an approximation method. Let $x = \sqrt{a}$, where $a > 0$. Then starting with an initial approximation x_0 of x , successive approximations x_1, x_2, \dots , of x are usually calculated by the following iterative formula:

$$x_i = \frac{1}{2} \left(x_{i-1} + \frac{a}{x_{i-1}} \right) \quad \text{for } i = 1, 2, \dots \quad (1)$$

We have found that in the majority of mathematics textbooks for schools (at least in Polish) formula (1) is presented as a “black box”, with no explanation of its origin. This formula is a special case of Newton’s method, called also the Newton-Raphson method, which is defined by using the first derivative of a function. It would be therefore difficult to develop this method even with high school students. However, in the special case of calculating a square root, using very simple geometric arguments, it is easy to explain formula (1) even to secondary school students who know only the formulas for the area of square and rectangle, and the notion of average of two numbers (see Fig. 1).

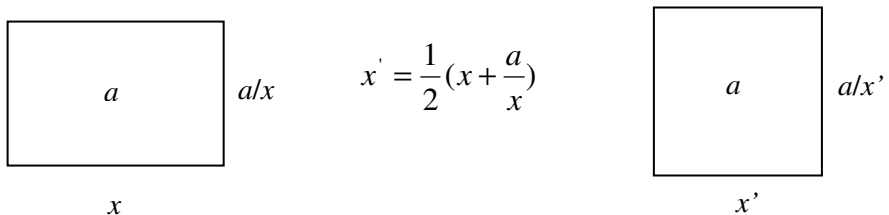


Fig. 1. Approximation of a square root

It is clear to students that if $x = \sqrt{a}$, where $a > 0$, then $x^2 = a$. Therefore they reformulate the problem: finding x means finding the side length x of a square, whose area is equal to a . If our guess for the side length x is not correct and we want to keep the area size equal to a , then the length of the other side of the rectangle should be equal a/x . In general we have $x \neq a/x$, and if x is too small then a/x is too big, and if x is too big then a/x is too small. Hence students easily conclude, that the correct answer for x lies somewhere between these two numbers, and as a next approximation we may take the average of the lengths of these two sides, hence we get the formula (1), (see also Fig. 1).

It is interesting that in a similar way students may obtain a formula for calculating roots $\sqrt[3]{a}$ – in the geometric interpretation. Instead of a square we take a cube and calculate its volume. In this case they take two sides of length x and one side of length a/x^2 to have the volume of the cube to equal a . Hence the corresponding formula becomes of the form:

$$x_i = \frac{1}{3} \left(2x_{i-1} + \frac{a}{x_{i-1}^2} \right) \quad (2)$$

In general, taking a k -cube, a cube of dimension k , students may obtain the formula for calculating roots of any order $\sqrt[k]{a}$:

$$x_i = \frac{1}{k}[(k-1)x_{i-1} + \frac{a}{x_{i-1}^{k-1}}] \tag{3}$$

Calculations according to formula (1) may be performed using MS Excel (see Fig. 2) even by students in secondary school. They can make experiments with different initial approximations and observe that only a few iterations are needed to obtain a very good approximate solutions.

A	B	C	D	C	D
Calculation of a square root of a =			2,00000000000000000000		2
Exact value of a square root of a =			1,41421356237310000000		=SQRT(\$D\$2)
Initial approximation of x =			2,00000000000000000000		2
		Iteration	Successive approximation	Iteration	Successive approximation
		Initial	2,00000000000000000000	Initial	=D6
		1	1,50000000000000000000	1	=(D10+\$D\$2/D10)/2
		2	1,4166666666666667000000	=C11+1	=(D11+\$D\$2/D11)/2
		3	1,41421568627451000000	=C12+1	=(D12+\$D\$2/D12)/2
		4	1,41421356237469000000	=C13+1	=(D13+\$D\$2/D13)/2
		5	1,41421356237309000000	=C14+1	=(D14+\$D\$2/D14)/2
		6	1,41421356237309000000	=C15+1	=(D15+\$D\$2/D15)/2
		7	1,41421356237309000000	=C16+1	=(D16+\$D\$2/D16)/2
		8	1,41421356237309000000	=C17+1	=(D17+\$D\$2/D17)/2
		9	1,41421356237309000000	=C18+1	=(D18+\$D\$2/D18)/2
		10	1,41421356237309000000	=C19+1	=(D19+\$D\$2/D19)/2

Fig. 2. Calculation of $\sqrt{2}$ in MS Excel

Conclusions. Geometric and graphical tools may help in developing and presenting numerical calculations without using more advanced tools, such as calculus. On the other hand, such methods may be used for introducing more advanced topics.

1.3 Introducing Recursive Thinking

Motivation. Problem solving using recursion is more popular in computer science than in mathematics, at least in schools. Recursion, however, is another way of looking at iteration, that is at a procedure which has to be performed a number of steps. It is maybe more difficult to introduce and to understand recursion, but as a reward students get much simpler solution methods.

Recursion is one of the main approaches for problem solving. It is especially popular in designing computer solutions, since in a program which uses recursion, much of the work is done by the computer itself and need not to be explicitly written in the program. Algorithms and programs which use recursive are usually shorter and

more “readable” than their iterative counterparts, although in general their computer execution takes much longer.

1.3.1 Fibonacci Numbers, Again

Recurrence relations are usually introduced (not only in schools) using **Fibonacci numbers**, which originally came out as the answer to the question about the population of rabbits in not too realistic a birth model. We used to introduce these numbers using another “story” (which can be found in many textbooks) together with some real-world situations, in which these numbers occur.

Here is the “story”: A professor S. has his office on the second floor and the staircase from the first floor to the second floor consists of 12 (in general n) steps. He is a very chaotic person and he takes one or two steps at a time. In how many ways can he reach his office?

Students usually start to make pictures and calculations for staircases with 1, 2, 3, 4 ... steps and obtain correct answers: 1, 2, 3, 5,... After a while, we ask them to guess the next number. If they did not hear about these numbers before, they have no idea how these numbers are changing. Then we suggest them to look at the last step of the staircase and try to answer, how Professor S. can reach this step – in this way we move our students to think recursively. They suddenly see the rule – there are two ways to reach a given step: taking either one or two steps at the end, so the following relation is satisfied:

$$b_i = b_{i-1} + b_{i-2}, \quad \text{for } i > 2 \quad (4)$$

with $b_1 = 1$, and $b_2 = 2$, where b_i denotes the number of ways Professor S. can reach step i . We therefore obtain $b_i = F_{i+1}$, where F_i is the i -th Fibonacci number.

In a similar way we can develop with students an answer to the following question: how many subsets does the set $\{1, 2, 3, \dots, n\}$ have that contain no two consecutive numbers?

Fibonacci numbers 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... look quite strange. Except for relation (4), there is no regularity in this sequence. We usually avoid to present Binet’s formula for b_i , since it contains square roots and hence does not help in understanding these integer numbers. But we are very enthusiastic about real-world objects in which one can find Fibonacci numbers, such as: pine cones (see Figure 3(a)), sunflowers, leaf arrangements in some plants, vegetables and fruits, etc. Interesting is also the relation between Fibonacci numbers and the golden ratio, so we build an approximation of a golden rectangle using these numbers (see Figure 3(b)).

1.3.2 Iteration Versus Recursion

A difference between iteration and recursion lies in the way a formula is interpreted, as shown in the table. In general, iteration is a repetition of the same procedure and recursion is a reduction of a solution of a problem to simpler versions of the same problem. Recursion may be interpreted in some cases as implementation of iteration.

Iteration	Recursion
$b_1 = 1, b_2 = 2,$ $b_i = b_{i-1} + b_{i-2}, \quad \text{for } i = 3, 4, 5, \dots$	$b_1 = 1, b_2 = 2,$ $b_i = b_{i-1} + b_{i-2}, \quad \text{for } i > 2$

Iteration means that numbers b_i are calculated in the order: $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots$. On the other hand, recursion is interpreted as follows: if we want to calculate b_i , for a given i , then if $i = 1$ then $b_1 = 1$, if $i = 2$, then $b_2 = 2$, and if $i > 2$, then $b_i = b_{i-1} + b_{i-2}$, therefore to calculate b_i we have to find b_{i-1} and b_{i-2} in a similar way using the recursive formula.

In the above explanation we can use a sequence which is defined as an arithmetic progression, for instance $a_i = a_{i-1} + r$. Another good example is a recurrence relation derived from the Hanoi Tower puzzle, $h_i = 2h_{i-1} + 1$ for $i > 1$ and $h_1 = 1$.

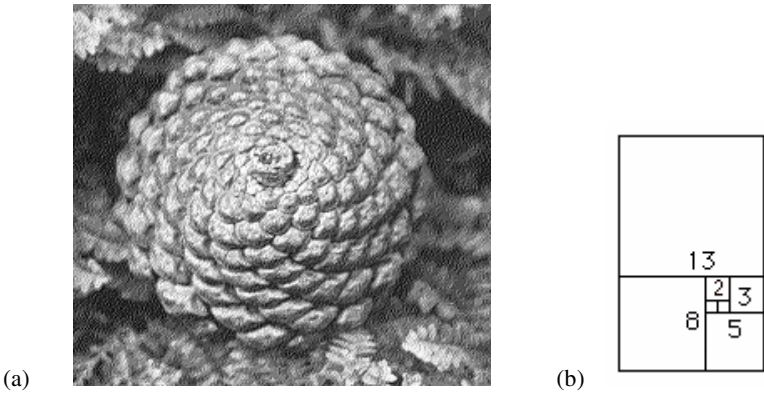


Fig. 3. (a) A pine cone – one can easily count that there are 13 “left” spirals and 8 “right” spirals” (b) An approximation of a golden rectangle

Conclusion. Maybe recursion and a recursive way of thinking and solving problems is the most solid and concrete bridge between informatics and mathematics education. We are convinced that recursion introduced while developing computer solutions in informatics classes may contribute also to mathematics education and in particular to solving pure mathematical problems.

1.4 Introducing Logarithm

Motivation. Logarithm is a very important function in computer science, not only because “logarithm” is an anagram of “algorithm”. It appears as a measure of data size in computer memory and more important, it describes the number of steps in a divide and conquer approaches for problem solving.

We first present three examples related to **logarithmic functions** in which, however, we avoid to use any reference to this function. Then we propose an informal method of introducing the concept of logarithm.

Example 1.4.1. For a given nonnegative integer number n find its **binary representation**. Students easily find that such a representation is generated in a successive divisions of n and the resulting quotients by 2. They divide n by 2 and take the remainder r (0 or 1) as the least important digit of the representation. Then, apply this procedure to the quotient q and continue as far as the quotient is nonzero.

For $n = 77$ we have $(1001101)_2$. Then we ask our students, how many binary digits has a decimal number n in its binary representation or equivalently, how much space in the computer memory we need for storing number n .

Example 1.4.2. A telephone book consists of 1000 pages. Find the page which contains the telephone number of Mr. Smith M.M. checking the smallest possible number of pages. Searching for a right page, students discover (or they already know) that the best strategy is a **binary search**, that is to keep splitting the remaining pages into two equal parts and to eliminate the part which does not contain the entry with Smith M.M., until only one page remains, which should contain Smith's telephone number.

n	q	r
77	38	1
38	19	0
19	9	1
9	4	1
4	2	0
2	1	0
1	0	1
0		

We ask our students, how many pages they have checked and what is the smallest such number.

Example 1.4.3. (More advanced) Use **Euclid's algorithm** to find the greatest common divisor of two given numbers n and m and try to find two number n and m for which Euclid's algorithm performs the largest number of steps (a step consists of calculating the remainder of two numbers). After students make some guesses we propose them to consider $n = 55$ and $m = 34$, or $n = 34$ and $m = 21$. Then ask them to compare the numbers in columns 1. and 3. They should notice that in the same row, the number in the third column is at least twice smaller than that in the first column. (Students should also notice that 21, 34, 55 are Fibonacci numbers!)

n	m	r
34	21	13
21	13	8
13	8	5
8	5	3
5	3	2
3	2	1
2	2:1	0

One may ask, what these three examples have in common. The answer is suggested in the course of performing the calculations. Starting from a given number (n – to be represented binary, number of pages in a telephone book, numbers n and m in Euclid's algorithm) we divide this number by 2 until it becomes equal 1. For instance, if $n = 1000$, then our students get the sequence (the result of division is round up):

1000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1

Therefore, number 1 is obtained after **10** divisions of n and successive quotients by 2, what is an approximate value of $\log_2 1000$ (exactly, we have $10 = \lceil \log_2 1000 \rceil$, the smallest integer not smaller than $\log_2 1000$).

Conclusion. One may define the value of $\log_2 n$ algorithmically as the number of steps in which, successive divisions of n and the resulting quotients lead to 1.

1.5 Polynomials, Number Systems, Horner's Rule and Its Applications

Motivation. Without going into details we explain to our students that computers (processors) perform only 4 basic arithmetic operations. Calculation of any other operation or function must be performed as a sequence of these four operations. Polynomials, whose values can be computed using only these four operations, play a very

important role as functions which can be used to approximate any other continuous function. We show below that polynomials are also important in arithmetic operations on integers.

Polynomials appear in school mathematics mainly in linear and quadratic equations. Indirectly, polynomials of higher degrees are used to represent numbers in **a positional system** to any base. We have for instance:

$$(77)_{10} = 7 \cdot 10^1 + 7 \cdot 10^0, \quad (77)_{10} = (1001101)_2 = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

and in both cases the number representation is **a polynomial** in the base of the system with coefficients from the set of digits of the system. We then introduce (or develop) **Horner's rule** and apply it to such polynomials – for our binary example we have:

$$\begin{aligned} (1001101)_2 &= 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \\ &= (((((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1. \end{aligned}$$

There are a number of useful observations based on this representation of a number.

1.5.1. We ask our students to calculate the decimal value of a binary number given in another positional system without using a single memory (see Fig. 4). In fact, this is the fastest possible algorithm (it was proved that Horner's rule is optimal).



Fig. 4. Computing decimal value of $(1001101)_2$ using a regular calculator with no memory

1.5.2. Fast calculation of a power x^n is a crucial step for instance in the RSA cryptographic schema, especially for large exponents. Using Horner's rule representation of exponent n it is easy to devise a simple algorithm which computes x^n efficiently. Let us use our example:

$$\begin{aligned} x^{77} &= x^{((((((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1)} = (x^{((((((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1)})^2 \cdot x = ((x^{((((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1)})^2)^2 \cdot x = \\ &= (((x^{((1 \cdot 2 + 0) \cdot 2 + 0) \cdot 2 + 1)})^2 \cdot x)^2 \cdot x = (((x^{(1 \cdot 2 + 0) \cdot 2 + 0})^2 \cdot x)^2 \cdot x)^2 \cdot x = (((((x^2)^2)^2 \cdot x)^2 \cdot x)^2)^2 \cdot x \end{aligned}$$

Therefore, to compute power x^{77} we calculate the following sequence of powers:

$$x^2 = x \cdot x, x^4 = x^2 \cdot x^2, x^8 = x^4 \cdot x^4, x^9 = x^8 \cdot x, x^{18} = x^9 \cdot x^9, x^{19} = x^{18} \cdot x, x^{38} = x^{19} \cdot x^{19}, x^{76} = x^{38} \cdot x^{38}, x^{77} = x^{76} \cdot x.$$

using only 9 multiplication. It is quite easy to observe that the number of multiplications depends on the length of the binary representation of the power and is equal to the number of binary positions in the representation minus 1 plus the number of 1's in the representation minus 1. Since the length of binary representation of n is proportional to $\log_2 n$, the number of multiplication needed to calculate x^n is also proportional to $\log_2 n$.

To compute x^n we may also apply a recursive procedure, such that $x^n = (x^{n-1}) \cdot x$, when n is odd, and $x^n = (x^{n/2})^2$, when n is even. It is an interesting exercise for students to verify that this procedure generates the same sequence of powers as the method based on Horner's rule, described above.

Conclusion. Polynomials and their properties and algorithms play an important role not only in approximating continuous functions, but also in designing and performing operations and algorithms on integer numbers, especially using representations of integers to different bases.

1.6 Finding the Best and the Second Best Player

Motivation. As shown in 1.4, if information is kept ordered then we can much easier perform basic operations on a set of elements: searching, adding a new element and preserving an order, and removing an element. Finding the best (maximum) element is a special case of a search. It is one of the most frequently used “operations” in every day life. Therefore it should be interesting to know how efficiently we usually perform this operation.

To find the best element (item) among a number of elements we usually go through all elements keeping at hand the best element found so far. Therefore, if there are n elements, we begin with the first element at hand and then make $n - 1$ comparisons, and sometimes exchange the best element found so far (at hand) with a better current element.

This is a **linear search**, in which we check all elements as if they stand in a line. Students always argue, that if a tournament is played then the number of comparisons (matches) is smaller than that in a linear search, so we ask them to play tournaments with different numbers of players. After a while they find that in a tournament type competition there are the same number of matches played (see Fig. 5(a)) – Wojtek is the best player among 8 competitors and 7 matches have been played in this case.

At this point of a lesson we try to convince our students that in fact $n - 1$ is the smallest number of comparisons needed to find the best item (element, player) among n items. To this end we use a very simple and beautiful argument given by Hugo Steinhaus in his excellent book *Mathematical Snapshots* [5]: if Wojtek appeared to be the best player then each of the other players had to loose at least once (to Wojtek or to other players). Therefore at least $n - 1$ matches were played in the tournament (we assume that there are no ties). Thus, a linear and a tournament search for the best element are the **optimal algorithms** since they perform exactly $n - 1$ comparisons.

As an extension, we then ask our students, whether a looser in the final of a tournament is the second best player of the tournament? Students easily notice that if Wojtek is removed from the tournament tree (see fig. 5(b)), then Bartek, who lost in the final game, has to play against competitors in the subtree Wojtek came from. Therefore Bartek must be placed in the beginning of Wojtek’s tournament route and only matches along this route, except the last one, have to be played again. Therefore, $\log_2 n - 1$ matches have to be played to find the second best player, since a tree of a tournament among n players has height $\log_2 n$. Hence, $n + \log_2 n - 2$ matches (comparisons) are needed to find the best and the second best player among n players (elements). At this point we inform students that this is also an optimal algorithm with respect to the number of matches played.

We have demonstrated that a tournament tree, and in general, a search tree, is a very useful tool in approaching questions concerning information search, in designing solution methods, and in analyzing their properties.

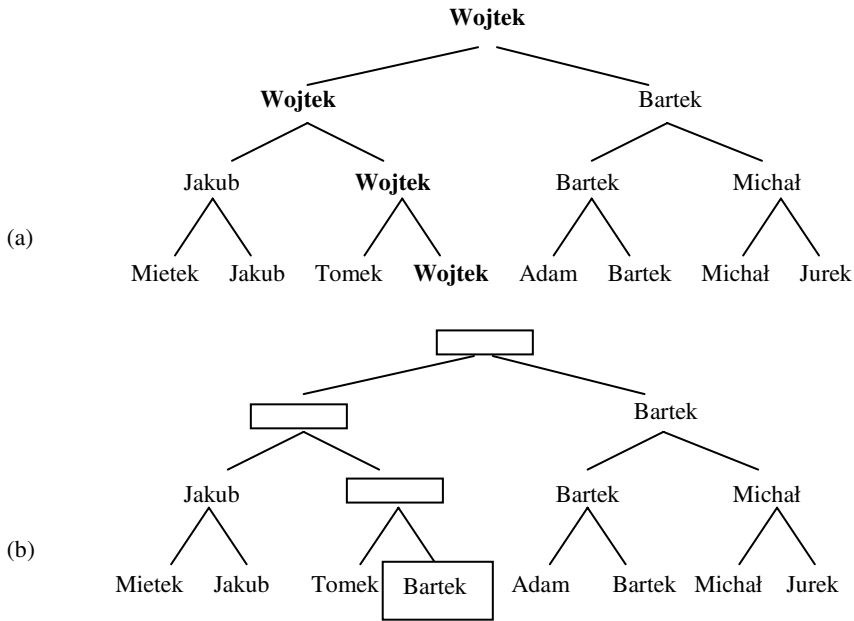


Fig. 5. (a) A tournament among 8 players with Wojtek as the winner. (b) The second stage of the tournament to find the second best player.

Conclusion. Finding the best (or the worst, or maximum, or minimum) element in a collection of elements is probably the simplest problem which can be used to evaluate a solution method with respect to its complexity and optimality, and also its practical efficiency. With such an example, students are able to touch a very important area of computational complexity of algorithms. Although computers appear to our students as very fast machines, it is a nice opportunity to convince them that *the best way to make computers go faster is to make them work with less steps* (Ralph Gomory, IBM).

2 Conclusions

We used a number of topics, typically regarded as a part of informatics education, to show how they can enhance substantially mathematics education in the area of discrete mathematics, known also as *mathematics of our time* and as *mathematics of the computer era*.

All the topics discussed here are present in the textbook for informatics [2] used in high schools in Poland. This textbook meets the curriculum and evaluation standards for school informatics (at the high school level) approved by the Ministry of Education. One can also find these topics in Level III (*Computer Science as Analysis and Design*) in *A Model Curriculum for K-12 Computer Science*, published by ACM in 2003 [1] and also at Stage IV (*ICT Specialization*) in *A Curriculum Structure for Secondary Schools*, published by IFIP/UNESCO in 2002 [3].

Most of the topics presented in this paper are, however, still absent in mathematics education (in Poland). They are missing in the curriculum and evaluation standards for school mathematics, in textbooks, and in mathematics teachers preparation. One may expect some changes in mathematics education when, according to the model for ICT development [3], informatics education and mathematics education will be thought of as interrelated and integrated subjects. This will happen when mathematics education will pass the first stage (*Discovering ICT Tools*) and the second stage (*Learning How to Use ICT Tools*) and reach the third stage (*Understanding How and When to Use ICT Tools*) to use informatics tools to achieve particular goals in mathematics education. The most extensive and deep integration of both subjects is expected at the fourth stage of ICT development in education (*ICT specialization*) when discrete mathematics will play an important role in developing computer solutions to real-world mathematical and computational problems.

We expect that reaching the third and fourth stages in both, educations in informatics and education in mathematics, will be facilitated in Poland due to a special program of teachers training at university level. Future teachers graduating in mathematics or in informatics have to take both specializations: teaching mathematics (didactics of mathematics) and teaching informatics (didactics of informatics), and also a course on using ICT tools in mathematics education.

References

1. ACM K-12 Task Force Curriculum Committee, A Model Curriculum for K-12 Computer Science, ACM (2003)
2. Gurbiel E., Hard-Olejniczak G., Kołczyk E., Krupicka H., Sysło M.M.: Informatics (In Polish), Vols. 1 and 2, Textbook for high school, WSiP, Warszawa (2002-2003)
3. Information and Communication Technology in Education. A Curriculum for Schools and Programme of Teacher Development, IFIP/UNESCO (2002)
4. Polish Olympiad in Informatics (in Polish), Warszawa (1993/1994)
5. Steinhaus H.: Mathematical Snapshots, G.E. Stechert Press (1939); 3rd edition, Oxford University Press, New York (1969)
6. Sysło M.M.: Algorithms (in Polish), WSiP, Warszawa (1997)

Evolution of Informatics Maturity Exams and Challenge for Learning Programming

Jonas Blonskis¹ and Valentina Dagienė²

¹ Kaunas University of Technology
Sukilėlių 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. Informatics as a compulsory course of comprehensive school curricula has been taught for three decades. The maturity exam in informatics was established in 1995, although its content has been changed several times since then. This paper analyzes the conception of the maturity exams in informatics and provides explanations of the examination's structure, content, and goals. Exams in informatics have a double mission: to evaluate students' knowledge and skills and to encourage teachers and students toward the study of informatics. Considering that programming as a branch is fairly important to the state's economy, the decision was made to establish a special exam in programming. This paper presents the results of the pilot exam in programming.

1 Background

The compulsory course in informatics in Lithuanian comprehensive schools began in the school year 1986–1987. Theoretical content dominated then; the main attention was paid to enhancing the understanding of algorithms and programming skills [5]. In the school year 1994–1995 informatics was incorporated into the elective maturity exams' block of sciences together with exams in such disciplines as biology, chemistry, and physics. After nine years of the existence of informatics in schools, the discipline has matured and therefore it was decided to establish a maturity exam in informatics.

When creating the informatics course in compulsory schools and seeking to evaluate properly the achievements of students, both, existing curricula and their accordance with the general principles of Lithuanian education were taken into account. [12, 13]. Discussions took place on what kind of evaluation of the students' achievements in the information technologies (IT) field should be considered as most relevant and would have the most positive impact on the whole system of teaching and learning. It is worth mentioning such miscellaneous positive optional activities in the field of IT as workshops, competitions or contests. While such special events as the Olympiads in Informatics or Logo contests should also be mentioned, this is an aspect of informal teaching. Exams are a more natural feature of formal teaching.

In 1995, the first maturity exam in informatics was created. Later the achievements, advantages and disadvantages were analyzed [4]. Until 2002, the school exams in informatics took place each year and had more or less the same structure. In 2002–2003, the curricula for comprehensive schools in Lithuania as well as the education standards were essentially revised and changed [8, 9]. Thus, a need has arisen to reassess and to formulate new exams in informatics.

The mission of informatics as a separate subject (which later was renamed information technologies) is ternary. First of all, attention is paid to enhancing and nurturing the information culture. This should be taught during lessons in all subjects, while the course in information technologies must systematize and summarize it.

Secondly, the goal is being set for all students to gain computer literacy and integrate the use of computer technology when learning any subject, as well as applying it for any other need.

Thirdly, the IT capabilities of each student have to be developed with a particular emphasis on programming. This field has substantial significance for the Lithuanian economy and can help students make choices concerning their future occupation.

Considering this, as well as the aspects of teaching this subject on different levels, the maturity exams in informatics are focused on evaluation of achievements in the information culture, assessment of students' informational abilities, and their computer literacy. The exams are specific for each school.

Taking into account the needs of higher education institutions and the economical development of the country, the maturity exam to evaluate students' skills in programming was established. A model exam was prepared and, in spring of 2005, the pilot exam was completed. Since the pilot was so successful, the exam in informatics programming will be given nationwide this year.

Thus, in Lithuanian comprehensive schools there are two types of exam in informatics (information technologies) at the moment: a school exam that covers the general curriculum (in fact, it continues in the direction taken in 2002) and a national exam that assesses knowledge of the programming module provided in the advanced course of informatics.

2 Goals and Objectives

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 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 discipline.

In the school exam in IT, the goal of enhancing and nurturing the information culture is the most important. This goal is emphasized in the general curricula for both

basic and secondary schools [5]. Furthermore, the exam includes more specific goals such as computer literacy, development of students' skills, generalization and improvement of knowledge in the field of information and communication technologies (ICT), and the understanding of the terms of informatics.

The notion of computer literacy fluctuates. In Lithuania, computer literacy is very often associated with ECDL – European Computer Driving Licence [6]. The Lithuanian Computer Literacy Standard shifts this association towards ECDL even more. Even though the ECDL requirements were criticized for being unnecessarily technical, they are formulated specifically and efficiently, and therefore are a fair basis upon which to obtain a primary acquaintance with the computer. In order to check computer literacy a course credit was introduced.

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 recourses of our country. However, programming is not an easy occupation: it requires much effort and obtaining 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.

In other words, the goal of the programming exam is to evaluate the students' ability to model different problem situations in algorithms, and to provide the possibility for them to show their computer skills and knowledge in programming.

3 Changes of Models of the Informatics Exam

During the first five years (1994–1999), the developers of the maturity exam in informatics followed the very same conception which strictly corresponds to the general curriculum for informatics. The exam had the same structure each year and contained two theoretical questions and three problems.

Students were acquainted with the formulation of topics in advance. The exam questions were formulated in a more concrete or constricted way, nevertheless their character was broad enough, requiring reasoning and a fair knowledge of the field. The topics embraced the compulsory course and the main conceptions associated with common human information activity. By answering the questions, students not only showed their knowledge in a certain field but also had a chance to demonstrate their ability to express thoughts in a clear, logical and correct manner, all of which have essential importance in the society based on ICT. However, after carrying out a survey of teachers-consultants, the majority of them disapproved of some tasks. Instead of the essay they suggested a test [3].

The practical part of the exam consisted of three problems, two of them paper-based and one – computer-based. The first two problems were similar enough to mathematical problems. Their content was related to algebraic logic (rearrangement of logical expressions, applications of logical law, and algorithmic schemes), calculation systems and the main concepts of information theory (measurement of information and encoding).

The third problem was related to algorithms. The solution had to be written in programming language symbols, usually performed with a computer and tested with various data.

Since 2000, when the profile of teaching was introduced into schools, the course in informatics has essentially changed. The course in informatics was based on the same principles as any other subject in profile (humanitarian or science) secondary schools. The content of profile enhancement is expressed by levels: general and advanced.

When drafting the exam, long discussions took place concerning which level should serve as basis. Since it is a school exam the decision was made to follow the content of a general level course. In addition, the content of this level seemed to be homogenous, which meant – more acceptable for an exam.

The goals of the exam in informatics were formed; these goals embraced two main trends. One is focused on the user. It is more practical and cognitive and is intended to evaluate the students' skills for using ICT. Another is focused on programming skills and it is intended to evaluate the professional interest of students in the basics of computer operation. Since the content of informatics teaching has become broader, both trends appeared to be too difficult to evaluate in one exam. Therefore, two models of the maturity exams in informatics were developed: 1) one in information technologies and 2) another in programming.

The content of the exam in IT is based on the general curriculum of information technology for grades 11–12. Since the course is quite modest, only 70 hours, the exam is fairly compact. The scheme of the exam is prepared according to five teaching topics (*i.e.* text formatting, usage of spreadsheet, Web, and preparation of presentations, as well as social and ethical aspects). When drafting the content, the recommendations of world experts were taken into consideration [1, 10, 11, 14].

The programming exam is based on the module of the advanced course, which may be chosen by students in 11th grade [2]. The module consists of 70 hours. The teaching of programming embraces four main fields: 1) basic constructions of Pascal; 2) data structures; 3) algorithms; 4) a version of the Pascal language in a Free Pascal environment.

4 Pilot Exam in Programming

The pilot exam in programming was completed in the spring of 2005 with one grade lower than the graduates (*i.e.* 11th grade). In the 2004–2005 school year, there were 55287 students in 11th grade in 704 Lithuanian comprehensive schools. Of that number, 3932 of them (from 367 schools) intended to take the national exam in programming, however just 1064 (from 114 schools) registered and the exam was taken by 973 students from 112 schools.

The exam consisted of a test (11 questions of IT and 10 questions of programming) and two programming tasks. The programming task – to write a program for the given problem – appeared to be the most difficult to solve. 730 of the students tried to solve the programming tasks: 702 of them undertook the first task and 417 – the second one. Both tasks were undertaken by 389 students. Since it was a pilot exam, some students didn't try to solve the problems at all.

260 students received more than zero points for each programming task. 203 students received more than 10% of all points (*i.e.* from 6 to 50). 9 students scored all possible points (*i.e.* 50).

4.1 Content of Exam Curriculum

In Lithuanian schools, each subject's exam has its own curriculum, which is more concrete than the general subject's curriculum. The programming exam curriculum closely corresponds to the content of the module. Three main fields are emphasized: algorithms, data types and structures, and constructs of a programming language (Table 1).

Table 1. Components of curriculum of programming exam

Algorithms	Data structures	Programming language (Pascal)
Calculation of sums (of product, quantity, and arithmetical average). Search of the maximal (minimal) value. Data input/output. Data sorting. Ability to modify algorithms according to the particular data structures	Integer and real, char, boolean, and string Text file. One-dimensional array. Record. Ability to create uncomplicated data structures.	Program structure. Commentary. Variables Assignment and statement. Relational and logical operations, if statement Loops. Compound statement; Procedure and function. Lists of parameters and arguments. Standard mathematical procedures and functions. Procedures and functions related with files.
Programming environment. Technology of structural (procedural) programming. Testing. Program documentations. Arrangement of dialogs. Program writing (style)		

The programming test (25%) is intended to evaluate students' knowledge and perception of tools required for programming (*i.e.* programming language, data structures, and algorithms). The proportion of the test questions is the following:

- control structures, elementary data types, and text files – 10%;
- procedures and functions – 5%;
- arrays – 5%;
- record and array of the record type – 5%.

In the test, questions from the programming language dominated as well as questions from data structures, integer, real, boolean, array, and text (Table 2). The exam's tasks covered half of the material of the optional programming module. The exam did not include (neither in test nor in problems) tasks from: char, strings, and records, variables of record type and arrays, modification of algorithms according to particular data structures, development of data structures.

Table 2. Programming concepts in tests

Test question	1	2	3	4	5	6	7	8	9	10
Score for test question	4	2	2	2	2	2	2	5	2	2
Assignment	+	+	+	+		+	+	+		+
For	+		+					+		
While			+			+	+			+
Compound statement			+					+		+
If		+				+	+	+		+
Logical operations (and, or, not)		+							+	
Structure of program		+	+	+	+			+	+	
Write, WriteLn		+	+	+		+	+	+	+	
Read, ReadLn							+	+	+	
Data input from file							+			
Array	+		+					+		
Procedure, call					+					+
Function				+				+		

4.2 Writing Programs

During the exam in programming, students had to write programs for two problems. The tasks consisted of several parts. Their goal was to evaluate students’ practical skills, the ability to choose and to create data types for the tasks, as well as to apply algorithms and tools for program structure in particular situations. The students’ abilities to program are described in Table 3. It may be concluded that not all of the students who passed the exam are able to develop programs; nevertheless they have some knowledge and idea of programming.

Table 3. Students’ abilities to write programs

Have seen	Are acquainted	Understand	Have learned
1 %	11 %	21 %	31 %

The first task was much easier than the second one. In order to accomplish it students had to be able to read numbers from a file and to write into a file those numbers that were the squares of the even numbers. They also had to calculate how many of such numbers were found. The goal of this task was to evaluate the primary skills and basics of programming knowledge.

The second task was intended to assess the students’ knowledge and ability to process numeric arrays, to calculate procedures, and functions. The calculations here are more complicated. The goal was to determine whether the main module’s topics such as array, procedures, functions, and work with files were learned.

We examined the programs of the second task as well. From 62 correctly running programs, just 20 were written according to the guidelines given in the task. The majority of program tests show that students have a rather different understanding of

Table 4. Comments on students' programs

Basic comments	Number of programs
Program is written properly and runs correctly. "Trifles": global parameters of the Loops, local array type, irrational use of records, output file is being opened during the procedure of data input, etc. In one program both functions were recursive and correctly running.	34 (20 of them were fully correct)
Program runs correctly. There are no procedures or functions asked in the task, they are different, there is just part of them, or there are just procedures. There are small mistakes (e.g. the result will be wrong if data contain just one value or the first value is the requested one). Sometimes it happened that the function asked for in the task was written as if they were two. There were rare mistakes of value parameters and variable parameters in procedures and function headings. In several programs the values t1 and t2 of the data file were not used and, instead, the concrete values of the given example were written.	28
Type of array and all variables described correctly. There is not any other text in the program.	25
Both files are correctly prepared for the running.	15

Table 5. Program evaluation criteria

Evaluation criterion	%
Program test result. The number of points gained depends just on the correct results.	23
Program according to the task. File processing, algorithms of the procedures and functions, procedures and functions headings parameters list and arguments in a call, structure of the program are evaluated in points.	50
Data types and variables. Suitability of the data types for the task, their correctness and rationality. Objective fitting of the variables, their place in the program, validity of the auxiliary variables and their place in the program.	17
Style of the programming. Accordance with spelling rules, the commentary on the essential parts of the program, structuring of the program's text layout, comprehension of the names of the constants, types and variables as well as their suitability for the task.	10

programming, practical skills and reasoning ability. Some students did not pay attention to the guidelines when reading the task. During an exam, after all, it is important to demonstrate the ability to apply the obtained knowledge rather than to

simply write a correct program for the given task. In Table 4 some of the comments and related numbers are given.

4.3 Evaluation Criteria

Some criteria for the evaluation of programs were developed (Table 5). The proportions of the points’ allocated for each particular task were shifting in a rather narrow range, depending on the particular complication and extent of the task.

5 Some Remarks on the National Exam

After the analysis of the pilot exam and evaluation of the remarks provided by teachers, the following conclusions were made.

1. Tasks of the exam (test questions and programming tasks) are oriented towards knowledge in the programming module. They are understandable and do not require deep knowledge in other subjects (*i.e.* mathematics, physics, chemistry, etc.).
2. The scheme of attention assignment is suitable;
3. The complexity of the programming tasks is appropriate;
4. The first task of the national exam will be an analogue to the second one of the pilot exam. The second task of the national exam will be intended to evaluate abilities to use the data type record. The proportional allocation of the evaluation points will remain the same, nevertheless, tasks should not be too complicated, taking into account the time limits of the exam;
5. When elaborating the evaluation of the programs by points main attention should be paid to the algorithmic element (Table 6).

Table 6. Evaluation of program development

Parts of program development	Points (%)
Analysis and development of data structures	20
Algorithms of data processing	40
Programming constructions	10
Programming environments	10
Programming technology	10
Debugging and testing	5
Programming style	5

6 Perspectives

The conception of the maturity exam in informatics or, as it is named now, in information technologies, has been changed several times since this compulsory subject was introduced in comprehensive schools. A rather theoretical exam in

informatics was shifted to a more practical one; a substantial part in information technologies is tested. Programming, which some times ago was just one part of the exam, has developed into a separate exam. Also, tasks in the nature of a test were introduced as well.

Although the main goal of the school exam in information technologies is to evaluate students' knowledge, the additional goals, such as influencing the appropriate students' future choices, are relevant here as well.

According to the draft of the maturity exam in IT, prepared in 2002–2003, two elective exams were introduced: the school exam and the national exam. The school exam consists of a general curriculum of the information technologies course and the main attention is paid to the practical tasks related to text and tables creation by using a spreadsheet. The national exam in programming is being prepared according to the advanced module of programming and will be launched in 2006.

Since software owned by schools is different and the knowledge level of informatics teachers is sometimes inadequate, the preparation of the tasks for the exam's general part is the most problematical question.

The most important part of the exam in programming is, obviously, the evaluation of the programs. 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 the second task, students had to write and use a function that finds the highest value in a number array.

Sometimes even in simple operations one might find alternative algorithmic realizations. It is very important for assessors to "behold" the essential parts in programs, as well as their correspondence to the requirements, and to be at their most objective during the evaluation.

References

1. Anderson, J., Weert, T.: Information and Communication Technology in Education. A Curriculum for Schools and Programme of Teacher Development. Unesco, 2002
2. Dagienė, V., Blonskis, J.: Teaching of programming in advanced course of informatics. Lietuvos matematikos rinkinys, 2002, 42 t., spec. nr., 229–234 [in Lithuanian]
3. Dagienė, V. Alternation of concepts of Informatics matura exam. Informacijos mokslai, Vilnius, 2001, vol. 16, 39–47 [in Lithuanian]
4. Dagienė, V. The first Informatics matura exam. Informatika, Vilnius, 1995, N. 24, 7–23 [in Lithuanian]
5. Dagiene, V.: The Model of Teaching Informatics in Lithuanian Comprehensive Schools. Journal of Research on Computing in Education, vol. 35, N. 2, 2002–2003, 176–185.
6. ECDL: European Computer Driving Licence. Available at: <http://www.ecdl.lt/modules/news/> [accessed 2006-06-12]
7. Gage, N. L., Berliner, D. C. Educational psychology. Boston, Houghton Mifflin com., 1992.

8. General curriculum for general education school in Lithuania and general education standards for grades XI–XII, 2002. Vilnius, Ministry of Education and Science [in Lithuanian]
9. General Curriculum and Education Standards: Pre-school, Primary, and Basic Education. Vilnius, Ministry of Education and Science, 2003 [in Lithuanian]
10. Information and communication technologies in education: A planning guide / Eds. T. van Weert, and J. Anderson. UNESCO, 2002
11. Ipsen, A., Thorslund, J.: Curricular Reform and Life-skills in Denmark. Curriculum Change and Social Inclusion: Perspectives from the Baltic and Scandinavian Countries. Final Report of the Regional Seminar, Unesco, 2002, 64–69
12. Conception of Education in Lithuanian. Vilnius: Leidybos centras, 1992 [in Lithuanian]
13. Strategic attitudes of education development in Lithuania (project). Švietimo gairės. 2003–2012. Vilnius, Švietimo kaitos fondas, 2002 [in Lithuanian]
14. Pelgrum, W.J., Anderson, R.E. (eds.): ICT and the Emerging Paradigm for Life Long Learning: a Worldwide Educational Assessment of Infrastructure, Goals, and Practices. Amsterdam, IEA, 1999.

Objective Scoring for Computing Competition Tasks

Graeme Kemkes, Troy Vasiga, and Gordon Cormack

University of Waterloo
Waterloo, Ontario, Canada
{gdkemkes, tmjvasiga, gvcormack}@uwaterloo.ca

Abstract. Computing competitions like the International Olympiad in Informatics (IOI) typically pose several problems that contestants are required to solve by writing a program. The program is tested automatically on several sets of input data to determine whether or not it computes the correct answer within specified time and memory limits. We consider the controversy of whether and how to award partial credit for programs that fail some of the tests. Using item response theory, we analyze the degree to which the scores from these automatic tests, separately and in various combinations, truly reflect the contestants' achievement.

1 Introduction

The International Olympiad in Informatics (IOI) [1] is a programming contest for secondary school students. Competitors are given a set of tasks (typically six) for which they are to program solutions. The solutions are subject to a number of automatic tests; each test presents the program with a set of data and evaluates automatically whether or not the program produces the correct output within specified time and memory limits. The vast majority of contestants' submissions fail at least one test; a substantial fraction (one-third or more) fail at least one test run for every solution that they submit.

The question we address here is: how should failed tests be scored? Our thesis is that test runs should be categorized according to objective criteria, that points should be awarded according to category, and that a program failing even one test in a category should be awarded no points for the category. Our investigative approach demands careful attention to the difficulty of each category; more precisely to the *discrimination* of each category – how much more likely is a contestant with more ability than another to be able to prepare a submission that achieves points in the category? We argue that current practice yields tests with too little diversity of difficulty – too difficult for most and too easy for a few – and hence with poor discrimination among the majority of contestants. We argue that the solution is not to increase the median level of difficulty, as was the trend for several years. We argue that current practice of awarding marks in proportion to the number of tests passed impairs discrimination and underrepresents the difficulty of the tasks. Further, we propose an alternative testing policy which yields much better discrimination ability.

Each test run uses data selected in advance by the contest designers to test various possible sources of error or inefficiency. The data for the test runs is secret; the contestants may not see it before their programs are scored. Prior to IOI 2004, contestants were informed only of the criteria necessary to achieve full marks; that is, the correctness criteria, the maximum input data size, and the memory and time limits. Submissions that were incorrect or inefficient – the vast majority of submissions – could be expected to receive a partial score to the extent that some test cases would fail to detect incorrect or inefficient programs. A contestant submitting an incorrect or inefficient program had no way of predicting what partial score it might achieve, and therefore little guidance as to how to try to maximize his or her score. To mitigate this problem, the *fifty-percent rule* was implemented in 2004; it states that (weaker) criteria must be specified in the task statement which, if met, will yield a score of at least 50%. Notwithstanding the fifty-percent rule, contestants still experience a great deal of uncertainty as to the score that they might expect to receive for a particular effort.

In a previous study, Cormack [3] suggests that partial scoring be eliminated; that a score should be awarded for a particular group of tests (that is, all the fifty-percent-rule cases for a given task, or all the non-fifty-percent-rule) only if the program passes all of them. Such scoring appears to discriminate better among competitors with high ability, but fails to discriminate at all among the bottom third, who receive a score of 0 under the proposed scheme. It has also been suggested that such all-or-nothing scoring excessively penalizes contestants who make one small mistake, resulting in a zero score for a group of tests. We propose an alternative scheme which accomplishes discrimination at both the upper and lower ability levels.

2 Overview of Item Response Theory

Item response theory (IRT) investigates the ability of a test to discriminate among students with various levels of ability. In contrast, classical theory (and common practice) considers only the difficulty of a test case – as evidenced by the overall success rate – in assessing its contribution to scoring. According to item response theory, a test is composed of several items, each of which is graded as correct or incorrect. For each item, the probability that a student answers correctly is modelled as a function of her ability θ ,

$$P(\theta) = \frac{1}{1 + e^{-a(\theta-b)}}.$$

This function is called the *item response curve*. The parameters a and b depend on properties of the item: a is the discrimination parameter, b is the difficulty parameter. A good test contains items with a variety of difficulty parameters, covering the range of abilities of the students. The discrimination parameter of each item should be high.

These parameters have a quantitative interpretation. The difficulty parameter is the ability level at which the probability of success is 0.5, which is also the inflection point for this curve. The slope of the curve at this point b is $a/4$, which

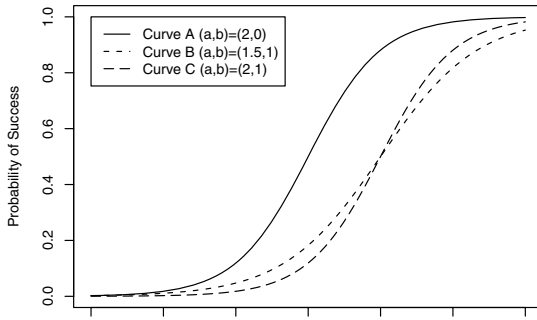


Fig. 1. Item Characteristic Curves

is simply a constant factor times the discrimination parameter. In other words, we concern ourselves with describing the item characteristic curve in terms of the inflection point and the slope at the inflection point. Some sample item response curves are shown in Fig. 1. These are three item characteristic curves (A, B, C) such that Curves A and C have the same discrimination parameter (i.e., slope at their respective inflection points), Curves B and C have the same difficulty parameter (i.e., inflection point), Curve A has a lower difficulty parameter than Curve C (i.e., the curve is shifted to the left), and Curve B has a lower discrimination parameter than Curve C.

For more information about IRT, see the introductory textbooks [2,5].

3 Scoring Schemes

As stated in the introduction, IOI 2005¹ had 6 problems for students to solve. Each problem had two batches of test cases: *Easy* test cases (the so-called “50% rule” cases), and *Hard* test cases.

We will analyze the IOI 2005 results using three different scoring schemes. These are

1. IOI scoring: as at the IOI, each test case is graded as correct (4 or 5 points) or incorrect (0 points). The score for each batch is simply the sum of the scores of the test cases in that batch. The perfect score on each batch is 50.
2. All-or-Nothing scoring: for each batch, we assign a score of 50 if the program produced correct output on every test case in that batch; otherwise, we assign a score of 0.
3. Significant Progress scoring: for each batch, we assign a score of 50 if the program produced correct output on any test case in that batch; otherwise we assign a score of 0.

¹ It is not the intention of this paper to criticize the IOI 2005 organizing committee in any way. Our evaluation is being performed on this IOI since it is the only available data set at the time of writing. The authors would be very willing to perform a similar analysis on data from other IOI competitions.

The All-or-Nothing scoring scheme is used in well-known programming contests such as ACM ICPC and TopCoder. (See [4] for an overview.) It has also been proposed for use at the IOI [3], but it was noted that many students would receive a total score of zero. Motivated by this observation, we developed the Significant Progress scoring scheme to reward students who make non-trivial progress on any batch.

In order to apply IRT, we need a measure of each student's ability. The only measures available to us are the results from IOI 2005. For a student at percentile rank p , we define her ability to be her standardized rank $\Phi^{-1}(p)$, where Φ is the cumulative normal distribution. This gives a normal (Gaussian) distribution of abilities with mean 0 and standard deviation 1. The item response curves are estimated using logistic regression. The individual item response curves are summed (with appropriate weights) to give the expected score as a function of ability.

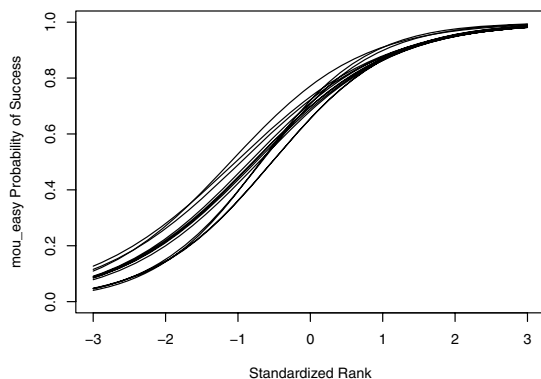


Fig. 2. Item Response Curves for Mountain Task (easy), IOI Scoring

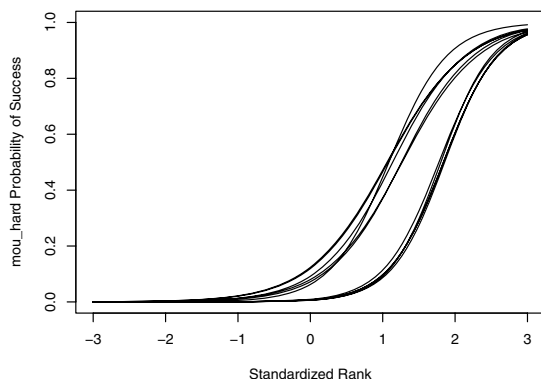


Fig. 3. Item Response Curves for Mountain Task (hard), IOI Scoring

3.1 IOI Scoring

According to the IOI scoring scheme, each test case is an item for which we fit an item response curve. The item response curves for all of the test cases of the “Mountain” task are shown in Fig. 2 and Fig. 3. Notice that the test cases of the easy batch (Fig. 2) all have similar shapes; their discrimination and difficulty parameters are very similar. These difficulty parameters are, of course, lower than the difficulty parameters of the hard batch (Fig. 3), as the curves for the hard test cases are further to the right.

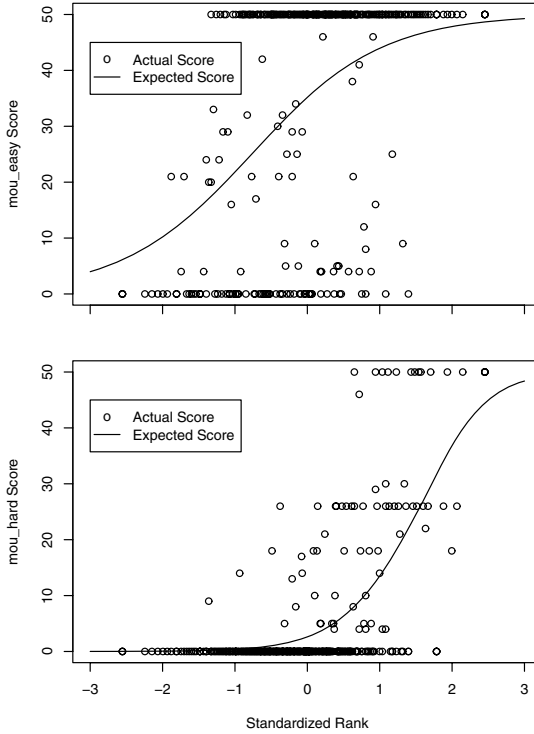


Fig. 4. Expected Score for Mountain Task (easy, hard), IOI Scoring

The expected score for a batch is a function of ability. We compute this function by taking the sum of the item response curves of its test cases, weighted by the value (4 or 5) of each test case. The resulting curves, together with the actual contestants’ IOI scores, for the easy batch and hard batch of the Mountain task are shown in Fig. 4.

The IOI 2005 competition was composed of 12 batches (6 problems with 2 batches per problem). We repeat the above steps for all of the other batches to get an expected score curve for each batch. These curves are shown in Fig. 5 (upper graph). Finally, in the lower graph, we sum these twelve expected score curves to get the expected overall score. The actual scores are also plotted.

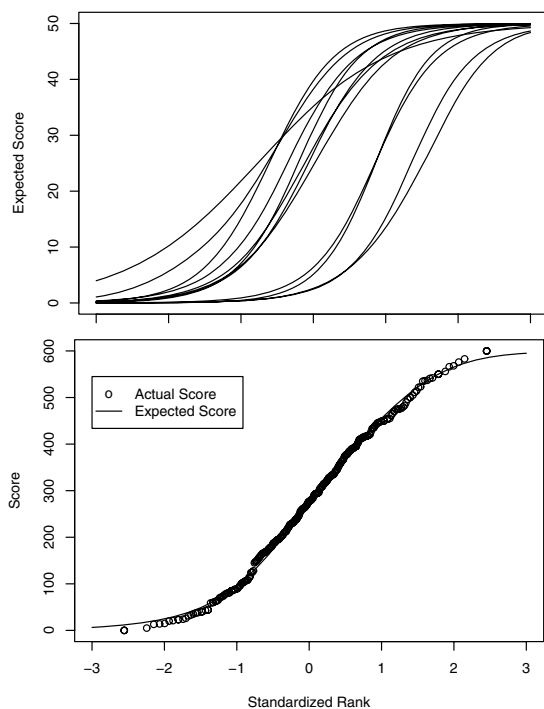


Fig. 5. Expected Batch Scores and Total Scores, IOI Scoring

3.2 All-or-Nothing Scoring

For the All-or-nothing scoring scheme, each batch of test cases is an item. The batch is marked correct if the student passes every case in that batch; otherwise the batch is marked incorrect.

To apply IRT to this scoring scheme, we fit an item response curve for each batch. The item response curves (scaled to give a maximum score of 50) for the easy Mountain batch and the hard Mountain batch are shown in Fig. 6. The actual scores awarded under this scheme are also plotted.

Repeating this procedure for each batch, we get the 12 curves shown in Fig. 5 (upper graph). When we compare these curves to the ones for the IOI scoring scheme (in the previous section) we see that the discrimination parameters are similar. The difficulty parameters of these curves are noticeably higher. The lower graph shows the expected total score, formed by summing these curves. The actual scores awarded under this scheme are also plotted, along with the IOI scores. The IOI scores are noticeably higher.

3.3 Significant Progress Scoring Scheme

The significant progress scoring scheme marks a batch as correct if the student passes any test case in that batch. For each batch, we fit an item response curve.

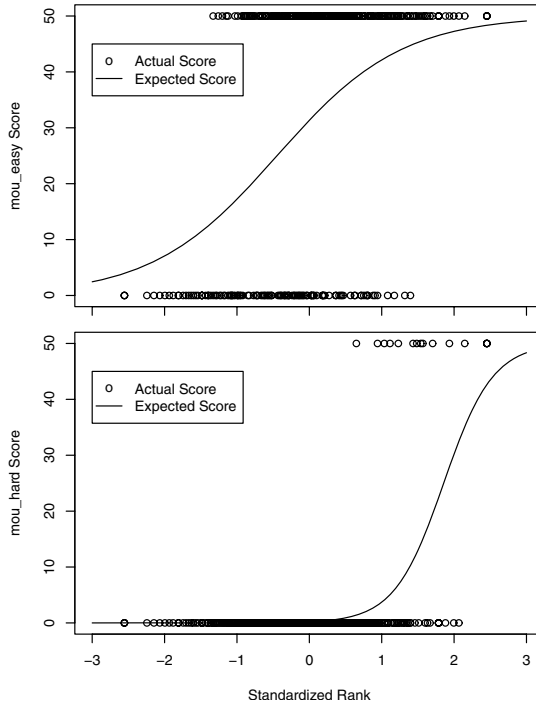


Fig. 6. Item Response Curves for Mountain Task (easy, hard), All-or-Nothing Scoring

The item response curves (scaled to give a maximum score of 50) for the easy mountain batch and the hard mountain batch are shown in Fig. 8 and Fig. 9, together with the actual scores awarded under this scheme. The curves for all twelve batches are shown in Fig. 10 (upper graph). When compared with the IOI scoring scheme, these curves have similar discrimination and lower difficulty. The curve of the expected total score, shown in the lower graph, shows scores that are much higher than the IOI scores.

3.4 Combining All-or-Nothing with Significant Progress

The All-or-Nothing and Significant Progress scoring schemes give good discrimination among high and low ability levels, respectively. It makes sense to use them both in a Combined scoring scheme. Each batch produces two items: one is marked as correct if any one test case is solved correctly, the other is marked as correct if all of the cases are solved. The resulting 24 item response curves are shown in Fig. 11 (upper graph); their sum produces the expected score curve shown below it. We see that this Combined scoring scheme yields good discrimination over a broad range of ability levels and produces an expected score curve that is nearly linear over the entire spectrum.

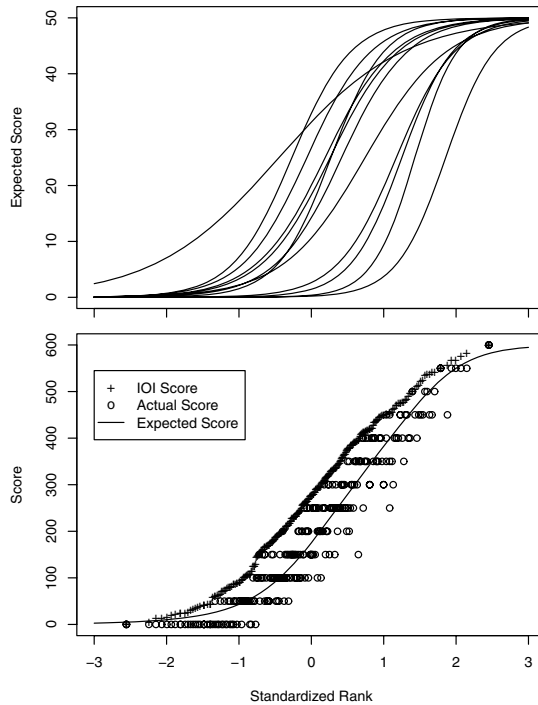


Fig. 7. Expected Batch Scores and Total Scores, All-or-nothing Scoring

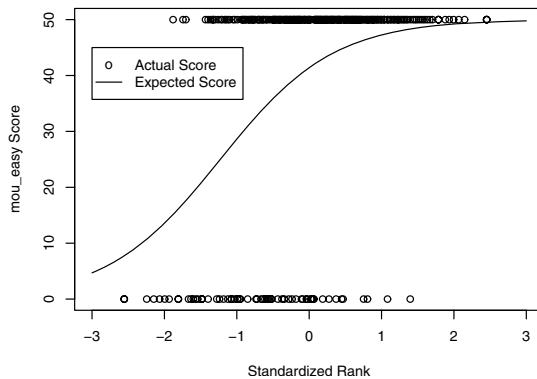


Fig. 8. Item Response Curves for Mountain Task (easy), Significant Progress Scoring

To quantify these observations, we computed the range of difficulty parameters for the item response curves of the IOI scoring scheme and the Combined scoring scheme. For the IOI scoring scheme, the parameters range from a low of -1.896 to a high of 1.855. For the Combined scoring scheme the range is nearly identical, from -1.896 to 1.856.

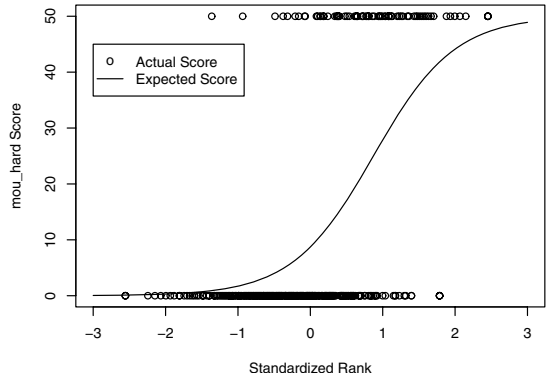


Fig. 9. Item Response Curves for Mountain Task (hard), Significant Progress Scoring

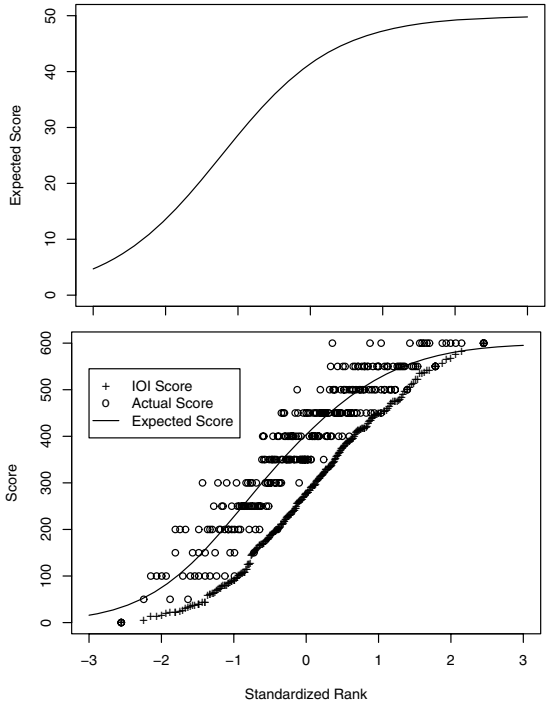


Fig. 10. Expected Batch Scores and Total Scores, Significant Progress Scoring

3.5 Partial Credit

The Combined scoring scheme uses only two pieces of information about each batch: “Did the student score 0?” and “Did the student earn a perfect score?” The scheme does not use the partial scores for each batch. Do the partial scores

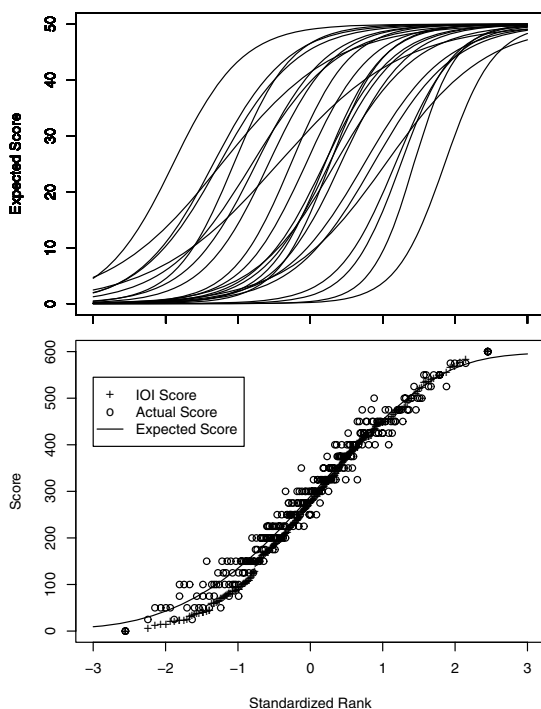


Fig. 11. Expected Batch Scores and Total Scores, Combined All-or-nothing and Significant Progress Scoring

provide any additional discrimination? To investigate this question we performed an additional experiment. For each batch, we isolated the population of students whose score was neither zero nor perfect. Then we created item response curves for each test case. By summing these curves (with appropriate weights) we get the expected score on this batch for this population. The expected score curves for the easy and hard batches of the Mountain task are shown in Fig. 12. They appear to have little discriminatory value. In fact, the curve for the easy batch suggests a negative correlation between ability and score! The expected score curves for all of the batches are shown in Fig. 13. In general, these appear to have poor discriminatory value.

4 Discussion

Presently there is no practical alternative to automated scoring at computing competitions such as the IOI. It is difficult for an automated system to determine the magnitude of the flaws in incorrect programs, which form the majority of the submissions at the IOI. Using item response theory, we have examined the discrimination and difficulty of the tasks at IOI 2005 under various automated scoring schemes. The current scheme awards scores in proportion to the number

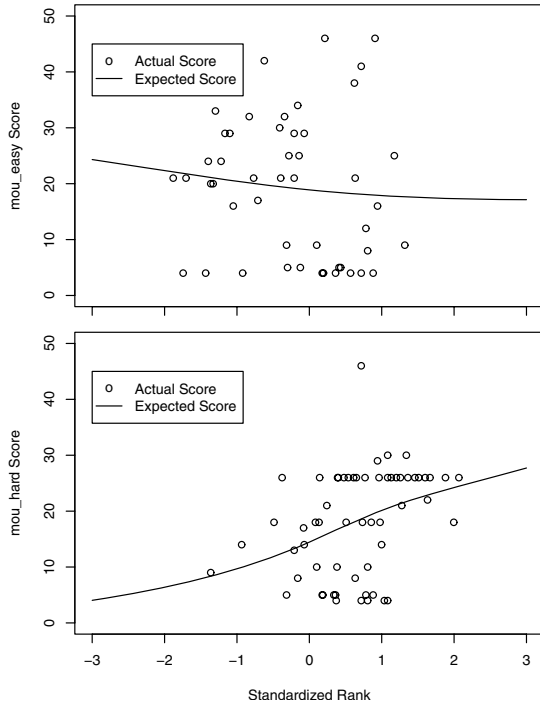


Fig. 12. Expected Batch Scores (easy and hard), Mountain Task, Part Marks Only

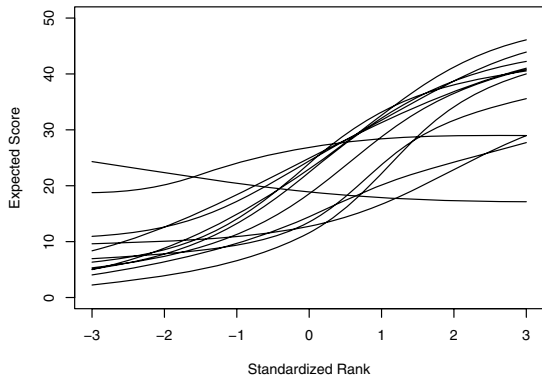


Fig. 13. Expected Batch Scores, Part Marks Only

of test cases for which the program produces correct output. This suffers from the drawback that there is no objective standard for measuring the number of cases solved by a program; it depends entirely on the authors of the test cases. Furthermore, we have shown that this scoring scheme yields poor discrimination among the submissions which receive partial credit.

We examined a Combined scoring scheme which awards credit for a batch of test cases according to whether the program solves all, some, or none of the cases. This scheme gives good discrimination over a broad range of ability levels. We argue that it assesses objective criteria which could (and should) be explicitly stated in the problem statements. The scoring would be improved by adding more batches of various difficulties, each evaluating specific, explicitly-stated criteria. Ideally, there should be an objective standard for the awarding of every mark in the contest.

This scheme makes the scoring more objective: two judges creating their own test data would likely award similar marks when evaluating a program. Contestants could predict the score that would be awarded for various submissions. Weaker students would have achievable goals. By making the scoring more predictable at all difficulty levels, contestants will have more confidence in the scoring and a more enjoyable contest experience.

References

1. The International Olympiad of Informatics (IOI), <http://www.ioinformatics.org/> (2006).
2. Baker, F. *The Basics of Item Response Theory* ERIC Clearinghouse on Assessment and Evaluation, College Park, MD (2001). (Also available at <http://edres.org/irt/baker/>)
3. Cormack, G. Random Factors in IOI 2005 Test Case Scoring. In *Perspectives on Computer Science Competitions for (High School) Students* (Dagstuhl, 2006).
4. Cormack, G., Kemkes, G., Munro I., and Vasiga T. Structure, Scoring and Purpose of Computing Competitions, *Informatics in Education* (5), 2006, 1-22.
5. Partchev, I. Visual guide to item response theory. Available at: <http://www2.uni-jena.de/svw/metheval/irt/VisualIRT.pdf>

Modelling and Evaluating ICT Courses for Pre-service Teachers: What Works and How It Works?

Lina Markauskaite, Neville Goodwin, David Reid, and Peter Reimann

The University of Sydney, Faculty of Education and Social Work (A35),
Sydney, NSW 2006, Australia

{L.Markauskaite, N.Goodwin, D.Reid, P.Reimann}@edfac.usyd.edu.au

Abstract. A design-based research for the restructuring of the Information and Communication Technology (ICT) literacy training for pre-service postgraduate teachers is reported. Conceptual rationales and the design of a newly implemented “Information Technology in Education” (ITE) course, which is based on the model of instructional design, are described. An accompanying longitudinal study is presented. Improvements in trainee teachers’ ICT-related general cognitive and technical capabilities; changes in beliefs about the use of ICT in their future career; and the ITE course learning experience are analysed. It was found that the implemented model of ICT literacy training fits best for more mature students who aim to acquire advanced technical skills and/or gain understanding of the use of ICT in the teaching profession. Research implications for future improvements of the pre-service teachers’ ICT training are discussed.

1 Introduction

During pre-service studies trainee teachers are expected to develop a well-rounded Information and Communication Technology (ICT) literacy,¹ which includes a range of ICT-related general cognitive and technical capabilities and professional awareness about the role of ICT in their future career [12, 19, 22]. There are many challenges in designing and implementing effective ICT literacy training in pre-service education [14]. Students, who start pre-service teachers’ training, do not have systematic and sufficient ICT technical skills [3]. It is difficult to develop their conceptual and practical understanding of integrating technology in education and to build professional attitudes about the role of ICT in a teacher’s work [5, 24].

Re-engineering of the curriculum of the Information Technology in Education (ITE) course, using principles of design-based research [7], has been implemented in a pre-service teachers’ training program at the University of Sydney. An accompanying longitudinal study that investigates the development of trainee teachers’ ICT literacy and aims to provide empirical evidence about the effectiveness

¹ “ICT literacy” is used as umbrella term in this paper. It covers all other similar terms, such as “ICT competence”, “ICT fluency”, “ICT skills”, “ICT proficiency”, that are used for the description of teachers’ ICT-related knowledge, skills, capabilities, values and other attributes in various sources.

of the redesigned curriculum as well as the ground for future improvements has been designed.

The main aim of this paper is to present the key results of this design-based research. The paper describes the main conceptual aspects of the new curriculum, which is based on the main elements of instructional design [1], and examines how the curriculum contributes to the enhancement of ICT literacy of trainee teachers. In particular the paper focuses on the following objectives:

1. To study changes in trainee teachers' ICT literacy during the ITE course.
2. To investigate trainee teachers' opinion about the usefulness of the ITE course.
3. To examine relationships of trainee teachers' opinions about the course with: (a) students' background characteristics; (b) initial ICT-related experience and (c) changes in ICT literacy during the semester.

Initially the main international and Australian ICT-related standards that provided an external framework for the objectives of the new ICT curriculum are reviewed. Then research literature on the 'best practices' in pre-service teachers' ICT training are summarised. Afterwards the key rationales that lie behind the design of the new curriculum are introduced and the model of the ITE course is presented. Next the main results from the empirical study are presented. Finally, suggestions for the improvement of teachers' pre-service ICT-related training are proposed.

2 Conceptual Foundation of the ITE Course Curriculum Design

A curriculum is a "decision-making action that integrates both intention and the manner in which the intention becomes operationalised in the classroom reality" [23, p. 24]. The process of the ITE course curriculum re-engineering was based on the principles of praxis that integrate theory-and-development and planning-and-practice [15]. As a starting point of the redesign process, three key elements of the curriculum were considered: 1) expected outputs; 2) course content and 3) instructional processes. These three elements can be described respectively in terms of: 1) current standards related to trainee teachers' ICT literacy; 2) ICT-related curriculum content for trainee teachers and 3) didactical and organisational elements of the pre-service training.

2.1 ICT Literacy Standards and Pre-service Teachers' Training

Trainee teachers are expected to possess a number of ICT-related capabilities before they start their professional career. Three types of standards set the key requirements for the ICT literacy of prospective teachers: 1) international and 2) national ICT-related standards for pre-service (and/or in-service) teachers and 3) ICT literacy standards for students.

Various internationally recognised standards for trainee teachers' ICT capabilities cover different areas and have different focuses. For example, the European standards have a holistic focus on a teacher's identity in the knowledge society and cover a set of interdependent areas such as technological proficiency to professional development and innovation [19]. The US standards have a similar, but more specific focus on the

capabilities of teachers to apply ICT for instructional purposes [12]. The international standards for the *Accreditation of Programs in Educational Communications and Instructional Technology (ECIT)* focus on the process of instructional design [1]. They cover five interrelated synergetic domains and 20 sub-domains of competence: 1) design; 2) development; 3) utilisation; 4) management and 5) evaluation.

In Australia, the responsibility for the school system is divided between the Commonwealth and State or Territory governments. At the state level in New South Wales, graduate teachers are expected to demonstrate the following proficiencies: 1) basic operational skills; 2) information technology skills; 3) software evaluation skills; 4) effective use of the Internet and 5) pedagogical skills for classroom management [22].

In New South Wales, Primary schooling includes Kindergarten plus Grades 1 to 6, while Secondary schooling involves Grades 7 to 12. Students' ICT literacy is assessed regularly in years 6 and 10 by the means of special Computer Skills Assessment (CSA) [20, 21]. The CSA requirements include: 1) the use of ICT for locating, accessing, evaluating, manipulating, creating, storing and retrieving information; 2) the ability to express ideas and communicate; 3) the awareness of the range of applications of ICT in society; 4) the ability to discriminate in the choice and use of the ICT and 5) the confidence to explore, adapt and shape technological understandings and skills [18]. All teachers across the subjects are required to contribute to the teaching of these capabilities. Thus, it is expected implicitly that all teachers will possess and teach those ICT literacy skills that are covered by the CSA.

In summary, all standards related to trainee teachers' ICT literacy explicitly or implicitly cover three areas: 1) generic problem-solving cognitive capabilities; 2) functional ICT capabilities and 3) professional competences to use ICT in a teacher's job.

2.2 Curriculum Content of the Best Practice of Teachers' ICT Education

The majority of pre-service teachers' training institutions has full autonomy in deciding about the content of the ICT-related curriculum and the methods of its delivery [10]. Some programs focus on functional ICT skills, whereas others concentrate on the use of ICT in a curriculum area and/or in teachers' practice in general. According to research, five content areas are covered in all good teachers' training programmes: 1) personal ICT competences; 2) ICT as a mindtool; 3) educational and pedagogical use of ICT; 4) ICT as a tool for teaching and 5) social aspects of ICT in education [13]. Two other areas – ICT in assessment and ICT policy – are also considered to be important; nevertheless they are not always present even in exemplary ICT training programs. Directed teaching of technical ICT capabilities and the use of ICT in education is not recommended [9, 13]. However only integrated ICT training may fail to provide a systematic structure of technological knowledge and later may limit trainee teachers' possibilities to apply ICT in education [25].

In summary, the content of a good ICT training program should cover both: 1) foundational systematic knowledge and functional ICT capabilities upon which trainee teachers' would be able to build their professional competence and 2) conceptual understanding and authentic experience of how ICT could be applied in the educational domain.

2.3 Didactical and Organisational Elements of ICT Training

There are a number of organisational models that could be used for the integration of ICT into trainee teachers' programs, such as: 1) a separate ICT subject for the acquisition of ICT skills and/or pedagogical knowledge; 2) ICT across courses; 3) practical use of ICT with children; 4) modelling or authentic planning, teaching and evaluation of the use of ICT for learning; 5) on-line interactions with teacher professional communities and others [10].

Most of the best programs on technology in teachers' education are an amalgam of several modes [8]. The best didactical approaches usually include a mix of the following four didactical techniques: 1) mediated lectures; 2) on-line support materials; 3) structured and supported lab sessions and 4) casual open access to resources. Many of the good programs make systematic use of peer learning, providing more experienced students with an opportunity to guide less experienced students. Three essential factors are common for all the best teachers' training practices: 1) faculty models sensible and proficient use of teaching technologies; 2) assignments are technology rich and 3) technology is seen as a natural support for teaching in both subject-oriented studies and pedagogical work in the faculty.

In summary, effective ICT courses should integrate various didactical techniques, and ICT-related training should be supported by a broader integration of ICT across various pre-service learning activities.

3 From Theory to Practical Design of the ICT Curriculum

The knowledge of the best ICT practices in pre-service teachers' training was used for the re-engineering of the ITE course curriculum at the University of Sydney. The following key decisions have been made consolidating all three conceptual aspects, discussed above, and designing the new course:

Standards and curriculum. The curriculum should be oriented on the ECIT model of instructional media design [1] which is to a large extent independent from short term technological developments. This model is based on the key steps of problem-solving and integrates the basic content areas of the national standards of ICT literacy for teachers and students.

Didactics. The delivery of the curriculum should be a flexible combination of the following three main modes: 1) lectures – for introduction of the main theoretical concepts, demonstration of the main tools and modelling of key skills; 2) guided tutorials – for students' work led by academic staff and 3) open lab workshops – for students' self-guided work and peer-assisted learning in technologically rich environment. All three modes should be supported to various degrees with online materials.

Other organisational elements. Students should be provided with settings that allow for hands-on experiences with a variety of software and hardware tools that they might encounter in their professional career. Experienced in ICT, students can be trained to act as peer assistant tutors, and can help less technology savvy fellow students to develop technical ICT capabilities.

Therefore, the new curriculum of the ITE course was based on the core elements of the ECIT standards [1]. Not all aspects of the ECIT standards were appropriate for beginning teachers, though. Thus, the model of instructional media design was simplified and adapted to specific needs of pre-service teachers (Table 1). The course mainly covered design, development and utilisation elements of instructional design. All ICT-related technical capabilities, which are required as a part of the national standards for teachers [22] and/or students [20, 21], were integrated into the course via “software focus”.

Table 1. The main elements of the “Information Technology in Education” course

Theoretical focus	Tutorial and self-study focus	Software focus
<u>Week 1–2:</u> Introduction to the ECIT model. Introduction to the Internet	Refining Internet search, downloading and evaluating sources	Internet browsers, subject directories, search engines
<u>Week 3–4:</u> Integrating pedagogical strategies, meeting learning needs, and promoting classroom interactivity	Productivity using a word processor, classroom resources, development and implementation	Low cost resources, paper aids, word processing
<u>Week 5–6:</u> From productivity skills to mindtools	Analysing and defining objectives, creating class databases, merging databases and documents	Database interrogation, construction, mail merge
<u>Week 7–8:</u> Choices effecting integration of learning variables, curriculum content and appropriate technology	Designing, developing and using computer based technologies for reference and learning	Visualisation and presentation programs, PowerPoint and SlideShow
<u>Week 9–10:</u> Matching learners, materials and contemporary instructional processes in interactive lessons	Identifying and applying techniques for integrating ICT innovations in various learning contexts	Web design: rationale and resources
<u>Week 11–12:</u> Sampling teaching resources, reviewing the ECIT model	Incorporating instructional processes in the development of interactive lessons	Web design: interactivity in sites and CD-ROMs

In the first year, as the redesign of the course led to a reduction in staff teaching time by about 50%, the hours saved were used for the development of resources for tutorials and open lab workshops. In the next years the efficiency of the ICT training will be monitored, and the lecture content, software focus and other organisational aspects will be modified on the basis of evaluation.

4 Method

4.1 Instruments, Procedure and Participants

An ICT literacy self-assessment questionnaire, based on theories of self-efficacy [4] and planned behaviour [2], has been designed to investigate changes in trainee teachers’ ICT literacy. The instrument was based on a dynamic model of ICT literacy

and covered present ICT-related capabilities, their sustainability and transferability into the future professional domain [16, 17]. Two scales in the questionnaire measured students' self-efficacy beliefs about their ICT-related general cognitive and technical capabilities. These scales were aligned with the national standards relevant to trainee teachers [6, 20, 21, 22]. The items were phrased in terms of "can", i.e.: "I believe I have a capability to <perform a task>". A six-point scale was used for answers: 0 – "Couldn't do that" and from 1 – "Not at all confident" to 5 – "Totally confident". One scale measured transferability of ICT-related capabilities and asked trainee teachers' about their intentions to use ICT in their future career. The items were phrased in terms of "will": "I believe that I and/or my students will <do a specific activity>". A five-point scale was used for answers: from 0 – "Very unlikely" to 4 – "Very likely".

Trainee teachers were asked to complete questionnaires twice – at the beginning and at the end of the ITE course. An additional scale for the assessment of the ITE course learning experiences was included into the second survey. The items inquired about the usefulness of the different elements (e.g., lectures, workshops) of the ITE course and were phrased in terms of "was useful": "The <element of the course> was useful for me". Students indicated their agreement using a six-point scale: 0 – "I did not attend/use" and from 1 – "Strongly disagree" to 5 – "Strongly agree". They also described the greatest benefits and the greatest barriers, experienced during the ITE course, in two open-ended questions.

Participants were the first year pre-service teachers at the University of Sydney. They followed a two year postgraduate course culminating in the award of a Master of Teaching degree with specialisation in Primary or Secondary (various subjects) teaching. All students were invited to participate. The participation was voluntary and confidential. The students completed the questionnaires outside the classroom hours. 217 students were enrolled in the program. 140 students (64.5%) participated in one or both surveys. 122 students (56.2%) completed the first survey, 89 students (41%) completed the second survey and 70 students (32.3%) completed both of them. For longitudinal comparison purposes, the analysis in this paper is based on the responses of the latter group.

4.2 Results

To answer the research questions the analysis was conducted in three steps. Initially trainee teachers' background characteristics and previous experience with ICT were investigated. Then, their ICT-related general cognitive capabilities, technical capabilities and beliefs about the use of ICT in their future career before and after the course were compared. The mean scores before and after the course for individual items, subscales² and overall scales were calculated and compared. Finally, students' ITE course learning experiences were studied. Relationships of the students' opinion about the usefulness of the ITE course with their background characteristics, prior experience with ICT and growth in ICT literacy during the course were investigated. Paired samples t-test and marginal homogeneity test were used for exploring significant changes. As these two tests gave essentially

² The subscales were constructed using factor analysis and the data from the first survey [17].

similar results, only results of the t-test are reported in this paper. Pearson's correlation was used for the investigation of relationships between the means of various indicators.

Background and Experience with ICT. There were 78.6% female and 21.4% male students in the sample. The age of respondents ranged from 21 to 53 years with an average 29.9 (standard deviation – 8.91) and median 26 years. On average, the students used computers for 15.6 (3.99) years. Just more than half (52.9%) of them were taught to use ICT in courses and about two thirds (67.1%) used ICT for learning various non-ICT courses in secondary school (grades 7–12) or/and in the last four

Table 2. Summary of average and subscale scores: ICT-related general cognitive and technical capabilities and beliefs about the use of ICT in future career before and after the course

Scale / subscale	#	Phase 1		Phase 2		Difference		
		M	SD	M	SD	M	SE	T(sig)
1. General cognitive capabilities – Average	70	3.54	0.627	3.78	0.475	0.23	0.071	3.267**
1.1 Problem solution capabilities	70	3.44	0.765	3.76	0.493	0.32	0.085	3.725***
1.2 Communication & metacognition capabilities	70	3.71	0.556	3.81	0.563	0.10	0.073	1.424 ^{NS}
2. Technical ICT capabilities – Average	67	3.10	1.010	3.65	0.812	0.55	0.076	7.235***
2.1 Basic ICT capabilities	67	4.20	0.728	4.31	0.637	0.10	0.076	1.346 ^{NS}
2.2 Analysis and production capabilities	69	2.65	1.326	3.38	0.992	0.72	0.102	7.137***
2.3 Information and internet-related capabilities	69	2.73	1.120	3.42	0.915	0.69	0.089	7.837***
3. Beliefs about ICT use in future career – Average	64	2.55	0.644	2.62	0.606	0.07	0.055	1.276 ^{NS}
3.1 Enrichment of teaching and learning	67	2.77	0.795	2.89	0.680	0.12	0.081	1.523 ^{NS}
3.2 Access, communication & self-based learning	68	2.34	0.826	2.29	0.669	0.05	0.077	-0.684 ^{NS}
3.3 Constructivist learning	67	2.27	0.812	2.30	0.771	0.03	0.077	0.338 ^{NS}
3.4 Teaching of general cognitive capabilities	67	2.96	0.775	3.06	0.680	0.10	0.084	1.159 ^{NS}
3.5 Teaching of ICT capabilities	67	2.41	0.869	2.45	0.810	0.03	0.069	0.478 ^{NS}
3.6 Professional activities and development	66	2.94	0.680	3.04	0.719	0.10	0.077	1.273 ^{NS}

Notes: # – Number of respondents; M – Mean; SD – Standard Deviation; SE – Standard Error; *** – $p < 0.001$; ** – $p < 0.01$; * – $p < 0.05$; NS – Not Significant.

years. Many respondents (88.4%) indicated that they learned most about computers at work and/or were self-taught. Just more than half of students (51.4%) used computers on average 10 hours or less per week and others (48.6%) used computers for more than 10 hours.

General Cognitive Capabilities. On average, the students were between “Moderately confident” (3) and just above “Quite confident” (4) with their various general cognitive capabilities (Table 2). There was a significant ($p < 0.01$) positive change in students' confidence from 3.54 (0.627), at the beginning, to 3.78 (0.475), at the end of the semester. The biggest statistically significant ($p < 0.01$) growth was observed in

students' confidence with their problem solving capabilities, namely: to outline a plan; to find information and select appropriate tools and to manage information.

ICT-related Technical Capabilities. On average, the students were between just above "Not at all confident" (1) and almost "Totally confident" (4) with their various ICT-related technical capabilities. Already at the beginning, the students were more than "Quite confident" about their basic ICT capabilities, such as managing files, performing basic tasks common to software applications, performing basic word-processing and communing via email. However, the majority of trainee teachers did not have or were not confident about their more advanced technical capabilities, such as how to design a database and create a website. During the semester, there was a significant ($p < 0.001$) positive change in students' confidence about their technical capabilities. The average score rose from just 3.10 (1.01) up to 3.65 (0.812). The main changes were in students' confidence to use advanced technical tools related to 1) the analysis and production and 2) information and Internet. The biggest growth ($p < 0.001$, exceeding on average one scale point, was in students' confidence with the following three capabilities: to design and interrogate their own databases; to create a basic webpage and to maintain a multipage website. The growth in students' confidence with their basic ICT capabilities was statistically insignificant ($p > 0.05$).

Beliefs about ICT in the Future Job. At the beginning of pre-service training, the students did not have strong beliefs about how they would use ICT in their future profession. The mode answer was "Possible" (2) to 21 items and "Likely" (3) to 7 items out of 34. On average, the students indicated that it is almost "Likely" that they would use ICT for various professional activities and would teach their students ICT-related general cognitive capabilities. However it is just above "Possible" that they would use ICT for students' self-based and constructivist learning. After the semester, the mode answer was "Possible" to 15 items and "Likely" to 15 items out of 34. However, there were very few (3 out of 34) significant changes in trainee teachers' beliefs. At the end of the semester, the prospective teachers indicated that it is more likely ($p < 0.05$) that their students would be involved in integrated ICT projects and that they would use ICT for communication with parents or guardians. In contrast, trainee teachers indicated that it is significantly ($p < 0.05$) less likely that they would search and review software for potential use by students.

The Course Learning Experience. Overall, the experience of trainee teachers about the ITE course was positive (Table 3). Students were most positive about the usefulness of various workshops run by the lecturers, more than 90% of respondents agreed or strongly agreed that, on the whole, these workshops had been useful to them. The trainee teachers were somewhat less positive about the usefulness of the key lectures. Nevertheless, more than 70% of them agreed or strongly agreed that on the whole and separately each of the three key lectures had been useful. More than 70% of students also agreed or strongly agreed that, on the whole, on-line learning materials had been useful for them. However, more than 15% of trainee teachers indicated that they had not used on-line learning materials, except for the introductory lecture. More than 17% of students also indicated that they had not attended optional sessions for individual work in computer laboratories, whereas more than 60% of trainee teachers agreed or strongly agreed that these optional sessions had been useful.

In two open-ended questions the majority of trainee teachers described the greatest benefits (74%) and/or barriers (52%), which they experienced during the ITE course. The students indicated one or several of the following beneficial aspects most often: learning about databases and mail merge (33%); learning about web design (28%) and enhancement of various other functional ICT capabilities (33%). About 12% of trainee teachers commented that they had benefited from the increased understanding of how ICT can be used in teaching. The greatest barrier, mentioned most often, was related to over-complicated curriculum and workshops which were too fast paced (19%). In contrast, some other students commented that the course curriculum had

Table 3. ITE course learning experience: usefulness of different elements of the course

<i>"The <element of the course> was useful for me"</i>	%						Mean rating		
	NP	SD	D	N	A	SA	#	M	SD
1. The Introductory lecture	5.8	1.4	5.7	14.3	57.1	15.7	66	3.9	0.83
2. The on-line material for the Introductory lecture	10.0	0.0	5.7	30.0	42.9	11.4	63	3.7	0.78
3. The lecture on Graphics	5.6	1.4	2.9	18.6	38.6	32.9	66	4.1	0.90
4. The on-line material for Graphics	17.2	1.4	4.3	27.1	37.1	12.9	58	3.7	0.87
5. The lecture on Databases	8.6	1.4	7.1	10.0	52.9	20.0	64	3.9	0.89
6. The workshop on Databases run by the lecturer	4.3	0.0	1.4	4.3	41.4	48.6	67	4.4	0.66
7. The on-line material for Databases	20.0	0.0	1.4	17.1	42.9	18.6	56	4.0	0.73
8. The workshop on Power Point run by the lecturer	7.2	0.0	1.4	14.3	51.4	25.7	65	4.1	0.70
9. The on-line material for Power Point	22.8	0.0	4.3	20.0	38.6	14.3	54	3.8	0.80
10. The workshop on Webpage run by the lecturer	4.3	0.0	2.9	8.6	37.1	47.1	67	4.3	0.77
11. The on-line material for Webpage construction	18.5	1.4	2.9	22.9	32.9	21.4	57	3.9	0.92
12. On the whole, the key lectures on different topics	4.2	0.0	2.9	18.6	54.3	20.0	67	4.0	0.73
13. On the whole, the workshops led by the lecturer	4.4	0.0	1.4	1.4	45.7	47.1	67	4.5	0.61
14. On the whole, the optional sessions for individual work	21.4	1.4	2.9	15.7	38.6	20.0	55	3.9	0.88
15. On the whole, the course on-line material	11.4	0.0	2.9	17.1	45.7	22.9	62	4.0	0.77
Overall ITE course learning experience – Average							67	4.0	0.48

Notes: NP – Did not participate/use; SD – Strongly Disagree; D – Disagree; N – Neutral; A – Agree; SA – Strongly Agree; # – Number of respondents (excluding NP); M – Mean (NP are included into the overall ITE course learning experience only); SD – Standard Deviation.

been too basic and/or the workshops had been very slow (7%). A number of students disliked Apple computers that had been used for the workshops (13%). However, others indicated that the exposure to Apple computers had been beneficial (7%) as these technologies are commonly used in Australian schools.

ITE Course Experience and Students' Characteristics. Using assigned ordinal values, average scores for each item of the ITE course evaluation and all items together were calculated and their relationships with other students' characteristics were examined. There were no significant differences between females and males

opinions about the usefulness of the course. Previous experience of using ICT in non-ICT subjects was also not related to students' opinions. However, the students who had not learned about ICT in courses previously rated the usefulness of the following elements of the ITE course significantly higher: the lecture about databases (4.16 (0.779) and 3.67 (0.924), $t=2.308$, $p<0.05$); the workshop about databases (4.68 (0.750) and 4.22 (0.722), $t=2.996$, $p<0.01$) and the online learning material (4.35 (0.745) and 3.83 (0.730), $t=2.589$, $p<0.05$). The students who used computers more than 10 hours per week evaluated the usefulness of the online learning materials for almost all topics (with exception for webpage construction) significantly ($p<0.01$) higher than those students, who worked with computers for 10 or less hours per week (average for five topics – 3.99 (0.924) and 3.50 (0.628), $t=3.278$, $p<0.01$). The average ITE course learning experience did not correlate significantly neither with students' confidence and beliefs before nor after the course, or with the average changes in their confidence and beliefs. However, students' initial confidence with their general cognitive and technical capabilities significantly ($p<0.001$) negatively correlated with changes in the corresponding scores after the course (-0.689 and -0.577, respectively) indicating that students, who initially were less confident about their capabilities, experienced bigger growth in their confidence during the semester.

Profiles and open-ended comments of five students, who on average rated their ITE course learning experience the highest (more than 4.7) and the lowest (less than 3.5), were examined in detail. The proportion of the female and male students in both groups was similar to the proportion of females and males in the sample. The majority of students (80% or more) who were satisfied were quite mature (35 years or more). Furthermore they had different ICT-related technical confidence (usually above group average) and used computers more intensively (more than 10 hours per week). In the open-ended comments, many of them indicated that they benefited most from learning new advanced technological skills and pedagogical knowledge: "I really learned how technology can be useful in the classroom," one student wrote. In contrast, the majority (80% or more) of the least positive students were younger (30 years or less), had lower than the sample average ICT-related technical confidence and used computers less intensively (10 hours or less per week). All positive and negative comments of these students were related to technical skills only.

5 Discussion and Conclusions

The redesigned curriculum of the ITE course was based on the core elements of the ECIT standards and the model of instructional design [1]. It was believed that future teachers will need to be able to *analyse* students' needs for ICT support, *plan* for the use of ICT in their classrooms, *evaluate* available alternatives, *configure* ICT for specific learning activities, *manage* the ICT-enriched classroom, and *evaluate* critically their practices. The course model adopted permitted the integration and enhancement of ICT-related cognitive and technical capabilities into a rich framework of instructional design. The model also provided for a meaningful combination of various didactic methods (lectures, workshops, online materials, etc.) and gave opportunity to utilise staff time more efficiently.

The initial empirical study showed marked diversity in students' backgrounds, their ICT-related confidence and beliefs. The majority of postgraduate trainee teachers used ICT for a long time. However, the substantial minority had never been taught systematically to use ICT and never applied ICT for the learning of other non-ICT subjects. The students were between moderately and quite confident about the majority of their cognitive and basic ICT capabilities, but they were much less confident about their capabilities to perform more advanced tasks, such as analysis and production with ICT, and information and Internet-related tasks. Students did not have a strong opinion about how they would use ICT in their future job.

The ITE course (and possibly other courses) taught during the first semester, contributed to the improvement of students' confidence with their ICT-related general cognitive and advanced technical capabilities. The ITE course quite well matched an average level and profile of students' ICT literacy. The overall instructional media design process involved enhancement of students' cognitive capabilities (particularly planning and other problem-solving skills). The selected software focus and hands-on workshops covered the development of presentations, databases and web-pages.

However, there were very few significant changes in trainee teachers' beliefs about the use of ICT in their future job. Although didactical and other professional aspects of ICT use in schools were not addressed directly in the ITE curriculum, it was believed that the students exposure to instructional design will indirectly enhance their understanding about the application of ICT in a teachers' job. In addition, it appears that two significant changes in students' beliefs were likely to be caused by another course, as the issues of school-home-communication and integrated learning were covered in the professional unit "Introduction to Teaching and Learning", which was taught via group discussions. This incidental finding suggests that more authentic and discursive learning about the use of ICT in educational practice could be an effective way for enhancing students' professional awareness.

Overall, the students were quite positive about the usefulness of the ITE course, particularly workshops. However, about one fifth of trainee teachers have never attended individual sessions and/or used online learning materials, indicating that they had little interest in learning about educational technologies beyond that required by the course curriculum. The analysis of open-ended comments and profiles of those students, who valued their ITE course experience either most or least suggested that the course was more appropriate for more mature students who used ICT regularly, but did not possess advanced ICT capabilities. These students were not overloaded with technical complexity of the course and were able to independently link instructional design activities with the use of ICT in a classroom. Younger and less experienced ICT students concentrated on technical aspects of the course and were unable to make individual interpretations of how ICT can support their profession. As suggested in the literature [5, 11], these students probably need a more explicit curriculum that exposes them to the conceptual and practical understandings of using ICT in education.

The findings indicated several main directions for future changes in the ITE course. Lectures and workshops led by lecturers should focus not only on instructional design, but also link the design of ICT-based media with its use in the classroom. New collaborative and more discursive learning modes that promote students' critical thinking about the role of ICT in a teacher's job need be introduced.

The course should provide more opportunities for students to adapt tasks and learning methods to their individual level of ICT-related capabilities and learning needs. The quality of various individual learning modes should be improved, so that students would be able to develop necessary ICT-related technical capabilities almost independently. These findings and emerging recommendations should be explored in follow-up studies.

References

1. AECT: Standards for the Accreditation of Programs in Educational Communications and Instructional Technology (ECIT). 4 ed. AECT, Bloomington, IN (2001)
2. Ajzen, I.: The Theory of Planned Behavior. *Org. Behav. & Hum. Decis. Proc.* 50 (1991) 179-211
3. Aust, R., et al.: Learning Generation: Fostering Innovation with Tomorrow's Teachers and Technology. *Int. J. of Tech. & Teach. Educ.* 13(2) (2005) 167-195
4. Bandura, A.: Self-efficacy. In Ramachaudran, V.S. (ed.): *Encyclopedia of Human Behavior*. Academic Press, New York, NY (1994) 71-81
5. Basham, J., Palla, A., Pianfetti, E.: An Integrated Framework Used to Increase Preservice Teacher NETS-T Ability. *Int. J. of Tech. & Teach. Educ.* 13(2) (2005) 257-276
6. Bundy, A. (ed.): *Australian and New Zealand Information Literacy Framework. Principles, Standards and Practice*. 2 ed. ANZIIL, Adelaide (2004)
7. Cobb, P., et al.: Design Experiments in Educational Research. *Educ. Res.* 32(1) (2003) 9-13
8. Davis, N.: Technology in Teacher Education in the USA: What Makes for Sustainable Good Practice? *Tech., Pedag. & Educ.* 12(1) (2003) 59-84
9. DiPietro, K.: The Effects of a Constructivist Intervention on Pre-Service Teachers. *Educ., Tech. & Soc.* 7(1) (2004) 63-77
10. Downes, T., et al.: *Making Better Connections: Models of Teacher Professional Development for the Integration of ICT into Classroom Practice*. DEST, Canberra (2001)
11. Hughes, J.: The Role of Teacher Knowledge and Learning Experiences in Forming Technology Integrated Pedagogy. *Int. J. of Tech. & Teach. Educ.* 13(2) (2005) 277-302
12. ISTE: *National Educational Technology Standards for Teachers*. ISTE (2001)
13. Kirschner, P., Davis, N.: Pedagogic Benchmarks for Information and Communication Technology in Teacher Education. *Tech., Pedag. & Educ.* 12(1) (2003) 125-147
14. Kirschner, P., Selinger, M.: The State of Affairs of Teacher Education with Respect to Information and Communication Technology. *Tech., Pedag. & Educ.* 12(1) (2003) 5-17
15. Lovat, T.J.: Curriculum Theory: The Oft-missing Link. *J. of Educ. for Teach.* 14 (1988) 205-213
16. Markauskaite, L.: From a Static to Dynamic Concept: A Model of ICT Literacy and an Instrument for Self-Assessment. In Goodyear, P., et al. (eds.): *ICALT 2005. Proceedings*. IEEE Computer Society, Los Alamitos, CA (2005) 464-466
17. Markauskaite, L., et al.: Exploring the Fit of an Information Technology Course to the ICT Literacy of Trainee Teachers. In SITE 2006, 17th International Conference, Orlando, Florida, March 20-24 (2006)
18. MCEETYA: *Annual Report. ICT*. MCEETYA, Canberra (2000)
19. Midoro, V. (ed.): *A Common European Framework for Teachers Professional Profile in ICT for Education*. Menabo, Ortona (2005)

20. NSW BS: Trial Year 10 Computing Skills Assessment. NSW Board of Studies, Sydney (2003)
21. NSW DET: Computing Skills Assessment. Year 6. NSW DET, Sydney (2004)
22. NSW IT: Professional Teaching Standards. NSW Institute of Teachers, Sydney (2005)
23. Smith, D.L., Lovat, T.J.: Curriculum: Action on Reflection. 4th ed. Social Science Press, Tuggerah (2003)
24. Wang, Y.-M.: When Technology Meets Beliefs: Preservice Teachers' Perception of the Teacher's Role in the Classroom with Computers. *J. of Res. on Tech. in Educ.* 35(1) (2002) 150-161
25. Watson, G.: Pre-Service Teachers' Views on their Information Technology Education. *J. of Inf. Tech. for Teach. Educ.* 6(3) (1997)

Sustaining Local Identity, Control and Ownership While Integrating Technology into School Learning

Deirdre Butler¹, Carol Strohecker², and Fred Martin³

¹ St. Patrick's College, Dublin City University
Drumcondra, Dublin, Ireland
deirdre.butler@spd.dcu.ie

² Strohecker Associates, Dublin, Ireland
carol@stroheckerassociates.com

³ University of Massachusetts Lowell, Department of Computer Science
Lowell, MA 01854 USA
fredm@cs.uml.edu

Abstract. Empowering Minds (EM) is a project that has deeply integrated technologically expressive materials, including robotics and presentation and programming software, into the educational practices of teachers at more than a dozen primary schools across Ireland. Here, we present Empowering Minds as a model of constructionist learning and teacher professional development. As the project continues to grow, organisers are challenged to aim beyond sustainability to scalability, encouraging participants to develop ways of maintaining their local identities and their senses of control and ownership of the work and ideas. In this paper, we describe a collaborative project among four participating schools, including one teacher's account of his students' adapting a folkloric story in order to appropriate it for their own technical, narrative, and learning purposes. We emphasise the importance of a diverse, supportive community and adequate time for teachers' self-directed learning to develop and epistemological changes to occur.

1 Introduction

Through the Empowering Minds project (EM) we are putting principles of constructionism into practice within primary schools in Ireland. Students and teachers are learning side-by-side about robotics and computer programming, in creative contexts that invoke and depend as much on arts, history, and culture as on science and engineering. The EM project demonstrates the importance of self-motivation and of a supportive learning culture. EM also illustrates how confidence and self-esteem develop and help to further promote learning, for both students and teachers.

We focus on the role of teachers as self-determined learners and on transformation of classrooms as studio-style, constructive learning environments. As EM grows to include more schools – and more types of schools, as well as community centres – we are particularly concerned with reflecting and sustaining local identity, control, and ownership as people adapt the learning model to their own circumstances.

EM settings are more akin to artists' studios than traditional classrooms. Each learner decides what projects to work on and therefore the contents of the learning and how to approach it. People of mixed generations and multiple learning styles have access to the community, the materials, and the concepts that the materials afford. Decision-making within the community is democratic. These methods emphasise dignity, equity, and inclusion, promoting the values of identity, control, and ownership.

In this paper, we develop three themes that characterize work in the EM project:

- *Identity*, which grows and clarifies as individual learners consider their aims and preferences, and their roles within the group. Community members strengthen and sustain these aspects of identity by affirming them through the social interactions.
- *Control*, which addresses the power relationships as learners decide what projects to undertake. Participants negotiate the sharing of power between teachers and children (each learners), among different schools, and between the researchers and all participating.
- *Ownership*, which develops as children and teachers engage in their own projects, individually and collaboratively, and as each participating school contributes to the project overall, demonstrating and maintaining its collective identity. Just as individuals and collectives have rights to decide what projects to work on and how to approach them, they have responsibilities to the project community to meet agreed-upon deadlines, share resources, and maintain the EM values.

The paper is organized as follows. In section 2, we introduce the Empowering Minds project. In section 3, we look closely at the work of a particular classroom during the 2002–2003 academic year, focusing on the how key EM themes are represented in this classroom's work. In section 4, we present concluding remarks.

2 Introduction to Empowering Minds

The Empowering Minds project was founded with three core principles [1, 16]:

1. To encourage children and teachers to develop technological fluency with project-based learning.
2. To use technology as an integrating agent for learning.
3. To establish a new model of teacher professional development, in which teachers are centrally included in the process of pedagogical activity design.

The project began in 1999 with a core group of 4 schools (urban-disadvantaged, suburban-advantaged, typical suburban, and rural) and 8 teachers; its first full year of operation was the 1999–2000 academic year. In the following year, the project approximately doubled in size.

In both years, teachers were provided significant hands-on project time for their own learning. When they brought the materials into their classrooms, it was on their own terms and with the support of the whole community. The discussion that follows further develops the values of the EM project.

2.1 Constructionist Learning

Constructionism counters the image of learners as vessels into which we must instill knowledge; rather we see learners as active builders of their own knowledge. Constructionists argue that people can build their knowledge best when they engage in building something in the world that pertains to the particular ideas. When the artefact also reflects personal decisions and aesthetics, it supports the engagement – and therefore the learning – especially well. The creator and others can regard and comment on such artefacts as they progress, and the learner's thinking benefits from the multiple views and discussions. The tools and materials influence the nature of the artefact and therefore the thinking.

EM emphasizes digital technologies both because they are prominent in contemporary culture, and therefore important from a skills-development perspective, but even more importantly because properties of computational materials characterize general phenomena that are worth knowing about: ideas like variability and feedback are fundamental to dynamic systems from thermostats to traffic patterns to weather systems to economic cycles to families and organisations. Learning about computation develops a basis for thinking about the world and how it works.

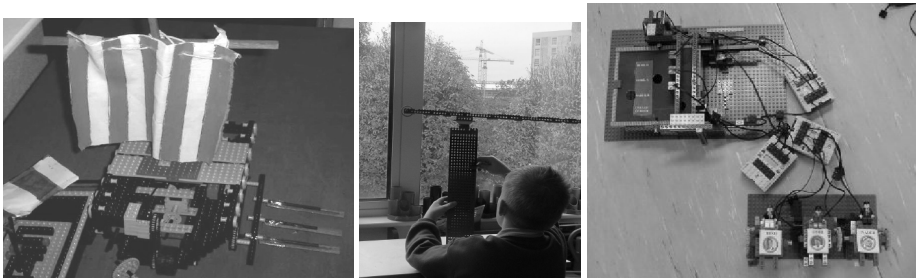


Fig. 1. Student Work: A Viking longboat; testing a crane; a voting machine inspired by Florida's ballot debacle in the 2000 USA Presidential election

2.2 Expressive Materials

Our emphasis on the choice of computational materials (i.e., LEGO Mindstorms and Logo Microworlds programming) does not mean we take a technocentric view of change in schools and learning. The malleable, protean nature of these materials, which makes them useful in learning about the workings of dynamic systems, also makes them capable of rendering ideas through many manifestations. Constructions using computational materials can demonstrate a wide range of behaviours and styles of presentation reflecting the creators' whims and urges. Thus these materials are well suited to enabling individual engagement, expression, and learning (**Fig. 1**).

Many people see "the computer" as merely a set of office and reference tools, so that "computer literacy" means learning to use a word processor, spreadsheets, e-mail, web browsers and search engines. We go further by asserting that children and teachers need to learn how to write computer programs and to create robotic

constructions with working gears, motors, and sensors. By engaging in such computational materials they can develop understandings of powerful ideas and express their learning – and themselves – through colourful, dynamic forms.

Thus we see computational materials as provoking challenges to existing values and beliefs about learning, not as tools to fit existing curricula. The “information and communication technologies” (ICT) approach problematically bolsters the traditional transmissive mode of education. We see the increased range and wide availability of computational materials as enabling a new, constructive mode and as transforming the teacher’s role from that of consumer of existing courses and curricula to that of empowered, self-determined learner.

Computational materials are challenging, conversational, and connective: they address personal needs and interests, they invite contemplation and negotiation, and they support development of personal relationships and ideas that transcend traditional subject boundaries.

2.3 Atelier-Style Environments

The constructionist approach is based on an intimate connection between knowledge and activity. Therefore the selection of specific building materials is an important start in enabling Constructionist learning. Traditional classroom settings, however, are not conducive to the activity of building and to encouraging collaboration and the free exchange of ideas. Environments more familiar to artists, architects, and craftspeople are better suited. In these environments, participants engage with complex, open-ended problems over a protracted length of time, they are encouraged to collaborate, they address a heterogeneity of issues, and reflection is explicitly incorporated [3, 13]. These characteristics frame the *studio* or *Atelier* way of working.

The Atelier model is rooted in European traditions of artisanship. Many of the features of the design studio popular in today’s arts and architecture practices derive from the Atelier-based training at the École des Beaux-Arts in 19th-century Paris [4]. The model lends itself to technology research as well: “It is a way of working that emphasises experimental production: building, crafting and demonstrating become ways of situating...inquiries” [22].

Consistent with these practices, the EM workshops had teachers working side-by-side with more experienced practitioners – just as, later, the schoolchildren worked side-by-side with the EM teachers. The more experienced participants explained techniques and offered comments, interacting with the teachers “extensively, answering their questions, helping them solve problems and otherwise listening to their thoughts and concerns. This style of interaction was more as between peers than as between teacher and student; often the problem puzzling the [learners] was one we had not yet solved ourselves” [15].

As the teachers transferred this way of working to the classrooms, the more experienced personnel supported them heavily at first, through frequent classroom visits. Gradually the teachers and children became comfortable with the computational materials and this new way of working began to grow as part of their culture.

2.4 Teacher Development

As teachers generally have been passive recipients of “education” in their own lives, it is hardly surprising that they tend to perpetuate the “transmissive” model of learning. We needed to organise a strong “consciousness raising” process [8] in order to awaken EM teachers’ understandings of what learning could be and to enable them to broaden their perspectives on their practice.

The importance of broadening and changing teachers’ meaning perspectives cannot be overstated: this is the framework that shapes their perceptions of themselves and of others and their surroundings. Teachers, like anyone, form such a perspective through experience; therefore it is critical, if change is to occur, that teachers have opportunities to continually broaden their experiences. And in tandem with expanded breadth of experience comes a need for continual questioning of assumptions and existing practices.

We shaped the project around an important question of our own: what type of professional development would foster autonomy in teachers, empowering them to take control of their own learning? Drawing on the literature from adult learning [14,12] we ensured that the teachers’ personal needs and interests, along with their experiences, would form the starting point around which to challenge and build new perspectives on learning.

By themselves engaging in challenging learning experiences, the teachers found ways to understand their own learning, which empowered them to develop alternative ways of structuring learning environments for their students. Through the course of the project we saw strong and varied evidence that to “understand in a different way” increases potential for alternative actions [9, 7, 11, 2].

2.5 A Community of Learners

Senge (quoted in [17]) says that the first step in trying to develop a learning community is to find a group of people who are interested in doing things differently, or people at least willing to take a chance and make that leap of faith together. Next, the membership of this learning community needs to be as diverse as possible to ensure richness as people from multiple constituencies and at all levels work together collaboratively and continually. This diversity contributes to community members’ “enhancing their capacity to create things they really want to create” [17] (pp. 20).

We realized that we needed to include researchers, academics, and practitioners in a new group intended to support teacher learning, in order “to move beyond a balkanised system the fault-lines of which appear to exist around theory and practice, to a more learning- and learner-centred approach where all contribute” [23]. Support from others within the community enables a powerful learning environment, as other people are the greatest source of alternative views needed to stimulate new learning [10].

With this understanding, we developed a custom website for teachers to use in sharing their work with each other [19]. Members of the EM community had opportunities to interact personally as well as through a dedicated web site designed to support individual reflections and communication among participants (**Fig. 2**).

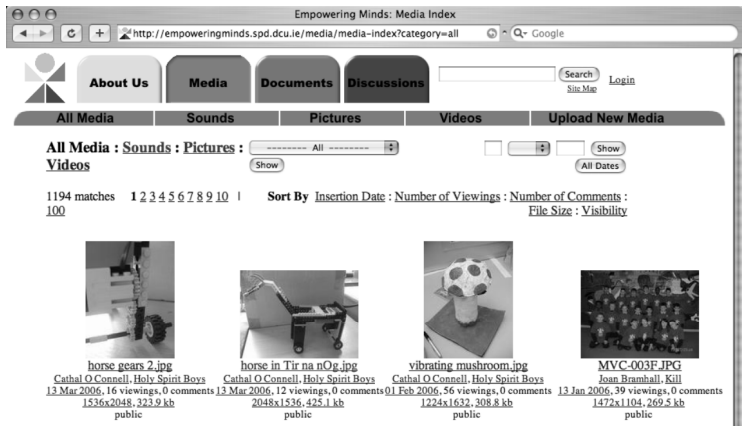


Fig. 2. Empoweringminds.ie Community Web Site

2.6 Sustainability and Scalability

During the first year of the project (1999) collaboration with Media Lab Europe began and four schools became involved, representative in size and socioeconomic range. The nine teachers spanned a range of ages and experiences, and included roughly equal numbers of men and women.

In the second year the project grew to include 13 schools and 29 teachers, particularly involving small rural and disadvantaged schools. They spanned a wide geographic area, including Dublin, Kildare, Kilkenny, Tipperary, Louth and Sligo.

Since 2003 the EM project is now growing beyond primary schools to an urban community. The Digital Hub, a high-tech office park located in an urban area that is undergoing renewal, operates a The Liberating Learning project includes 20 teachers in 10 primary schools, 10 teachers in 5 post-primary schools, and will soon include 2 YouthReach groups [6]. The number of children involved now exceeds 1200. Teachers who participated in EM will help to form the supportive community around the new teachers.

An interrelated network of supports enabled sustaining and scaling the EM project. More and less experienced teachers and children worked together within each classroom; classrooms worked together within each school; the participating schools cooperated and collaborated; the project coordinator traveled from school to school in order to provide materials and advice; and the resource team was available throughout.

In the next section of the paper, we present details of how the EM project unfolded in one particular classroom over the course of the 2002–2003 academic year.

3 A Collaborative Project Among Four EM Schools

Many of the teachers wanted to encourage cross-school collaboration among schools in the EM community. They wanted more than comments on particular models or once-off short interactions when they encountered a problem and needed help.

Through discussion at the summer workshop and other group meetings they realized that sustained collaboration would not happen by chance; they needed a common purpose to guide their efforts. To this end the teachers from four schools decided that they would work on a shared project theme. The schools were:

- Ballymun Boys National School (N.S.) – a large urban disadvantaged school in Ireland's capital city (Dublin);
- Kill N.S. – a medium-sized semi-rural / semi-urban school in southeast Ireland;
- Kilvemnon N.S. – a small two teacher rural school in southwest Ireland; and
- Stokane N. S. – a small two teacher rural school in northwest Ireland.

In keeping with the EM community's narrative theme [16], the collaborators chose to work with the "*Díarmaid agus Gráinne*" epic in Irish folklore [5], thus reflecting their shared national identity. This is a wonderfully varied story of love, adventure, action, pursuit, and excitement, liked by male and female, young and old alike.

3.1 Appropriating a Story

The time limit for this collaborative venture was 4 to 5 months, as participants wanted to display their joint project at the EM community's annual project exposition at St. Patrick's College, Dublin. The teachers decided to split up the story in order to encourage collaboration and to make sure the entire story was illustrated. After much back-and-forth communication among teachers and children, each school identified the parts of the "*Díarmaid agus Gráinne*" epic that they wanted to construct.

They chose 8 scenes from the story. Each school was to construct 2 scenes, one using LEGO Mindstorms materials with the programmable bricks and the other using clay models and the Microworlds or iMovie programming environment to create an animation. Each of the scenes would then be integrated to form a full rendition of the story.

The teachers believed that adopting this approach would lead to meaningful and sustained engagement. There would be a genuine reason for the collaboration as participants now had a shared context with shared problems. The schools were disparately located, so the main form of collaboration would have to be e-mail communication and the web site platform. Here we focus on one of the small rural schools (Kilvemnon N.S) in order to illustrate the depth of engagement with learning using a range of expressive computational materials that this collaborative project enabled.

3.2 Conor's Reflections

One of the teachers at Kilvemnon N.S., Conor, kept a reflective diary¹ during the development of the school's project constructions. This account details the range of materials used, the types of problems that emerged, the modifications to the story resulting from negotiations with materials, and Conor's appeals to the larger group for help. His account also demonstrates how the project facilitated integration of many aspects of curriculum, including heritage and culture, history, science and engineering, problem solving, writing, arts and crafts, design and construction,

¹ See <http://empoweringminds.spd.dcu.ie/discussions/reflections/one-reflection?id=35>

communication and collaboration, and language development. The latter had several manifestations through dialog among participants as they sorted out the story, decided on how to use the materials to illustrate parts of story, and explained and negotiated modifications to the story and its alternate endings.

3.3 Díarmaid's Escape

Here is the students' narrative as presented by the teacher:

The Lego scene we have chosen describes Díarmaid's escape from a fort he has built in the middle of a forest. He has built a stockade with 7 doors in it around the fort. His enemy Fionn surrounds the stockade, so Díarmaid cannot escape.

In our scene we want Díarmaid to walk around to each of the 7 doors, pause at each one to find if there is a means of escape. At the seventh door he realises that Fionn is outside it with lots of soldiers. So Díarmaid decides to make his escape here.

He actually pole-vaults out over the stockade using his spear, and lands well beyond the bank of soldiers surrounding the stockade. We want to get a good model of Díarmaid built to walk around to each door and stop, get each door to open, and finally at the seventh door get Díarmaid to escape over the stockade. (Conor's diary, Feb. 2003)

In order to enact this scene Díarmaid must be able to walk around the stockade and pole vault out. Initially the children's idea was to use a magnet to move him around and to build a crane to lift him, to give the impression that he was pole-vaulting out over the walls of the stockade.

However, after many different design attempts, they discovered that their magnet idea wouldn't work: the board they were using as the base was too thick, and even varying thicknesses of board did not solve the problem.

The children then decided to embed a programmable brick within the Díarmaid model. This meant that the model was now autonomous: it could be programmed to move around as required. Previously, the children had built projects using light sensors to form a path guiding the movement of an autonomous robot. In their plan for Díarmaid, he would follow a black line as he moved from door to door looking for a way out of the stockade (**Fig. 3**).

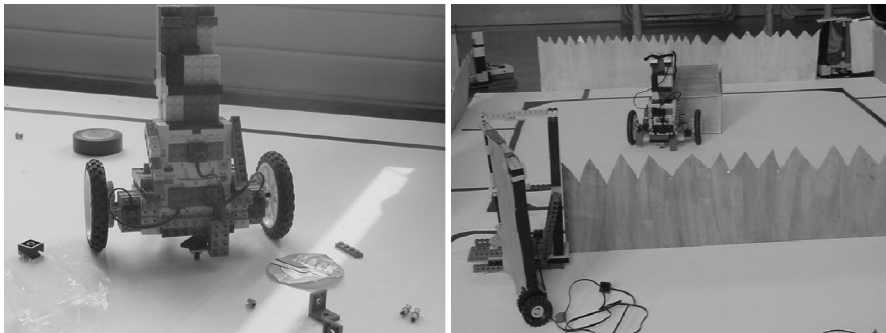


Fig. 3. Díarmaid with programmable brick (left); Díarmaid in the stockade

This new design caused other problems. Díarmaid was now very heavy, and despite several different attempts at designing an appropriate crane, the children could not construct one capable of launching heavy Díarmaid out of the stockade. It remained a problem to decide how he could pole vault away from his enemy Fionn and the awaiting army.

3.4 Control and Ownership

Unable to construct a scenario that remained true to the original storyline, the children and teacher discussed at length what to do. They decided they could program Díarmaid to approach each door looking for Fionn, as outlined in the original story. Then he could send a signal to the door in order to open it and see what's behind it. If escape were not possible he would move on and try each door in turn. This solution meant that Díarmaid needed a new program each time he moved to a new door. However, the brick can only store 5 programs at a time. After considering numerous possibilities, they changed the story from 7 to 5 doors. However, as Conor notes:

We still have a problem with getting the Díarmaid lifted out over the wall... so we'll have to use our imaginations to create a 'twist' in the story, to get him to escape in a different way. (Conor's Diary, March 2003)

They decided that, because of their crane-building difficulties, they would take poetic license and change Díarmaid's escape strategy. He would no longer pole-vault, but would trick Fionn by faking an exit from door 1 and escaping through door 5:

Once they had made the story their own, the main programming elements were getting Díarmaid to follow a black line, sending a message to each door's programmable brick, writing the programme for each door to open, and ensuring that Fionn would move from gate 5 to gate 1 at the right time. **Fig. 3** shows the work.



Fig. 4. Chess scene for claymation; learning chess with computer software

3.5 Use of Claymation

In addition to expertise with LEGO Mindstorms, EM teachers have attended workshops where they learned how to do claymation-animation using digital cameras and Microworlds software. Conor led his students into such a project.

The children created clay figures and animated Díarmaid hiding in a tree and dropping berries to help Oisín beat Fionn in a game of chess. A direct outcome of this part of the project was that all the children learned how to play chess – in order to

illustrate how Díarmaid helped Oisín to win the game, they had to understand the moves and strategies (**Fig. 4**).

Microworlds software requires capture of individual digital photographs, plus programming the timing between them in order to create the illusion of fluid motion. The volume of photographs to be taken, the complexity of the timing, and the sequence pauses motivated Conor to try Apple iMovie to see if it was a better solution.

You can cut out pieces you don't want, slow down the clips (the first three clips in this piece are slowed down), put in voice-overs and backing music, titles, subtitles etc. The big advantage is that the finished clip can be exported to Quicktime, so it can be viewed on any computer, used in Presentations eg PowerPoint etc. For anyone doing the Claymations it's worth giving this method a go. I know it takes the programming element (as used with Microworlds) out of the project, and teachers rightly feel that using the Microworlds with the children is important. I'm not arguing that we shouldn't use Microworlds for Claymations, but using the iMovie is worth a try. (Conor's Diary, May 2005)

Conor's diary describes in detail the development of the claymation scene and provokes consideration of trade-offs between materials and aims. His level of digital fluency, which developed considerably as a result of his engagement with the EM community, is fully apparent. Conor also displays an ability to make independent decisions as he discusses what technology is appropriate to do what at a given time. Encouraged to explore, he does not accept materials as given, to be used in a rote manner, but can decide how to use them for his own purposes. This confidence in decision-making indicates deep understanding of the technology and the spirit of the project, as well as strong senses of identity, capability, and ownership.

4 Conclusions

As teachers and children work together, EM schools have become "learning organizations" [20]. Project leaders, researchers, and all participating support each teacher and child in developing their learning and their individual identities. Likewise, across the network of collaborating schools, participants work to cultivate and maintain the collective identity of their locale, as well as to develop control and ownership of their contributions and their learning. Further, cross-school collaborations built on an aspect of shared national identity through interpretations of an epic story from Irish folklore.

Constructive projects with expressive materials in Atelier-style learning environments provide evidence of learning about self and computational ideas beyond what any test could measure, provoking challenges to the epistemological foundations of traditional schools and curricula. Rather than perpetuating models of everyone learning the same fixed knowledge in the same way, the protean quality of digital materials enables individual exploration and expression as learners engage ideas that span disciplines.

EM demonstrates that it is possible and desirable to support individual styles and local identities while achieving digital fluency. Technological development does not

have to mean being swallowed in globalisation or general patterns of learning, but instead can enhance particularity, individual realisation, and personalised learning. Most importantly, the project models how working with digital technologies can promote the values we want to live by, nurture the kind of people society needs, and develop the very future we want to create.

Acknowledgements

Ireland's National Centre for Technology and Education funded the first phase of the project. Additional funding came from Ireland's Higher Education Authority, supporting collaboration with Media Lab Europe. Continued funding comes from the Liberties Learning Initiative through Diageo Ireland, which includes Guinness. Further support is provided by St. Patrick's College and by the National Centre for Technology and Education. Senior advisors included Hugh Gash (St. Patrick's College) and Seymour Papert (MIT Media Lab). Glorianna Davenport, Jamie Rasmussen, and John Bilotta also made contributions to the project, including working directly with teachers and students. Teachers who participated in Empowering Minds will help to form the supportive community around the new teachers.

References

1. Butler, D., Martin, F., and Gleason, W. "Empowering minds by taking control: Developing teachers' technological fluency with LEGO Mindstorms." In *Proceedings of the SITE 2000*, pp. 598–603, Charlottesville, VA, 2000.
2. Butler, D. *Self-determined Teacher Learning in a Digital Context: Fundamental Change in Thinking and Practice*. Unpublished doctoral thesis, Dublin City University, 2004.
3. Cavallo, David. *Technological Fluency and the Art of Motorcycle Maintenance: Emergent Design of Learning Environments*. Unpublished doctoral thesis, MIT, pp. 225, 2000.
4. Chafee, R. The Teaching of Architecture at the Ecole des Beaux-Arts. In A. Drexler, ed., *The Architecture of the Ecole des Beaux-Arts*, MOMA, New York, 1977.
5. *Díarmaid agus Gráinne*. <http://timelessmyths.com/celtic/ossian.html#Pursuit>.
6. The Digital Hub Liberties Learning Initiative. <http://www.thedigitalhub.com/>.
7. Dunne, Joseph. *Back to Rough Ground: 'Phronesis' and 'Techne' in Modern Philosophy and in Aristotle*. London: University of Notre Dame Press, 1992.
8. Freire P. *Pedagogy of the oppressed*. London: Penguin, 1972.
9. Gadamer, H.G. *Truth and Method* (Wahrheit und Methode, 2nd ed., 1965) Translation edited by G.Barden and J. Cumming. London: Sheed and Ward, 1975.
10. von Glasersfeld, E. Cognition, construction of knowledge and teaching. *Synthese*, 80, 121–140, 1989.
11. Grundy S. *Curriculum: Product or Praxis*. London: Falmer Press, 1987.
12. Knowles M., Holton, E., Swanson A. *The Adult Learner: the definitive classic in adult education and human resource development*, (5th Edition), Butterworth-Heinemann, 1998.
13. Kuhn, S. Learning from the Architecture Studio: Implications for Project-Based Pedagogy, *Intl Journal of Engineering Education*, Vol. 17, Nos. 4 & 5, pp. 349–352, 2001.
14. Lindeman, Eduard. *The Meaning of Adult Education*. New York: New Republic, 1926.

15. Martin, F. *Circuits to Control: Learning Engineering by Designing LEGO Robots*, PhD dissertation, MIT Media Laboratory, pp. 65., 1994.
16. Martin, F., Butler, D., and Gleason, W. "Design, story-telling, and robots in Irish primary education," *Proc. of IEEE Systems, Man, and Machine Conf.*, Nashville, TN, 2000.
17. O'Neill, John. On Schools as Learning Organizations: A Conversation with Peter Senge. *Educational Leadership*. April 1995, pp. 20-23
18. Papert, Seymour. Perestroika and Epistemological Politics, in Harel & Papert (eds.) *Constructionism*, Norwood, NJ: Ablex Publishing, pp.13-27., 1991.
19. Rasmussen, J. Butler, D. Davenport, G. "A web-based environment for assembling multimedia learning stories in Irish primary education." In Proc. IEEE ICALT 2002.
20. Senge, Peter. *The Fifth Discipline*. Randon House, 1990.
21. Senge, Peter, et al. *Schools That Learn : A Fifth Discipline Fieldbook for Educators, Parents, and Everyone Who Cares About Education*. NY: Doubleday-Currency, 2000.
22. Strohecker, C. Media Lab Europe (promotional brochure). Dublin, 2002.
23. Sugrue, C., Morgan M., Devine D., & Rafery D. *Policy and Practice of Professional Development for Primary and Post-Primary Teachers in Ireland*. Report commissioned by the Research and Dev't Committee, Dep't of Education and Science. pp. 37., 2001.
24. Ueda, Nobuyuki. *Multimedia Unplugged: A Workshop on Learning Designs at the neoMuseum, Japan* in Articles by CRN (Child Research Net) Advisory Board Members. From <http://www.childresearch.net/CYBRARY/MABM/MEMBER6.HTM>, 1999.

Designing Digital Technologies for Layered Learning*

Ken Kahn¹, Richard Noss¹, Celia Hoyles¹, and Duncan Jones²

¹ London Knowledge Lab, Institute of Education, 23-29 Emerald Street,
London WC1N 3QS, UK
{K.Kahn, R.Noss, C.Hoyles}@ioe.ac.uk
<http://www.lkl.ac.uk>
² firbush@hotmail.co.uk

Abstract. Designing digital technologies for deep learning is a highly non-trivial enterprise. In this paper, we discuss one approach we have adopted that seeks to exploit the possibility of affording diverse layers of engagement that exploit the interconnectivity available on the web. In a nutshell, we describe a system that offers learners the possibility of engaging with difficult scientific and mathematical ideas without the necessity of interacting with complex layers of symbolic code – while making that interaction available at all times.

1 Introduction

Our prior research concerning computer games [1] has indicated that games designed for learning authored by students tend not to suffer the difficulty noted with educational games in general around poor production values. In the case of constructed games, even if they are relatively simple and crude, students tend to become dedicated to their creations. However acquiring the skills to make computer games requires a major investment in time and effort. Here, we present some novel design research that will illustrate an approach that aims to reap the advantages of learning by programming games without the disadvantages of having to know *a priori* how to program.

Our starting point derives from the principle that a powerful way for students to learn mathematics is through their long-term engagement in collaborative projects for which they take responsibility individually and collectively (see, for example, [2]). From this basis we have developed the following principles for designing for this learning:

- sequenced activities that can engage students at a *variety* of levels in what we name *layered learning*;

* This paper is a greatly extended version of Kahn, Noss, Hoyles and Jones (in press): Designing for diversity through web-based layered learning: a prototype space travel games construction kit. Submitted to the ICMI 17 study group on *Technology Revisited*.

- *flexible* tools that have adjustable parameters, can be combined in different way and can also be programmed, and thus allow students to investigate each activity for themselves¹;
- *collaborative interactions* as part of the activity sequence through which students discuss their emerging ideas in the context of their game design;
- *multi-player game playing* either at one computer or over the web.

The second design requirement is worthy of particular attention, as it implies that designers can both empower and constrain students by offering a set of programmable components or modules that can be customised to suit diverse students' goals, as well as tuned to the knowledge domain. Programmable modules have the added advantage of providing a consistent way to combine and modify tools, in the control of the learner.

There have been numerous attempts to design a programming-based approach to learning over the years. The most successful have achieved tangible learning outcomes across various topics (music, mathematics, language, physics (some of these are discussed in [3])). They have also provided important pointers to the possibilities of learning that transcends the procedural and superficial by encouraging a playful – yet mindful – spirit of enquiry on the part of learners, aiming to break down the curricular silos that so often characterise traditional schooling. For the most part, what has been missing has been generalised success in tapping into students' own interests on a wide scale and engaging them in debate, investigation and production. To achieve this aim, we have built and tested a Space Games Construction Kit (SGCK) along with a narrative 'metagame' to assist learners in navigating their way through the sequence of activities of design and game playing. This work can be seen as a continuation of the research we have undertaken within the WebLabs Project [4]. In WebLabs children constructed models by assembling program fragments and then published their models on the project web site. These reports were then read and commented upon by other students who downloaded and ran the models. The programs were constructed in ToonTalk [5], and this enabled students to build programs from scratch. In contrast, in the SGCK the programs were pre-built and are in a dialect of Logo called Imagine Logo [6]. So, whereas the students in the WebLabs project needed to master the programming system (see Harel [7] and Kafai [8] for studies of children engaged with game construction through programming from scratch), those engaged with the SGCK could interact without any programming knowledge if they so desired, and were unable to construct programs from scratch or make major program alterations.

2 Description of the Game Environment

The game environment consists of three major components: the game construction kit, the metagame, and usage scenarios. We will describe these in turn.

2.1 The Space Travel Games Construction Kit

We began by considering the classic computer game of *Lunar Lander*. In the process of trying to land upon the moon, a player engages with the laws of motion, playing in

¹ In the current research aiming to involve groups spread geographically it was important that the tools could be accessed through a web browser - although this is not a 'principle'.

a virtual world where the laws of gravity and momentum are not obscured by friction or atmospheric drag as they are on Earth. The Space Games Construction Kit can be used to build a variety of games similar to *Lunar Lander* (see Sherin [9] for a different approach to a similar problem). The construction kit provides small program fragments together with tools for customising and composing them. Thus the software extensions allow *layered* exploration appropriate for populations of students with differing backgrounds and ages; that is, users can choose how far to delve into the workings of the tools.

The innovative aspects of the software include:

1. A child-friendly interface for the composition and parameterisation of pre-built program fragments;
2. An underlying computation model that has been simplified and made composable by building upon multiple independent processes. This is critical for the deeper levels of engagement where students inspect and edit program code;
3. An underlying physics model based upon conservation of momentum that is simpler than the one commonly used based upon first and second derivatives of position and the first derivative of momentum. Yet it is capable of modelling the same phenomena;
4. The concept of a “metagame” where games are made within an overarching narrative game structure.

2.2 The Narrative Metagame

The metagame starts with the player being hired as a game developer at a game company. The player receives help from a team of simulated experts including a programmer, a scientist, a historian, an assistant game designer, and an animator. A player may skip over all this and jump into game making, but the metagame motivates the scenarios within a project format, as well as providing structure, background information, guidance, and a gradual introduction of new features and capabilities.

The metagame embodies the design of a learning sequence. Learners are presented with a goal and need to interact with their virtual teammates in order to acquire both the required components and the knowledge to proceed. The response of each teammate to a visit by the player is scripted but also depends upon both the current state of the game being constructed by the player and the history of the player's interactions with all the teammates. This gives the player freedom to visit the teammates in any order and with any frequency. Furthermore, each game component has an associated help button. When a component's help button is pressed, the player is informed which teammates have something to say about it. For example, the programmer, the scientist, the historian, and the game designer all have a unique perspective when giving an explanation of the component which implements gravity.

The first task in the metagame is to acquire program fragments and artwork to build a very simple game based upon an astronaut who is adrift and needs to reach her space ship. At first this is accomplished using only components involving horizontal motion. Next the game maker is faced with designing conditions to separate player

success and failure. The next task is to give the ship a vertical velocity, and then to add vertical acceleration to the astronaut. By combining the horizontal and vertical components the game elements can be programmed to move diagonally. The final game in this sequence involves making the game into a cooperative game between two players over a network. Although the metagame guides the learner along a designed path, he or she is free at any point to invent and play games other than those in the metagame's sequence. To proceed, to the next phase, however, the required game making tasks need to be performed.

The next part of the metagame is to build a Lunar Lander game. Here gravity is introduced. Learners are faced with making decisions such as what speed constitutes a safe landing. Next we introduce player-configurable gauges which help in the task of landing safely on the moon without using too much fuel. The next challenge is to construct the best autopilot program. This is accomplished by recording a good "manual" landing and then editing the parameters of the recorded landing to produce an optimal landing. A two-player version is then introduced where players can compete to land with the safest landing speed using the least amount of time or fuel.

2.3 The Activity Sequence

The students in the first part of our study were given the following written instructions. The second group of students instead relied upon the virtual teammates of the metagame that provided guidance and background science and mathematics.

Phase 1: an exploration of game making.

- *Moving in outer space:* An astronaut is adrift and needs to reach her space ship. Build a game by acquiring program fragments and artwork, which can be accomplished using only components involving horizontal motion that is achieved by 'throwing' rocks (previously collected by the astronaut).
- *Agreeing upon some constraints:* for example, will the astronaut get back safely? Is she going too fast when she hits the spaceship?
- *Creating a new game:* Now invent a new game. For example, the space ship has started to move off in a vertical direction. Can the astronaut reach it now? The final challenge in this sequence involves making the game into a cooperative game between two players over a network.

Phase 2: Building and playing a Lunar Lander game. To proceed to this phase, all the required game-making tasks in Phase 1 need to be performed. Here gravity is introduced.

- *Landing on the Moon:* for example, what speed constitutes a safe landing? At this point we introduce dynamic configurable gauges to measure speed, acceleration, mass and other parameters which monitor and graph these values and can help in the task of landing safely on the moon. Added challenges are introduced: e.g. do not use too much fuel!

- *Using the Autopilot:* The settings of a 'manual' landing consists of all the variables and how they are changed can be captured by an autopilot. The next challenge is to construct the best autopilot program, by recording what is deemed a good landing and then tweaking the parameters of the recorded landing to produce an optimal landing.
- *Multiplayer game over the net:* Players can compete for example to land with the safest landing speed using the least amount of time or fuel. Students are also challenged to invent new games to play.

3 The Potential Layers of Learning

Both the metagame and the construction kit were designed to support *layered learning*. At the first layer, engagement is mainly through reading, watching and making conjectures based on observation and for example describing a motion and reflecting on it. The tools for this layer are basic, perhaps only a handful to control a simulation on video, or the timing of movement. At a second layer, the learner can begin to manipulate behaviours that define various sorts of motion and predict and test out the effects of different values. At a third layer, the learner might explore further how variables relate to each other, for example position, velocity and acceleration, by reference to the values set by sliders. And finally, a fourth layer that engages with these relationships either by modifying existing programming code or by writing new programs or fragments of programs.

In a follow-up study we provided 'behaviour gadgets' that support rotation, grasping, releasing, and springs. Using these building blocks, students can build a detailed model of the astronaut in space rotating her arm while holding a rock and then releasing it. This enables the students to take the act of throwing, that previously was an atomic action, and explore a deeper layer involving additional concepts such as angular momentum. Students can delve into deeper layers of learning in different domains such as physics, mathematics, computation, or game design.

4 Game Construction: The Interface

When the player is ready to build the game, a game panel with images of the astronaut and lander is presented (see Figure 1). Beside it is a control panel with a start button. Pushing the start button initially does nothing, since the game elements have yet to be given programs. The control panel also has a button that causes the *behaviour gadgets panel* to appear (Figure 2). It contains behaviour gadgets that consist of one or more *code boxes*. A picture can be given a behaviour by placing a behaviour gadget on its back. The behaviour can be altered by setting sliders on the gadget's *settings* page. The code boxes of a behaviour gadget can be removed, whereupon they expand to display the code that implements the behaviour. Portions of the code that can safely be edited without programming expertise are colour highlighted.

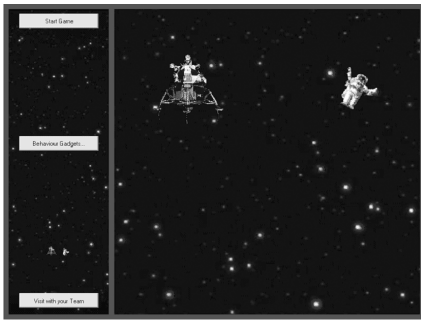


Fig. 1. The initial game construction page

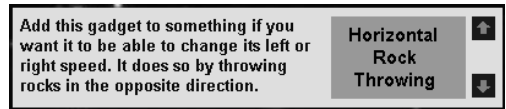


Fig. 2. Behaviour gadgets can be dragged from the Behaviour Gadgets Panel. Initially, the behaviour gadgets panel contains only a horizontal velocity gadget and a horizontal rock throwing gadget. As the metagame progresses, the behaviour gadgets panel acquires more elements.

A behaviour gadget contains buttons for setting parameters and obtaining help from the virtual teammates. It contains at least one code fragment. Removing a code fragment from a behaviour gadget causes it to expand to full size as illustrated in Fig. 3.

HORIZONTAL VELOCITY CODE

```
; The following causes my position to be updated by my velocity.
; It updates 100 times a second by 1/100th of the velocity.
; The velocity is how much it moves in a second.
; This code only applies to motion to the left or right.
repeat_every 1/100
  [change_my_horizontal_position_by
    my_horizontal_velocity/100]
```

Fig. 3. The Horizontal Velocity Code Box after removed from the Behaviour Gadget

As a layered learning design, our software includes code fragments that students may, but need not, read and edit. To encourage delving deeper, some behaviours cannot be altered by moving sliders. Instead, the student is encouraged to change the code to improve the game they are making. One example of this is the code box for running out of air, which is a part of the reached or missed ship gadget. The game is too easy (and boring) if one can very slowly drift back to the ship. Running out of air introduced an intrinsic time limit.

The total mass of rocks (i.e., total fuel), the largest rock (the maximum rate of fuel usage), and the rock velocity (the propellant velocity) can all be adjusted by moving sliders. As one does so, the system makes calculations to show derived values such as force and to perform unit conversions. These parameters reflect real engineering tradeoffs. For example, adding more rocks/fuel does increase the duration of manoeuvrability but at the cost of a greater total mass and hence a smaller acceleration from identical rock throws.

Limitations of space prevent us from illustrating a range of further panels, behaviours and other objects that control instrumentation (for example, gauges that monitor any of 13 values in graphical or numerical displays, including velocity, acceleration, remaining fuel, total mass, the application of thrust (by throwing rocks out in the opposite direction), autopilot facility (a recording of all the changes to

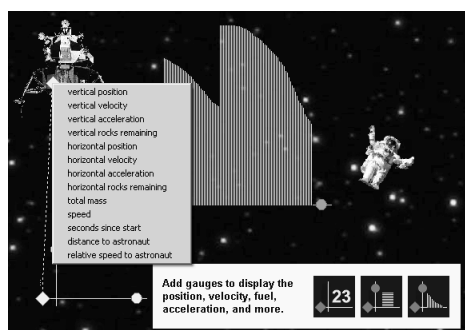


Fig. 4. A snapshot of a dynamic graphing gauge displaying the vertical position of the astronaut as she falls and the connection of a second gauge to attributes of the lander

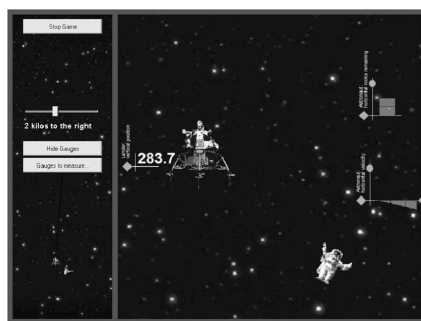


Fig. 5. A snapshot of game play with three active gauges

thrusters made manually), and a two-player version of the Lunar Lander game typically involving a race to be the first to land safely on the moon.

5 Some Illustrations of Learning

We now very briefly outline some of the learning issues that are emerging from the iterative design/test cycle with three groups of students: two drawn from a large, urban comprehensive school (one "Year 7" class aged 11-12; one "Year 8" class, aged 12-13) and a small group of 3 students (aged 12-14) from a second school in an after-school setting. We should stress that we are in the very early stages of working with students, and that up to now, we have considered students (and their teachers) as primarily co-designers of the emerging system, rather than as 'subjects' with whom to evaluate learning. The learning issues we outline here, therefore, should be regarded as a tentative set of issues for further exploration, rather than a definitive list of 'learning outcomes'.

Developing understandings of Newton's third law: In the first task the Year 7 students began with a relatively basic knowledge of the physics concepts involved. For example, R suggested, "you could throw some rocks away and that would make her lighter so she would move." She was apparently unaware that the effect of gravity in space is negligible (in the setting in which the game takes place). Through the course of the activity and experimentation with the horizontal rock thrower R and her coworkers appeared to develop some appreciation that throwing a rock would develop an opposite movement proportional in velocity. Indeed, later in the session two of the students worked with the theory that "throwing larger rocks makes her move faster." *Minimising time* (of astronaut to spaceship, or lander to moon) proved a motivating task, particularly for the Year 7 students. Most students used an iterative strategy, e.g., Tom and Alex were delighted to refine their strategy again and again by optimising the use of the horizontal rock thrower against the speed of reaching the spaceship.

Using the gauges: Throughout the sequence much use was made of the gauges and interpretation of their output. Students found the gauges relatively easy to put in place and interpret and tended to refer to them constantly as a guide to their use of the rock thrower or thrust. When landing on the moon some students applied far too much thrust causing the lander to move upwards and disappear off the screen. Reading the vertical velocity gauge which they had set up for the lander they predicted how the lander would “keep on getting slower until zero. Then it will fall back again because of gravity.” In the two-player game the ability to attach gauges to the opponent’s lander was a particularly successful feature, enabling one group to make a close comparison with the other and to adjust the strategy second by second.

Composing horizontal and vertical velocities: Coming up with the hypothesis that to achieve diagonal movement a combination of horizontal and vertical thrusts would be needed, appeared surprisingly ‘obvious’ to the students and was tested by, for example, using both horizontal and vertical rock throwers to the astronaut and using both simultaneously.

Gravity: None of the Year 7 students immediately made a connection between the rock throwing astronaut and the rock throwers for the lander, although this was evident for all of the Year 8 students. The Year 7s also only had a sketchier concept of gravity. Only two – Tess and Alex – volunteered that the lander game would be different from the astronaut in that there would be a gravitational pull near the surface of the moon.

Collaboration, competition and motivation: Beating previous best scores proved highly motivating, especially for the team of Year 7 boys. Collaboration centred around agreeing what the two teams should have in common; the total mass of the projectiles, an agreed safe landing action, a value for gravity and the vertical starting position of their landers, for which they sought and found a new gauge, previously not used. Attempts to minimise fuel use became more sophisticated. The boys realised that with their agreed safe landing speed of 30 metres per second, they needed only to keep just below this figure to ensure a safe landing and minimal fuel consumption. Before this they had been trying to reduce the velocity to the minimum regardless of fuel use.

In summary, the competitive element of the two-player version appeared as a considerable motivation to the students: they enjoyed seeing the opposition’s ship on their screen and being able to monitor its progress through gauges.

6 Reflections on Design

Overall the software did allow access to diverse students at many layers of learning, it stimulated huge interest and discussion (both inside and outside classrooms) and students used quite sophisticated ideas in pursuit of their game making and playing. The students came up with many ideas for improvement. For example, they suggested the need to reduce the amount of reading required in the initial stages and simplify some terminology that was too complex in places. Further suggestions included that a choice should be offered between reading and listening to instructions and that more complex instructions might be communicated through demo buttons or tutorials – for example showing them how to set up a gauge or to use a behaviour gadget. Perhaps

the most interesting suggestion was that the software might be structured into what some students described as “levels”, such as those commonly found in computer games.

Where the lunar lander software failed to meet the initial expectations was in giving students easy access to the programming code and in creating situations where they would want to analyse and adjust that code. It would seem possible that if the software were remodelled into a series of levels – corresponding in some way to the previously defined expected layered learning model – analysis and use of the relationships inspectable in the programming code or as recorded by the autopilot could become not just a real possibility but an integral part of the game.

When designing a toolkit to be used to construct scientific models (and games based upon these models) one needs to determine the underlying ontology of the system. How should time be modelled? How should concurrent processes behave? What are the primitive notions of motion and space and which concepts are to be constructed from those primitives? Are the usual ways of conceptualising the laws of motion that are based upon algebra and calculus optimal for computational modelling?

We chose to build upon a discrete conception of time and forces, by providing behaviours that create independent simultaneous computational processes that implement horizontal and vertical velocity, horizontal and vertical thrust, and gravity.

We created new computational primitives upon which everything else rests. One is called *repeat_every*. Its first argument defines the frequency with which the second argument is executed. For example, constant gravity near the surface of the moon is implemented by the following code fragment:

```
repeat_every 1/100  
  [change_my_vertical_velocity_by  
    value_of_slider_for_gravity/100]
```

This causes the vertical velocity of the associated object to be increased by the value of gravity (or decreased if gravity is negative as it usually is). To approximate the continuous change of velocity this program is run every 1/100th of a second. The concept of a sampling rate is relied upon here.

Note that simultaneous with the execution of the gravity code fragment there can be other processes that are also incrementing or decrementing the vertical velocity, perhaps due to the use of thrusters (or rock throwing). The relative ordering of the execution of these processes on a sequential computer will have no material effects.

We chose to make the notion of position be the only computational primitive for modelling motion. So when the code implementing the vertical velocity runs, it relies only upon this primitive to change the vertical position by the product of the current vertical velocity and the elapsed time (1/100th of a second in this example).

```
repeat_every 1/100  
  [change_my_vertical_position_by  
    my_vertical_velocity/100]
```

Alternatively, we could have chosen velocity as the only primitive notion. Then the updating of position would still be a function of the current velocity but in an opaque manner. If both position and velocity were primitive notions, then there could be confusion if both were used as control variables. When any part of the program changes the position, should the velocity change accordingly? Does velocity measure something or specify something? Doing both can lead to confusion. It also hides the mechanism relating them.

While position is the computational primitive underlying the construction kit, it is velocity that is the physical primitive. It is implemented in terms of position and time but other behaviour gadgets are expressed in terms of changes in velocity. It is interesting to note that perhaps speed (velocity without concern for direction) is the human psychological primitive. Piaget, for example, suggested that speed is primitive and that time is not, but instead understood in terms of speed [10].

Another epistemological question is how to model forces. Should forces define acceleration, which in turn defines velocity, which defines position? A sequential program that models all the processes could be built in this way, but it would lack the modularity and composability of the concurrent processes on which we rely. Consider the difficulties that would arise if one process implemented gravity by setting the acceleration to the appropriate value, while a thruster process implemented force by adding to or subtracting from the current acceleration. Clearly, the order in which the gravity process and the thruster processes run will drastically affect the model.

One reason we chose to build upon the conservation of momentum, rather than $F = ma$, is that it provides a concrete and discrete way to think about forces. We suggest that this approach is likely to be more accessible than the alternative, which relies upon a notion of continuous rates of change. It also makes the mechanism underlying rocket thrust transparent. Throwing a one kilogram rock once per second is the same mechanism as real rocket thrusters that “throw” a trillion trillion molecules (“rocks”) per second. Force is the derivative of momentum and as such seems a more complex and difficult notion than the discrete change in momentum that underlay our approach.

Our de-emphasis of the concept of force is consistent with modern physics. Nobel laureate Frank Wilczek, recently wrote in “Whence the Force of $F = ma$?” [11]:

... the concept of force is conspicuously absent from our most advanced formulations of the basic laws. It doesn't appear in Schrödinger's equation, or in any reasonable formulation of quantum field theory, or in the foundations of general relativity...

... If $F = ma$ is formally empty, microscopically obscure, and maybe even morally suspect, what's the source of its undeniable power?

Our perspective on forces is closer to diSessa's concept of momentum flow [12]. Momentum flow is an alternate way to conceptualise elementary mechanics where one considers momentum as something that acts like a conserved fluid that flows within and between objects. Formally, it models the same phenomena as the framework based upon modelling forces and accelerations. Pedagogically, it leads to a promising alternative way of thinking about momentum and forces.

The prototype demonstrates the promise of these ideas in learning Newtonian mechanics. Currently, we have sketched a plan for how the current prototype could be

extended to include universal gravity (i.e. orbital mechanics) and atmospheric drag. In a subsequent project we are exploring other aspects of mechanics such as the conservation of angular momentum. Clearly these ideas can be applied in many other areas in physics (e.g. special relativity). Some areas, however, such as quantum physics, are particularly challenging.

Games based upon ecology and animal behaviour fit this framework well. Behaviour gadgets could give the elements of the game the ability to hunt or evade predators, eat or starve, reproduce and so on. These behaviours could be customised and combined to create a wide variety of animal behaviours and ecologies. [4]

It is plausible that the range of phenomena modelled by the NetLogo [13] and StarLogo [14] communities can fit into our framework. By doing so they would benefit from the structure and guidance of metagames for making games and the support for component composition and parameterisation (behaviour gadgets). It would be interesting to explore whether many popular commercial games such as *Civilization*, *SimCity*, or *The Sims* could inspire game making games for learning about history, government, or psychology.

7 Future Research Directions

In addition to more systematic and varied field studies of student use of the Space Travel Games Construction Kit, we have considered about twenty enhancements to the construction kit and the metagame. We also designed several other mini space travel games that could be strung together with the ones described here to provide a richer and deeper experience. One promising direction for future research is to explore monitoring or debugging tools to help students understand the execution of their programs in detail.

Acknowledgement

We acknowledge the funding of the BBC, and the helpful comments of colleagues in the London Knowledge Lab, notably of Gordon Simpson and Diana Laurillard. Ivan Kalas designed and implemented the user configurable gauges. Peter Tomcsanyi and Ivan provided invaluable advice and technical support.

References

1. Noss, R. & Hoyles, C.: Exploring Mathematics through Construction and Collaboration. In K.R. Sawyer (ed.) *Cambridge handbook of the Learning Sciences*, CUP, Cambridge. (in press)
2. Harel, I., & Papert, S. (eds.): *Constructionism*. Norwood, Ablex Publishing Corporation, New Jersey (1991)
3. Noss, R. & Hoyles, C.: *Windows on mathematical meanings: learning cultures and computers*. Kluwer, Dordrecht (1996)
4. Simpson, G., Hoyles, C. and Noss, R.: Exploring the mathematics of motion through construction and collaboration, *Journal of Computer Assisted Learning* 22:2 (2006) 114-136

5. Kahn, K.: ToonTalk: An Animated Programming Environment for Children, *Journal of Visual Languages and Computing*, 7(2) (1996) 197—217
6. Blaho, A., Kalas, I.: Object Metaphor Helps Create Simple Logo Projects. *Proceedings of EuroLogo*, University of Sofia, Bulgaria (2001)
7. Harel, I.: "Children as software designers: a constructionist approach for learning mathematics", *The Journal of Mathematical Behavior*, 9 (1) 4 (1990) 3-93
8. Kafai, Y.: Children as designers, testers, and evaluators of educational software, *The design of children's technology*, Morgan Kaufmann Publishers Inc., San Francisco, CA (1998)
9. Sherin, B.: A comparison of programming languages and algebraic notation as expressive languages for physics, *International Journal of Computers for Mathematics Learning*, (2001) 1-61
10. Piaget, J.: *The child's conception of time*, (Translation: Pomerans, A), New York, Basic Books (1969)
11. Wilczek, F.: Whence the Force of $F=ma$? I: Culture Shock, *Physics Today*, October, (2004)
12. diSessa, A.: Momentum flow as an alternative perspective in elementary mechanics, *American Journal of Physics*, 48, (1980) 365-369
13. Wilensky, U.: NetLogo, <http://ccl.northwestern.edu/netlogo>, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1999)
14. Resnick, M.: *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*, Cambridge, MA, MIT Press, (1994)

ePortfolios in Australian Schools: Supporting Learners' Self-esteem, Multiliteracies and Reflection on Learning

Elizabeth Hartnell-Young

Learning Sciences Research Institute, The University of Nottingham, Jubilee Campus,
Nottingham, UK. NG8 1BB
elizabeth.hartnell-young@nottingham.ac.uk

Abstract. Electronic or ePortfolios are containers for selections of digital items – whether audio, visual, text, or a combination of these – generally used to show individual learning. Large-scale systems are being developed in Europe and the United States, based on specially-designed proprietary or open-source software. In contrast, most Australian ePortfolio projects in schools are small-scale, locally-developed attempts to take advantage of digital formats to develop a range of literacies, express learners' identities and present achievements to various audiences. This paper describes recent school-based examples reported by teachers and students and concludes that teachers believe that important outcomes lie in increasing self-esteem, improving multiliteracies and developing the skills of reflection. It suggests that it is important to build on the current knowledge and experience of teachers and students if the use of ePortfolios is to spread.

1 Introduction

This descriptive paper seeks to look at the development of multiliteracies and student self-esteem among those who are developing ePortfolios in an attempt to draw out themes that characterise the work in this field in Australian schools. Portfolios, as the name suggests, are portable containers for a range of artifacts. Print-based portfolios have long been popular in the visual arts, and, to some extent, in education. With digital technologies, portfolios have become digital or electronic: and are commonly known as ePortfolios. They are containers for selections of artifacts in the form of digital files containing audio, visual, textual material, and combinations of these. EPortfolios are created with a purpose in mind, such as assessment for learning, transition between levels of schooling, or preparation for employment. They consist of specific selections made for the purpose from a much larger repository or archive (the whole collection from which the selection is made) [1]. One person can therefore create several ePortfolios for different purposes by drawing on different selections of material from his or her archive. For this reason, a clear understanding of purpose is essential to the development of useful ePortfolios, and the structure and content of each ePortfolio can be tailored to the purpose.

In recent years in Australia, students in many schools have developed ePortfolios using readily-accessible presentation or web-publishing software. Lately new software has enabled individual ePortfolio developers and institutions to streamline

the process of storing and presenting material. There are broadly three types of software used in schools and in higher education: presentation software using slideshow formats; web authoring; and database software. Several institutions, particularly universities, have developed software to meet their own requirements, while there is also a strong open-source development community for ePortfolio software. Many ePortfolios are stored on web servers, and can be transferred to portable storage such as CD ROM, DVD, iPod or memory stick as the need arises.

Globally, there is growing interest in large-scale ePortfolio projects. Institutions, states and nations are using a variety of home-grown, commercial and open-source software. In Wales, a project launched by Careers Wales and the national government in 2004 created an ePortfolio space for all citizens. Europortfolio, a consortium including The European Institute for eLearning, European Schoolnet, the Centre for Technical Interoperability Standards (CETIS) and IMS Global Learning Consortium in Europe, has called for 'an ePortfolio for every citizen by 2010' (www.eife-l.org). The focus of these projects is often on recording the skills and competencies of learners in order to ensure employability and create data banks of graduate attributes or national workforce competence. In the context of a lifelong learning approach, improving the transitions between different institutions and maintaining evidence of informal learning are becoming seen as important purposes. In Australia, the Commonwealth Department of Education, Science and Training has also been exploring a system of ePortfolios since 2004. Universities such as Queensland University of Technology have implemented the use of web-based ePortfolios for all students, while the TAFE sector in Victoria has developed a site for all TAFE teachers and students (<http://eport.tafevc.com.au/>). There are as yet no large-scale projects introducing ePortfolios into schools.

1.1 Identity and Learning

It seems likely that ePortfolios have become popular in some settings due to the fact that digital formats make it simple to record, organise and publish material, and because the fundamental human needs of identity creation and learning through building knowledge underpin ePortfolio development. Dewey [2] regarded knowledge as a product constructed by people and containing the meaning of objects and events. Knowledge itself is in the form of objects (including principles and theories) to be considered, criticised and improved by the learners [3]. Knowledge building is activity directed outward towards the creation of knowledge itself, while learning is a personal consequence of this process, the aspect that is directed to enhancing one's own abilities and dispositions. As Scardamalia and Bereiter [4] put it, knowledge itself must be in the world, rather than in the mind. EPortfolios have the potential to present both the individual and collective pool of knowledge to a wider audience [5].

Central to learning in the 21st century is the notion of a plurality of literacies and the dynamic nature of becoming, rather than being, literate [6]. People express themselves and make sense of the world through multiple modes: linguistic, visual, audio and so on, in a continual process. *Multiliteracies* [7] is a term intended to acknowledge both cultural and linguistic diversity and the communication opportunities of new media. Those who are multiliterate can express themselves and make sense of the world through multiple modes. Further, they understand and control the media themselves so they can make informed decisions. Highly literate

people can critique the content and the effectiveness of the communication modes they have chosen to use, and their appropriateness for particular audiences. However, as Fehring [8] argues, this new world of literacies and multiliteracies brings with it a need to reconceptualise assessment practices. EPortfolios interact with assessment as they provide containers for material to be assessed, while the various literacies are integral to the development of ePortfolios that communicate with numerous audiences.

Cope and Kalantzis, [9] members of the New London Group, argue that multiliteracies are best learnt when situated in learners' own experience, including explicit teaching for design (in its broadest sense), investigating the cultural context and applying the designs in a new context. Applying their model to ePortfolio development, one would start with a purpose owned by the users, and provide explicit information about the tools, about learning processes, or other relevant needs. Then some critical appraisal by the users would establish which tools best suit the situation, and reflect on the communicated meanings of the content. Finally, transformed practice would see users contributing to the form and purposes as well as to the content of ePortfolios and the tools to develop them in an ongoing project of transformation. Teachers are usually essential to manage the process and assist learners to design and reflect through dialogic interaction.

1.2 Personalising Learning and Self Esteem

Hargreaves [10] has argued for more student involvement in designing learning pathways. Within the concept of *personalising learning* he argues for greater focus on student voice, assessment for learning, new technologies and learning to learn. EPortfolios clearly have the potential to support these aspects of learning. There is a continuum of user activity from consumption of digital material via the Internet and mobile delivery methods; through reproduction, such as compiling a digital presentation from other sources; to creation, such as making new digital products [11]. Enabling users to create and distribute their own work through an ePortfolio makes them active participants in the process of creating culture [12], and computers can help the learning process as users create knowledge through the digital medium.

Negotiation and representation of identity, or perhaps more accurately, the concept of multiple identities, is fundamental to the development of ePortfolios. Giddens [13] suggests that the fact that individuals have choices is a key to their ability to create self-identity. Whatever the extent of these choices, a person engages in what he terms the *reflexive project of the self*. EPortfolios have a part to play in the project, given their focus on the individual's life achievement and personal reflection, particularly if they allow for choices to be made in purpose, content and format. Cope and Kalantzis [9] argue that maintaining rich and diverse identities is essential, and that presenting self and culture across a range of media are central elements in the new economy. Further, Wenger [14] argues that the purpose of education is to create identities, and representing identities is an outcome of learning in social contexts. One outcome of identity creation is said to be increased self esteem, both individually and collectively. In a recent British report, [15] self esteem is seen as a form of human capital that has wide benefits to society, and the report suggests that government and other institutions have a role to play in shaping the context in which everyone can develop self worth. Digital stories and ePortfolios are potential forms for reflecting on and

presenting the multiple identities of individuals and the collective identities of cultural, social and work groups.

An important aspect of learning is reflection [16, 17]: whether through a structured process of documenting context, purpose, relevance and impact of the selected items, or a freeform *blog* (weblog containing reflection on the day's events, or important themes). This is frequently mentioned, but often difficult to enact, as it places a high cognitive load on individuals. Reflection through blogging, however, enables student voice, and can support learning in a flexible way [18] and therefore is a natural adjunct to ePortfolio development.

2 Method

This paper draws primarily on teachers' research into ePortfolios for forging identity and facilitating learning. Previous work in digital portfolios [1] and the growing interest in digital storytelling, or narrative forms, has informed the research. It is based on five cases, four of which have been taken from contributions made by practitioners to the first ePortfolio Australia conference [19], held in 2004. EPortfolio projects from primary, secondary, vocational and higher education sectors were presented during the conference, offering a lifelong learning perspective often missing from single-sector discussions. The cases included here illustrate a selection of portfolio projects in Australian school settings: a special developmental school for students up to eighteen years of age with intellectual disabilities (A); a private boys' school with primary and secondary sections (B); a government secondary college (C); a specialist senior secondary science and maths college (D), and a group or 'cluster' of rural schools (E). Each project reported a type of action research [20], and the written reflections of the teachers provided most of the data for this paper. The author also visited Schools A, B and D to see their projects in action. As baseline data were not available to conduct an analysis of student learning pre- and post-ePortfolio development, the interpretive judgements and the reflections of the teachers provided most of the data, with some examples of student voice. Sample ePortfolios were made available from three of the settings (A, B and C) so that an external viewpoint was possible.

The framework for analysis comes from the literature above. At first the important structural elements of purpose and audience are considered, and then three processes which are of particular interest to this paper: self-esteem, multiliteracy and reflection. In this paper, self esteem is a social competency reported by teachers and students, not measured by a psychometric test. The process of becoming multiliterate is more academic, and reported by teachers based on their own assessment, not on standardised tests. Reflection on learning is a process that was undertaken in both formally structured and informal ways, and manifest in written, oral or visual forms. These three are all dynamic in that they develop from the ePortfolio approach and feed back to it.

3 Findings

The ePortfolio artifacts were stored in all cases on local school servers, and all cases except one used html as the format for presentation (School B used slide presentation software). Three of the cases (A, D and E) developed structured templates.

3.1 Purpose and Audience

There are many purposes for ePortfolios, from the different points of view of individual students, teachers, parents, educational institutions, governments and employers who form the audiences. Table 1 shows a summary of the creators, purposes and audiences identified in the cases selected for this paper.

Table 1. Creators, Purposes and Audiences for ePortfolios in Selected Schools

	School A	School B	School C	School D	Cluster E
Creators	Students from 5-18, with the help of teachers	Final-year primary students	Middle years secondary students	Senior secondary students	Teachers and students in primary and secondary schools
Major Purposes	Show progress; represent identity	Support primary-secondary transition	Develop technical skills and reflection	Self-knowledge; planning; assessment	Teacher development; student learning
Audiences	Parents and carers	Future teachers	Class teacher	Pastoral teacher; public	Students; peers

At School A, a specialist day school for students aged 5-18 years with a moderate intellectual disability, teachers captured a great deal of video material to show students' progress. Three teachers in the action research team were supported by the IT Manager. They designed two Portfolio templates: one for primary students and another for secondary students. Working closely with their teachers, students gathered examples of authentic learning throughout the year via video, digital photos, scanned images and audio files, often with a focus on learning progression over time. These artifacts were inserted into the template during timetabled lessons, and students and staff collaborated to add reflective comments. A file management system was set up on the school's server to store the ePortfolios as they were being developed. Once the portfolios were completed, parents and carers were invited to visit the school to attend a portfolio presentation and view their child's learning in context. At the end of the school year the students' portfolios were saved on CD ROM and given to the family. A teacher reflected:

Whether it be counting, reading, swimming or drawing, every step of the journey is immensely important to both students and their parents, who can see that their child is learning, in a more concrete way than was previously possible.

The point at which students move from primary to secondary schooling is one of the important manifestations of mobility in school life, and schools are increasingly using portfolios at this time, particularly for the students to represent themselves to future teachers, and to reflect on their schooling so far. In the Year 6 (upper primary) class of the boys' school (B), ePortfolios were intended to support the transition to secondary school, as one student stated:

Portfolios are for Year 7 so the teachers know what we're up to, have a view of what we're going to be like.

The boys used presentation software to create representations of themselves and their achievements. The portfolio structure was chronological according to the school terms, and the teachers gave students broad guidelines as a focus for their collection, with titles such as 'About Me' and a requirement for videos showing progress in music, as well as photos and reflections on their learning. The ePortfolios of these students used narrative forms more than the other cases.

In School D, the focus was on planning for learning, as each student had an Individual Learning Plan negotiated with their tutor. Groups of about twelve students spent forty minutes daily with their Tutor to plan their day, examine learning and career pathways, receive learning support and to work in groups on learning activities. This was seen as an important aspect of personalising learning for each student, and was recorded in the ePortfolio as a basis for future planning discussions between student and tutor.

Cluster E used a set of employability skills developed in Australia as a focus, which assisted in clarifying both the purpose and concept of the ePortfolios. The teachers designed structured templates linking the statewide curriculum structure, employability skills and thinking skills. These were designed to assist students to represent themselves in a narrative form as learners and as individuals.

3.2 Self Esteem

Students frequently noted that they captured evidence of 'firsts' (the first time they had achieved a new skill, or produced a new product) when collecting material for their archive. This gave them an opportunity for later reflection on progress, or celebration of their achievement, indicating an intrinsic value of the artifact. In contrast, some students appeared not to see longer-term value in much of their school work, as a teacher of ICT noted:

A huge problem was that some students had no artifacts to represent their learning. They had thrown out their work, or deleted digital material, after their teachers had marked it. [We] need to encourage students to produce work of value to them rather than the teacher (Teacher, School C).

She felt that they had not been prepared sufficiently for the ePortfolio approach by other teachers. However at least one of the students had artifacts to use:

Today I updated my ePortfolio. It has lots of stuff about schoolwork. It was fun to create (Student, School C).

In School B, students planned their collections of evidence to capture important memories around the school buildings that they were soon to leave, as well as images of assignments and models that they were proud of. They used mobile camera phones provided by the researcher, and from the outset, they were keen to use the 'cool phones', so their motivation was enhanced. In contrast, the students of School A often have greater than average difficulties in presenting themselves to an audience, particularly when leaving the school and seeking employment. One teacher noted that the ePortfolio process appeared to have a positive effect:

It was pleasing to see many of the students' attitudes towards themselves improve, especially within the 13-18 year old age group (Teacher, School A).

3.3 Multiliteracies

As noted above, the concept of multiliteracies includes traditional concepts of literacy while adding aspects such as visual, kinaesthetic and critical literacies. In the case of Cluster E, the teachers decided to create their own ePortfolios before requiring their students to do so. They reported that through designing and thinking through their own ePortfolios, they engaged immediately with the multiliteracies and had to acquire skills they would be asking of their students, such as considering their intended audience, what they wished to communicate, and how to do this most effectively. Many teachers chose to portray themselves as learners in their ePortfolios, with their students as audience. First they shared their presentations with one another for critique and feedback, and later did the same with their students. Previously print-based portfolios had been used throughout cluster schools, but were largely a collection of student work samples and artifacts for display. Now, the coordinator reports, the student as learner is central to the process, leading to a sense of narrative as students represent themselves both as learners and as individuals. She reflected on the development of critical multiliteracies among the students:

Students require the technical skills of recording sound, making hyperlinks, inserting text and video-clips, but the more demanding aspect is skilling students to demonstrate that they are thinking at a deep level. Students are being asked to become more critical about the way they are presenting evidence about their achievements. They are challenged to think closely about what they are going to say, what they wish to communicate and what is the most effective medium in which to do this. They are being asked to demonstrate their own unique skills and attributes and to have a clear notion of their intended audience whether it be parents, peers, teachers or members of the wider community. Importantly, ownership of this process lies with the student (Teacher, Cluster E).

Students in School B found that ready access to the camera function enriched the range of literacies:

It made it a bit better doing the portfolio with the camera because you could actually take pictures instead of just writing about it, if you wanted to.

Figure 1 shows the students transferring material from the mobile phones to their ePortfolios on the school's server (via USB cable or Bluetooth), thus practising a range of technical and social skills. It was observed that dialogue and discussion took place during this process.

A teacher of ICT noted both her surprise and her responsibility in relation to literacy development among her students:

Most of the previous topics we had studied were about software skill development and although I knew many of our students' written language skills were in need of improvement I had no idea just how weak they were until the students wrote their ePortfolios. This revelation led to my redesigning the course to focus more on developing written language skills (Teacher, School C).

An important aspect of becoming multiliterate is presenting to audiences wider than those in the school community. School D reported that students were able to project their identity widely through their ePortfolios, attending a technology expo and international science fairs to present their work as a 'unified representation of their learning'.



Fig. 1. Students transferring artifacts from mobile phone to server (School B)

3.4 Reflection on Learning

It was observed by several teachers that through developing ePortfolios many students recognised themselves as learners and began to reflect on their learning more formally. Similarly, many parents found viewing their child's learning in context extremely powerful. In School A, parents who viewed their students' completed ePortfolios were reported to have been 'overwhelmed' by the work their children put into creating their ePortfolios and by the obvious learning that was captured through the artifacts and reflection.

Teachers have an important role to play in developing a culture and skills of reflection. One teacher found that the students needed more scaffolding than she had initially provided:

It was putting their reflection into words that they found difficult. When I asked some probing questions and provided some examples, the students' writing improved (Teacher, School C).

The artifact in Figure 2 shows the type of reflection common among students. This student combined the historic notion of the 'first' picture with pride in the grade given by the teacher, while also commenting on the technical skill that underpinned the product.

It appears that reflective writing is something that requires a great deal of scaffolding. Other forms of reflection have been attempted, such as recording audio or video comments, and structured questions were provided in some schools. Teachers acknowledged that they too often have difficulty with reflection.

Computer Graphics

I chose my santa for this portfolio because I like it and I got an A for it. It was also my first picture I had to make in that subject.

From doing this work I learnt how to use polygons to make shapes to form pictures.



Fig. 2. Student's reflection on digital artifact (School C)

4 Conclusions and Implications

The small range of projects documented in this paper indicates shared themes and raises many questions for further exploration. With the increasing interest in ePortfolios, schools and education systems require an understanding of the wide range of purposes that ePortfolios can serve, in order to guide further development of appropriate software tools. They also need to recognize and understand the audiences that might value the resulting ePortfolios.

Self esteem is clearly linked with understandings of identity, and the evidence presented here seems to indicate that embarking on the process of developing ePortfolios and, through this process, reflecting on both their learning and their identity, has positive effects on student self-esteem. A teacher observed the lifelong learning skills enabled through ePortfolio development and the role of teachers in negotiating identity with their students, when she wrote:

Identifying skills such as teamwork, listening with empathy and understanding, interacting within the community, and being persistent, requires us to value and acknowledge diverse aspects of students' lives and interests. Students are encouraged to draw upon wider experiences that may well be found outside the school context, to create a richer picture of who they are (Teacher, Cluster E). This would be a useful area for further research, using more specific measures of learners' self esteem.

Where ePortfolios are seen as communicating devices, looking outward, they embrace the multiliteracies. However Pachler [12] warns that although enabling users to create and distribute their own work makes them active participants in the culture creation process, they need to be taught basic and higher order skills, such as informatic, visual and critical media literacies. This has an intrinsic benefit, as well as enabling us all to engage in dialogue with software producers and distributors who devise ePortfolio solutions. How these skills are taught and assessed is a critical area of teachers' and researchers' work in the 21st century.

Reflection is an essential element that differentiates an ePortfolio from a simple repository of artifacts. It can be difficult for many learners to find the time and the skills for serious reflection, and rendered more difficult by many of the assessment portfolio solutions that focus on the administrators' needs for assessment data, rather

than an individual's deep learning. A solution could include interconnected systems: an archive of student work, an assessment management system to document achievement of standards, and an authoring environment where students can construct their own ePortfolios and reflective, digital stories of learning [17]. Because of its apparent links to both increasing self esteem and to actual learning, reflection needs more attention, with research to identify and share the ways in which it has been scaffolded, and recorded.

If ePortfolios are to support personalised learning, student voice and the agency of the learners must be acknowledged. School D's philosophy stated that 'learning is enhanced when students possess deep understanding of their preferred approaches to learning and are able to self-direct and plan their learning'. However the students showed a range of responses to ePortfolios, from great enthusiasm and engagement to disinterest. One teacher wrote perceptively of the need to involve the whole community:

Our challenge for the future is to engage the school community in the process of developing ePortfolios for all students, examining the most useful programs and formats for constructing ePortfolios and discovering the best uses for the ePortfolio. (School D).

Similarly, the ePortfolio Australia conference in 2004 highlighted a need to continue to document and draw together the many ePortfolio projects in all education sectors in Australia. This paper is only able to touch on a small sample of school projects, but the themes that run across them deserve richer exploration. If it is found that ePortfolio development truly supports learning, more learners should be able to be involved. At the same time, designers of ePortfolio systems will benefit from the knowledge of teachers and students in designing flexible systems that enable planning with students, capturing a wide range of artifacts, scaffolding the processes of reflection and feedback, and assessing learning.

References

1. Hartnell-Young, E. and M. Morriss, *Digital Professional Portfolios for Change*. 1999, Arlington Heights, IL: Skylight Training and Publishing.
2. Dewey, J., *How We Think*. 1910, Boston: Heath and Co.
3. Bereiter, C. and M. Scardamalia, *Rethinking Learning*, in *The Handbook of Education and Human Development: New Models of Learning, Teaching and Schooling*, D. Olson, N. Torrance, (Ed.). 1998, Blackwell: Cambridge (MA) p. 485-514.
4. Scardamalia, M. and C. Bereiter, *Schools as Knowledge Building Organizations*, in *Developmental Health and the Wealth of Nations: Social, Biological and Educational Dynamics*, D. Keating and C. Hertzman, Editors. 1999, The Guildford Press: New York. p. 274-289.
5. Hartnell-Young, E., *ePortfolios for knowledge and learning*, in *Handbook of Research on ePortfolios* A. Jafari and C. Kaufmann, Editors. 2006, Idea Publishing: Hershey, PA. p. 124-133.
6. Unsworth, L., *Teaching Multiliteracies across the curriculum*. 2001, Buckingham: Open University Press.
7. New London Group, *A Pedagogy of Multiliteracies: Designing Social Futures*. Harvard Educational Review, 1996. **66**: p. 60-92.

8. Fehring, H., *Critical, analytical and reflective literacy assessment: Reconstructing practice*. Australian Journal of Language and Literacy, 2005. **28** (2): p. 95-113.
9. Cope, B. and M. Kalantzis, eds. *Multiliteracies: Literacy learning and the design of social futures*. 2000, Routledge: London.
10. Hargreaves, D., *Personalising Learning: Next steps in working laterally*. 2004, Specialist Schools Trust: London.
11. Hartnell-Young, E. and F. Vetere. *Lifeblog: A new concept in Mobile Learning?* in *IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE)* 2005. Tokushima, Japan.
12. Pachler, N., *Connecting schools and pupils: to what end?*, in *Issues in Teaching Using ICT*, M. Leask, Editor. 2001, RoutledgeFalmer: London. p. 15-30.
13. Giddens, A., *Modernity and Self-Identity: Self and Society in the Late Modern Age*. 1991, Stanford, CA: Stanford University Press.
14. Wenger, E., *Communities of Practice: Learning, Meaning, Identity*. 1998, Cambridge, UK: Cambridge University Press.
15. McCarthy, H., *The Self-Esteem Society*. 2004, The Cosmetic, Toiletry and Perfumery Association: London.
16. Schon, D.A., *The Reflective Practitioner: how professionals think in action*. 1983, New York: Basic Books.
17. Barrett, H., *Researching Electronic Portfolios and Learner Engagement*. 2005, The REFLECT Initiative Researching Electronic portFolios: Learning, Engagement and Collaboration through Technology.
18. Maag, M. *The Potential use of "Blogs" in Healthcare Professionals' Education*. in *EdMedia World Conference on Educational Multimedia, Hypermedia and Telecommunications*. 2004. Lugano: AACE.
19. ePortfolio Australia. *Proceedings of ePortfolio Australia Conference*. 2004. Melbourne: ePortfolio Australia.
20. Kemmis, S., *Action Research*, in *Issues in Educational Research*, J. Keeses and G. Lakomski, Editors. 1999, Pergamon: Oxford. p. 150-160.

Metacognition in Web-Based Learning Activities

Giuseppe Chiazzeze, Simona Ottaviano, Gianluca Merlo, Antonella Chifari,
Mario Allegra, Luciano Seta, and Giovanni Todaro

Italian National Research Council Institute for Educational Technology,
Via Ugo La Malfa, 153, 90146 Palermo, Italy
{giuseppe.chiazzeze, simona.ottaviano, gianluca.merlo,
antonella.chifari, mario.allegra, luciano.seta,
giovanni.todaro}@itd.cnr.it
<http://www.itd.cnr.it/>

Abstract. One of the main challenges for those working in the field of ICT mediated learning is to develop innovative systems to support the knowledge building process. Did@browser, a system developed in the Institute for Educational Technology, is an attempt to meet this challenge. It is a new educational tool for middle schools which stimulates awareness of students' cognitive and learning strategies by asking metacognitive questions during hypermedia web surfing. This paper presents the features of the system, the pilot study carried out with pupils, the results, and the suggestions for further developments.

1 Introduction

Web-based learning takes place in a very different context from traditional learning. ICT allows the creation of learning environments in which to experiment new tools and educational methodologies to support students during learning.

In traditional educational contexts children's learning problems are strongly influenced by their difficulty in managing their cognitive processes [1].

Besides, if we consider the specific skills required during learning activities with hypermedia, it is important to support the student and enable him to manage his mental activity by understanding, controlling, and manipulating the cognitive processes used.

Moreover, if these processes are not learned and used by students, the *metacognitive miscalibration* persists into adulthood [2]. The technological tools for on-line learning assume a fundamental role in the students' education by helping to reduce the risk that poor ability to monitor their cognitive activities has a negative influence on their future learning processes [3].

Some research describes educational tools that stimulate the student to activate metacognitive skills. In some cases the software provides "constructivist" features to facilitate knowledge construction. [4; 5]. In other research the monitoring of students' activities is supported by written advice [6; 7] or graphic representation of the history of a student's activity [8]. In general, we have found that this software activates metacognitive processes without stimulating the awareness of the student.

The solution proposed by the Institute for Educational Technology is Did@browser, an educational tool for secondary schools, which poses metacognitive questions to students enabling them to monitor their learning processes during web surfing. It is important, in fact, that at this stage of a pupil's educational development he learns to be more independent during learning activities.

This paper presents the system, the testing phase, and the tools used during the research. It concludes with some considerations of the results obtained.

2 The Method

2.1 Tools and Aim of the Pilot Study

The following tools were used during the pilot study:

- The Did@browser System;
- The questionnaire "Surfing behaviour";
- An evaluation test and conceptual maps.

The aim of the pilot study was to test the above tools. In particular, the Did@browser system test assessed whether the strategy of posing metacognitive questions and receiving responses from pupils leads them to exercise executive control of their cognitive strategies during the surfing process and content learning.

The study was based on the analysis of surfing paths and students' answers provided by the Did@browser system, as well as on the surfing behaviour test to investigate the cognitive strategies employed during surfing and the evaluation test and conceptual maps to verify learning. The study allowed us to identify the strengths and weaknesses of each tool and the ways to improve them.

Sample and strategic training. The pilot study of the Did@browser system took place during the year 2004-05 with the collaboration of the Alberigo Gentile secondary school in Palermo. The work sessions were organized in the computer laboratory of the school for a total of 24 hours. The 27 students involved in the research experiment were attending the second class at the middle school. They were divided into two groups, an experimental and a control group.

The subjects of the two groups were divided up, balancing the level of scholastic competences, gender, and familiarity with the PC. In order to verify if and how Did@browser facilitated monitoring of the surfing behaviour of the subjects involved, we assigned the same tasks to both groups, but the experimental group was presented with the independent variable, namely the metacognitive questions.

Each group worked for 6 sessions of 2 hours each during which the students in pairs used the PC for surfing and studying the sites that we structured ad hoc. At the end of the experimental activities we organized two other sessions for discussion with the students and to give them feedback about the experience.

The contents of the activities were previously agreed upon by researchers and the teacher; we created two didactic sites, one focused on the circulation of the blood and the other one on Genetically Modified Organisms (OGM). The topics illustrated in the hypertexts had never been dealt with before by the teacher in the traditional class setting.

Each group spent 3 sessions studying each site. In each session a specific didactic aim was established. The activities in every session were structured in the following way: at the beginning of each session we introduced the task, the time and the activities to be developed; at the end of each topic of study we gave a learning test; at the end of all the activities we administered a self-evaluation questionnaire on surfing.

Table 1. Organization of the sessions with the relative assigned tasks

HYPERTEXT ON CIRCULATION OF THE BLOOD	
Session	Assigned Task
I	Surf the Hypertext freely. Identify the main topics. Find as much information as possible about: the circulation of the blood.
II	Surf the hypertext freely in order to study the information more carefully. Answer the questions in the questionnaire.
III	Study in detail the information about the heart and how it works.
HYPERTEXT ON GMO	
Session	Assigned Task
I	Study in detail the information about Genetically Modified Organisms.
II	Study in detail the information about DNA and how it works, highlighting the most relevant information.
III	Study in detail the information about DNA, how it works and its role in GMO.
CLASSROOM MEETINGS	
Session	Assigned Task
I	Reproduce the conceptual map which describes the links between the different areas of the hypertext regarding the circulation of the blood.
II	Questionnaire on surfing behaviour.

At the end of the experimentation we organized two classroom meetings. The first took place at the end of the session in which students worked with the hypertext about the circulation of the blood. During this meeting, the students were asked to reproduce the conceptual map which describes the links between the different areas of the hypertext regarding the circulation of the blood. In this way we were able to assess whether the students' conceptual map corresponded to the real structure of the site

and whether some fundamental concepts essential for surfing (links, nodes, toolbar, etc.) had been understood.

The second meeting in the classroom was held at the end of the experimentation and its aim was to provide feedback to the students about their activities and receive indications from them about the strengths and weaknesses of the system from a user's point of view.

Table 1 shows the organization of the sessions with the relative assigned tasks.

The Did@browser System. The Did@browser system is a new technological solution developed by the Institute for Educational Technology of the CNR in Palermo in order to support students during surfing and learning on the Net.

The system was based on a client-server architecture and it is composed of the server and two client components: the student and teacher client which are both available in the Internet Explorer.

The system stimulates self-monitoring of the cognitive processes used by students for surfing and learning by posing metacognitive questions during web surfing. Moreover, the teacher can customize a set of questions for each student in order to evaluate their activities and to stimulate their surfing skills, thus facilitating the effective achievement of learning goals.

The set of questions is planned according to the didactic activities that the students must carry out and the web site selected by the teacher. The two sets of questions used during the experimentation are given below.

The first set of metacognitive questions connected to monitoring surfing strategies comprises:

- Why have I clicked on this link?
- What information do I expect to find?
- What other surfing tools were there on the page?
- Why have I selected this link rather than the others on the page?
- Have I already explored the other objects on the page (images, links, text)? If not, do I expect to do so?
- Do I intend to return to this page? Why/Why not?
- Why have you returned to this page?
- Has the image which I've seen helped me to understand better?
- Have I found the information I expected on this page?
- What has interested me most on this page?

The second set of metacognitive questions to motivate students to evaluate the results of their activities and the cognitive strategies employed to learn the content consists of:

- What have I learnt from surfing on this site?
- Which items of the following information are useful for understanding the problem?
- Which reading methods were I adopting to study this page?
- Have I cut and pasted content on this page?
- What topics are dealt with on this site?
- What sequence of actions have I followed to surf the information on the site?
- Has my strategy of surfing enabled me to reach my aim?
- What do I think will help me to surf better next time?

We have experimented with the first set of metacognitive questions. The questions were selected by the researchers and associated to specific nodes of hypertext. The association between the nodes and questions was done so that the students could improve their awareness of strategies employed during surfing. When the student clicked on the link the system showed the question in a window. The student's surfing was interrupted and the system invited him to answer the question.

The system recorded in the log file the information related to the pages visited, the duration of the visits and the student's answers.

The evaluation of the system was finalized to assess whether the metacognitive questions activated during the surfing process had been a useful tool for the self-monitoring of the student.

In particular, the evaluation was based on the analysis of the answers given by students and on the surfing pattern analysis where the following parameters were considered: number of visited links, duration of visits according to the assigned task of the surfing session.

The nodes were classified according to topic areas of the hypertext, identifying the visited nodes and relative frequency of the visited links for each area. This information was evaluated considering the relevance and coherence with reference to the assigned task.

Moreover, we focused our attention on the number, frequency, typology, position of metacognitive questions and relative answers provided by students.

The analysis of the answers indicated that on the whole the use of metacognitive questions was perceived by students as a useful monitoring tool of their activity.

Table 2 shows some answers provided by students to some metacognitive questions.

The data analyzed suggests that this way of posing questions to the students creates two problems: the first is the verbal expression of their cognitive strategies and the second is the functional limitation of the system in associating questions to links. In the first case some students gave content focused answers to question focused on metacognitive processes, while others had problems describing the procedures.

In the second case, the system does not allow us to pose question related to a specific element of the page and this can be misleading for the students. Moreover, the analysis of answers has highlighted the need to distinguish clearly questions focused on learning contents from those focused on the improvement of surfing abilities. In fact, the questions on surfing, apparently unconnected to the context of study and to their task, created irritation or confusion in the subjects when they were unable to understand their relevance.

Besides, if it were possible to distinguish content questions from behavioural questions, we could better customize the questions for students. For example, surfing questions could be administered only to subjects who are less competent in using hypermedia.

We think that it could be useful to introduce meta-cognitive questions in a dynamic rather than, as at present, in a static way, using a real time analysis of log files. So we could pose prompts related to the behaviour of an individual user, customizing in this way the assistance provided by the educational browser.

The quality evaluation of surfing paths was carried out using the visualization software GraphViz [9]. This tool was chosen due to its versatility and the possibility to modify the visualization parameters and for the clarity with which the graphics are represented.

The system is able to graphically visualize the surfing paths of each subject through algorithms which can produce graphics from the bidimensional layout. The graphics produced in this way allow researchers to assess the structure and the relationships between the visited links of a hypertext. Besides, to find temporal data, we analyzed all the log files created during the Did@browser experimentation sessions. In particular, during every surfing session for each user and for each group we calculated the real time spent on the visited pages and the absolute and average time spent on the pages of a specific area of the site (e.g. blood, diseases).

To assess the paths we distinguished the episodic structure of the hypertexts, i.e., the choices that a single user made during surfing, from the emerging structure, or rather all the episodic structures defined by the users' whole surfing activities [10]. This method provides a generalized and representative surfing pattern of a whole user group.

We submitted the data from this pilot study to temporal analyses. Through the software visualization, mentioned above, we were able to observe that the emerging structures described by the users' surfing was relevant to the assigned tasks. In fact, the subjects spent the majority of their time on the nodes of the assigned study area. However, the number of subjects involved was too small to affirm that this result is an effect of the metacognitive prompts. Nevertheless we can conclude that the presence of metacognitive prompts did not disturb the students' surfing or their learning processes.

Table 2. Metacognitive questions and answers

Metacognitive questions	Answers
Why did you come back to the home page?	To visualize other surfing paths.
How did you understand that this is a link?	Because when I moved the cursor of the mouse it became a hand.
Do you intend to return to this page? Why/Why not?	To revise some topics that I don't remember.
Why have you selected this link rather than the others on the page?	Because I'm surfing the site in an orderly way.
What information do you expect to find?	Information about the circulation of blood.
What else can you click on this page?	There are 9 other links.

The Questionnaire "Considerations about surfing the web". The aim of the questionnaire was to evaluate how the subject perceives his surfing behaviour. A comparison between this evaluation and tracking data provides useful suggestions for improving the system.

The questionnaire consisted of 9 questions concerning:

- the self-examination of surfing behaviour;
- the subjective evaluation of the efficacy of content presentation in hypermedia;
- the subject's awareness of strategies used during surfing.

The questionnaire was administered during a meeting of the whole class, at the conclusion of the activities using Did@browser.

The answers permitted us to distinguish some behaviors common to a majority of the students involved. These will now be considered.

The visual components are the most attractive parts of hypermedia. In fact, the images are a means to facilitate comprehension of the contents; they make the presentation more attractive and exciting and thus improve learning.

The students expressed a preference for this means of presentation and their answers were supported by tracking data which showed that pages with images were visited more frequently. Pages containing a lot of links and written text proved to be less attractive and were less visited. Pages containing many technical terms were particularly uninteresting for the students.

From this analysis we can conclude that a good combination of written text and images is essential for improving comprehension and motivating web surfing.

The questionnaire also revealed the principal cognitive learning strategies adopted by the subjects.

First subjects read the whole text carefully before beginning another activity. In case of difficulty in understanding respondents tended to reread the text several times, using didactic tools such as glossaries and dictionaries to increase comprehension.

The analysis of the answers revealed that the subjects recognized that if they had to repeat the same surfing activity they would modify the strategies they had adopted previously. In fact, the students appeared to change their behaviour on the base of experience acquired during each surfing session.

Thus online learning was like a continuous training activity where the subjects tended to improve their strategies by avoiding the actions which were less effective for exploring and learning the topic's contents.

In conclusion, the results of the questionnaire allowed us to study the students' surfing behaviour and suggested some ways to improve the system. However, the questionnaire could be made more effective by selecting the items more carefully and refining the experimental design. In particular, we plan to increase the number of items in some areas in order to analyze the students' behaviour and strategies used during study of the hypertexts more accurately. With regard to the experimental design we think that the questionnaire should be administered before and after the study sessions in order to observe how the use of Did@browser modifies the students' perception of their surfing behaviour.

Assessment tests and conceptual maps. The assessment of learning and of the understanding of knowledge organization was carried out by means of tests and by drawing conceptual maps.

The main objective of the evaluation was to measure the overall performance of each student in relation to the effect the Did@browser tool had on the level of learning. The assessment tests were administered both to the control group and the experimental group at the end of the study using hypermedia.

Conceptual maps were drawn by the students at the end of the last session of surfing the site concerning the circulation of the blood. They were used to analyse and assess whether the metacognitive reflection stimulated by the system had an effect on the autonomous knowledge construction.

The analysis of the learning tests highlighted that both the experimental and control groups had good marks; the number of correct answers exceeded 70%.

The comparison between the conceptual maps of the two groups showed notable differences. It was carried out by evaluating the representation and the interconnections among the nodes of the maps, considering how many strategic points were reproduced and how deep the level of the graphic representation was.

We observed that the experimental group produced maps in greater detail compared to the control group, which frequently omitted some essential nodes. So we think that, as Schwarz et al. [11] indicated in their research, the metacognitive stimulus may have influenced the attention and the mnemonic performance of the subjects involved.

The evaluation of the details of the conceptual maps supports what has been stated so far. On average, the experimental group proposes a higher number of links which correctly describe the hierarchical structure of the areas of the site which they have studied. The results obtained from the learning tests and the production of conceptual maps, lead us to conclude that the strategy of posing questions during surfing did not hinder the learning of contents.

3 Conclusion

The qualitative evaluation of the answers to the metacognitive questions enables us to conclude that it is necessary to improve the effect of these prompts in two ways: firstly by separating clearly the questions concerning the study topics from questions concerning the surfing strategies and secondly by customizing them to suit the surfing behaviour of each student.

Another element to consider in optimizing the system is the use of multimedia prompts related to the contents, in order to indirectly stimulate metacognitive reflection about surfing and learning strategies and making the support provided by the system more enjoyable, as suggested in the literature by Labat [12].

The pilot study of the Did@browser system leads us to conclude that the users' surfing was pertinent to the assigned tasks and that the metacognitive questions did not disturb the students' surfing and learning processes. In fact the students had good marks in the assessment and test, in particular, the subjects in the experimental group improved their comprehension of the intrinsic structure of hypermedia and created more accurate conceptual maps.

Finally, the students considered the system to be a useful self-monitoring tool.

References

1. Ashman, A. F., Conway, R. N. F.: *Cognitive Strategies for Special Education*. Routledge New York (1989).
2. Smith, D. K., Moores, T., Chang, J.: Prepare your Mind for Learning. *Communications of the ACM*, Vol. 48, No. 9. ACM New York (2005) 115-117.
3. Smith, D. K.: Metacognitive Miscalibration and Underachievement in a Computer Literacy Course; Some Preliminary Observations. *Proceedings of the 15th Annual Conference of the International Academy for Information Management* (2000) 1-6.

4. Komis, V., Dimitracopoulou, A., Politis, P.: Contribution à la Création d'un Environnement Informatique de Modélisation. <http://www.modellingspace.net/Documents/Komis%20et%20all%20Poitiers98.pdf> (1998).
5. Zeileger, R.: Implementing a Constructivist Approach to Web Navigation Support. Proceedings of the ED-MEDIA'99 Conference, Eds. Collis, B., Oliver, R., June 19-24, AACE, Seattle, WA. <http://www.gate.cnrs.fr/~zeiliger/artem99.htm> (1999).
6. Coen, P.: A Quoi Pistent les Enfants Quand Ils Écrivent? Analyse des Processus Cognitifs et Métacognitifs en Jeu Dans une Tache d'Écriture Assistée par le Logiciel AutoéVal. http://www.unifr.ch/ipg/these/A_quoi_pistent/introduction/structure_introduction.htm (2001).
7. Lloyd, A.: A Software tool for Supporting the Acquisition of Metacognitive Skills for WebSearching. <http://www.cogs.susx.ac.uk/users/bend/aied2001/lloyd.pdf> (2001).
8. Puntambekar, S., Stylianou, A.: Designing Metacognitive Support for Learning from Hypertext: What Factors Come Into Play?. In: Hoppe, U., Verdejo, F., Kay, J. (eds.): Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies. Workshop proceedings. IOS Press Amsterdam (2003).
9. Gansner, E., Koutsofios, E.: Drawing Graphs with Dot. <http://www.research.att.com/sw/tools/graphviz/dotguide.pdf> (2002).
10. McEneaney, J. E.: A Transactional Theory of Hypertext Structure. Yearbook of The National Reading Conference, Oak Creek, WI: National Reading Miami FL. (2002) 272-284.
11. Schwartz, N., Andersen, C., Hong, N., Howard, B., Mc Gee, S.: The Influence of Metacognitive Skills on Learners' Memory of Information in a Hypermedia Environment. In Journal of Educational Computing Research, Vol. 31, No.1. Baywood Publishing Company (2004) 77-93.
12. Labat, J. M.: Quel Retour d'Informations pour le Tuteur?. Conférence TICE 2002, Lyon (2002) 81-88.

Development of Modern e-Learning Services for Lithuanian Distance Education Network LieDM

Aleksandras Targamadžė and Gytis Cibulskis

Kaunas University of Technology, Faculty of Informatics
Studentu str. 50, LT - 51368 Kaunas, Lithuania
aleksandras.targamadze@ktu.lt, gytis.cibulskis@ktu.lt

Abstract. Development of the Lithuanian Distance Education Network LieDM was started in 1998 by implementing a nation wide videoconference infrastructure. While the number of videoconference - enabled sites was gradually expanding, other e-learning services such as WebCT virtual learning environment, ViPS video lecturing system and CDK course development system were also introduced to the beneficiaries of the Network. The paper provides analysis of the trends in recent developments of learning technologies and defines priorities for further improvement of e-learning services in the LieDM Network.

1 Introduction to LieDM

The Lithuanian Distance Education Network (LieDM) was established in 1998-1999, according to an investment project of the Lithuanian Government and passed through several phases of expansion financed by other national and EU projects. The primary goal of LieDM is to promote establishment of the Information Society in Lithuania by developing and coordinating information and communication technology based system for Higher and Continuing Distance Education (DE)¹.

The main priorities of LieDM Network are:

- Co-operation among education institutions in DE development;
- Development of DE infrastructure based on modern information and telecommunication technologies;
- Virtual Universities and enhancement of e-learning;
- Integration of the Lithuanian DE system into other European and the international DE networks.

Initially, the studies provided by LieDM network were based on videoconference technologies. With the expansion of the network, web-based and other online learning forms were introduced and developed rapidly. Nowadays web-based e-learning prevails in distance education activities in the Network besides the intensive use of videoconferences. LieDM provides Lithuanian citizens with the means to update and improve their qualification and skills. It improves conditions for lifelong learning, and

¹ <http://www.liedm.lt>

expands the variety of education services. The Network provides equal learning opportunities for citizens, wherever they live, whatever their gender, nationality, social status, and physical abilities are.

At the first two infrastructure development phases were financed by Lithuanian government, the LieDM network has expanded up to 16 remote sites connected with IP video-conference technology via LITNET - Lithuanian Academic and Research Computer Network. During the implementation of the Phare2000 Regional Development Programme, several DE centres and classrooms were established at Vocational training centres and secondary schools in Klaipeda-Taurage, Mariampole and Utena regions (Fig. 1).



Fig. 1. Infrastructure of LieDM Network

Since 2001, the coordination of LieDM activities and further development of the network were committed to the new programme called “Information Technologies for Science and Higher Education (ITMiS)” initiated by the Lithuanian Ministry of Education and Science. Besides LieDM, there are two other sub-programmes called “Lithuanian Higher Education and Research Information System (LieMSIS)”² and “Lithuanian Academic Libraries Network (LABT)”³ coordinated and financed by the ITMiS programme. Initially dedicated to Higher Education, ITMiS is already opening its’ recourses and services to other target groups of Lithuanian Education System. All three “sister” projects are supposed to coordinate their activities and to complement each other by employment of existing resources, in order to form the Lithuanian science and education information environment intended for the following issues [1]:

² <http://www.liemsis.lt>

³ <http://www.labt.lt>

- Science and education data accumulation and employment in the activities performed by different institutions and by decision makers;
- Representation of Lithuanian science and studies in the global computer networks;
- Assistance to scientists, teachers and students in accessing necessary information;
- The use of information technologies in education and training.

Under the coordination of the Ministry of Education and Science and the board of ITMiS programme, several national projects were initiated and received the financial support from EU Social and Regional Development Funds. These projects are aimed at further development of infrastructure and human resources in the following areas: distance education and e-learning, e-libraries and e-publishing, e-portfolios and e-administration. All together, these projects are supposed to create an integrated e-service platform which could become an important prerequisite for the establishment of a Lithuanian Virtual University. By the term “virtual university” we refer to the infrastructure that would facilitate students in their learning process, would provide related support services to them in order to complete their degree program partially or totally online. This infrastructure would facilitate participation of faculty members in the learning process by providing necessary resources for teaching and performing online research activities [2].

2 e-Learning Services in LieDM Network

The LieDM network provides the set of services to its users, which enable development and delivery of DE courses and programmes. Most of these services are presented and could be accessed via the LieDM portal which also serves as the central access point to all information about LieDM network, its members, organisation of study process, schedules of videoconferencing events, as well as recourse reservation and management. Services used in LieDM network could be categorised into learning delivery services, learning content development services and support services (Fig. 2).

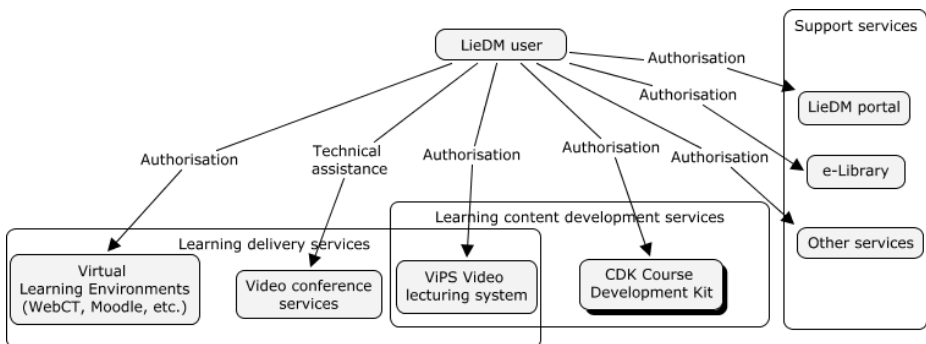


Fig. 2. Services in LieDM Network

Learning delivery services consist of virtual learning environments (WebCT, Moodle, Learning Space, etc.), video-conference services and Video Lecturing System (ViPS) which enables simultaneous streaming of live lectures over the internet. The same ViPS system can be treated as learning content development service, as it allows recording and editing video lectures. The Course Development Kit (CDK) is used for e-learning content developing and publishing on the internet. Support services consist of the LieDM portal, e-Library and other services, such as e-mail, database of study modules, etc.

2.1 Video-Conferences as Learning Delivery

Videoconferences as a synchronous communication tool provide great interaction possibilities and can simulate a face to face communication process. Current video-conference infrastructure of LieDM supports up to 96 simultaneously connected sites and allows delivery of high quality interactive lectures to multiple videoconference classrooms using two way audio and video, as well as dual screen video+data possibilities. S. Whittaker and B. O'Connail show in their work show that video supports visible behaviour and supplies important non-verbal information. It also provides visible environmental information, specifically on the availability of other people, which in turn facilitates initiation of spontaneous interactions. Finally, video provides dynamic visual information about objects and events in the shared environment [3, 4].

While using videoconference technology, teachers can adapt most of the traditional teaching methods they use in the classroom. Hence, they can be converted from face-to-face to distance teachers quite rapidly. Video-conferencing is the closest distance education has got to face-to-face teaching and it is one method that minimizes in distance students feelings of frustration regarding being taught by means of a method that is a rather inferior, makeshift alternative to face-to-face teaching [4].

Of course, there are some disadvantages, such as complicated eye contact and restricted body language, but it is possible to cope with that having in mind that it gives an opportunity to people who wouldn't otherwise have access to education. Although the videoconferences are mostly used for continuing and higher education there are many cases when the pupils and teachers from secondary schools are involved in different events. As availability of a broadband network is still very limited and video-conference equipment is available just in few schools, the infrastructure of the LieDM network is hosting this kind of events.

Administration and support of the LieDM video-conference network, distributed among different organisations, brings many challenges. Scheduling videoconferences and avoiding conflict of recourses, managing participants and evaluating the quality, accounting of the resources utilisation, writing the reports are just a few of them. The LieDM portal was developed to solve all these administrative tasks. The main purpose of the LieDM portal is management of online resources and events. For this purpose, a catalogue of courses and facilities providing the participants with self-registration possibilities to public events were developed. The portal is under constant development and it is supposed to become an advanced virtual community platform for all LieDM users.

2.2 ViPS Video Lecturing System

Though, being a powerful tool in synchronous communication, videoconferences also bring some restrictions in terms of time, place and pace of DE. The learners have to attend learning events at certain time and to come to the nearest videoconference classroom. Growing demand and interest in synchronous DE forces to look for alternative solutions, in order to overcome these restrictions. For this purpose a web-based environment integrating a solution based on video streaming was developed at Kaunas University of Technology (KTU) Distance Education Centre. In the framework of the Eureka project “TESTVIL – Tele-Education Software for Interactive Video Lecturing”, supported by the Lithuanian State Science and Studies Foundation, the ViPS system was developed. ViPS enables delivery of synchronous interactive video lectures to any internet user, as well as live recording and further editing of the lectures. The following possibilities are available for different users of ViPS system:

For the teacher/assistant:

- Exchanging text messages with participants of the broadcast;
- Sending short “yes/no” questions and monitoring answers;
- Synchronous browsing of uploaded slides and bookmarks;
- Sending multiple choice tests and monitoring answers;
- Answering students questions.

For the students/participants:

- Following information presented by the teacher (slides and web links sent by the teacher are opened automatically);
- Answering short “yes/no” questions;
- Taking multiple choice tests or surveys;
- Asking questions.

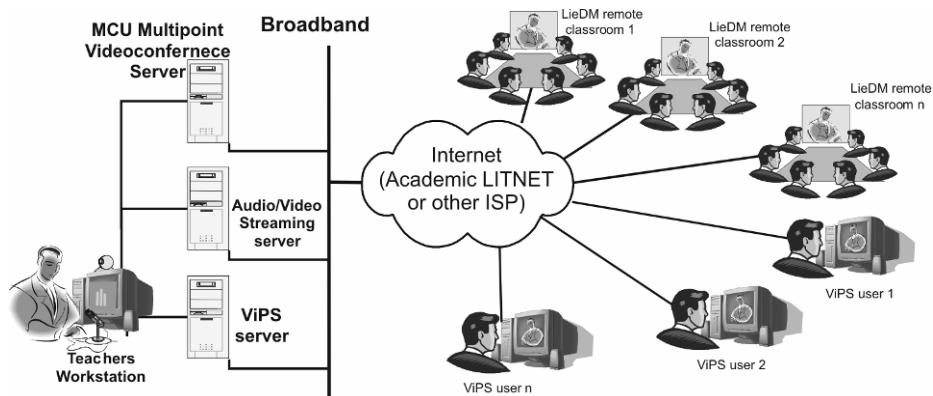


Fig. 3. Extending LieDM Network by Implementation of ViPS System

The main advantage of the ViPS system is its elegant integration to the videoconference solution in such a way, that delivering a video-conference lecture, the teacher can simultaneously broadcast it to all internet users using ViPS (Fig. 3).

2.3 Virtual Learning Environments

There is a variety of Virtual Learning Environments (VLE) used in Network partner institutions, but only the WebCT Learning Management System (LMS) is currently provided as centrally managed service to all LieDM members. WebCT provides a wide range of functions to teachers and students enabling different online learning activities. The majority of activities in virtual learning environments are based on asynchronous communication and can be more flexible for students and teachers in terms of time, place and pace. Usually teachers have advanced activity tracing possibilities for monitoring of students' behaviour and assessment. Students performing online activities not only read the material, perform tests and assignments, but also communicate with teachers and their peers (Fig. 4.).

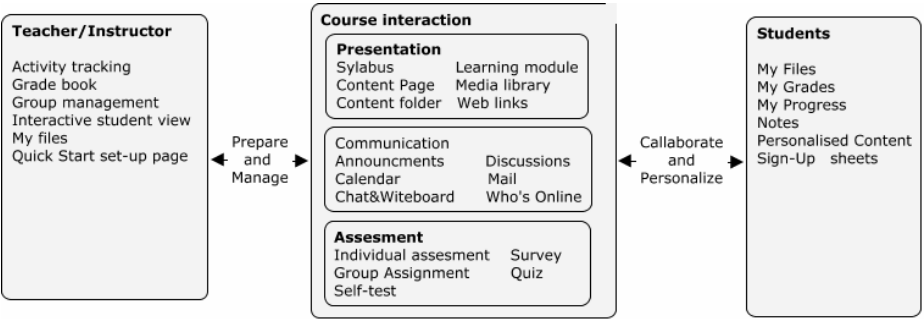


Fig. 4. Services and functions of WebCT LMS

Although registration of courses and students to WebCT is managed by LieDM's coordination centre, there is no single authorisation for WebCT and for the LieDM portal is not yet implemented.

2.4 Authoring of e-Learning Materials

WebCT and other VLEs are providing very good solutions for learning delivery, but typically they have quite pure tools for content development. Authoring tools are necessary for creating new material and integrating existing content which is often heterogeneous and only semi-structured [5]. CDK is an innovative tool developed by KTU Distance Education Centre. It is intensively used by teachers from many institutions for easy authoring of e-learning content. Material is formatted with a visual Web-based editor using a predefined set of styles. It is stored in XML (Extensible Markup Language) based documents. A specially developed Course Markup Language in combination with a great variety of XSL (Extensible Stylesheet Language) transformation templates enable high flexibility and good reusability of already developed materials. Transformation templates can represent different layout or design view of exported material or even can define the conversion to another file format (e.g. there are templates for transformation to .pdf). The material developed using CDK can be exported either to the teachers' computer or directly to the server after selection of the transformation template.

3 Further Development of LieDM Network

For further development of LieDM infrastructure, the main attention has to be paid to the international initiatives of standardisation of Learning Technologies. There are many organisations and research communities working in the development specification and standardisation of learning technologies.

The first standardisation initiatives were dealing with data interchange related to content and its management. In this context SCORM (Sharable Content Object Reference Model), LOM (Learning Object Metadata) and some other standards and specifications were developed. The best illustration of benefits of application of those standards is provided by SCORM “ilities” - reusability, accessibility, durability and interoperability [8].

The next phase of standardisation is more concerned with runtime issues, interactions, activities and services. The trend towards transactional approaches is driven by the growing acceptance of service oriented architectures and the development of Web service standards based on XML. IMS Learning Design specifications have to be mentioned as one of the examples of this trend. They try to standardise pedagogical and learning activity processes [9].

Fully acknowledging how standards are important in deploying e-learning infrastructure. We have to admit that standards as such are often ‘borne’ from the results of innovations and early market adoption, so they should not stop from innovating and implementing new methods and technologies even if those are going beyond standards.

3.1 Integration of LieDM Services

Another trend that can be observed in recent years is the shift from monolithic applications towards service oriented approaches (SOA), where flexible granular functional components expose services accessible to applications via loosely coupled standards-based interfaces. It is true in many organisations that a number of services used in different domains (e.g. library, e-learning, administration) are not specific to that domain, but may be common to more than one domain. There is also a clear recognition that data and resources created and managed by one domain are used by or could be of use to other domains. SOA has the potential to support a “create once, use many” best practice philosophy by exposing the services [7].

Besides services that are used in the LieDM network, there are several services that have to be developed further on as internal service of the LieDM portal:

- Course catalogue;
- Recourse management tools;
- Scheduling and subscription services;
- Communication tools;
- Learners and teachers portfolios;
- Evaluation tools.

External services should not be limited to the ones that are mentioned, and architecture has to be open for integrating other services, such as the e-library system of LABT, e-administration and e-portfolios of LieMSIS, as well as VLEs used by

other academic institutions. The important task is to unify user authentication and to enable single sign in through the LieDM portal. For this purpose a central LDAP users directory has to be established and shared with other centrally provided services (Fig. 5).

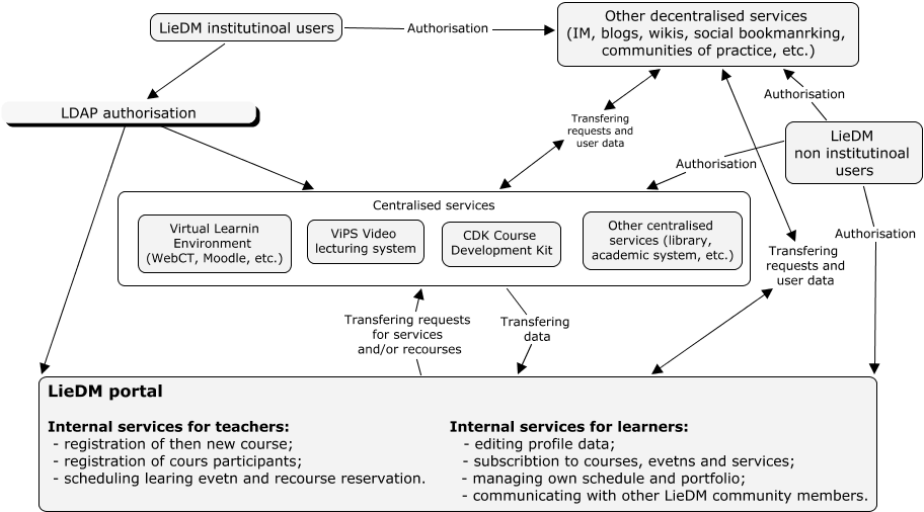


Fig. 5. Integration of services to LieDM portal

Still, taking the users' perspective, we cannot limit ourselves only with centrally provided services but should allow users to establish the links with personal services they use on the third party platforms as well. Here such decentralised services as Internet Messaging (IM), Blogs, Wikis, social bookmarking, virtual communities of practice and other more social communication oriented services are considered (Fig. 5).

3.2 ViPS. Developing Tools for Interaction and Harvesting of Metadata

During several years of intensive use of the ViPS system, it was realised that the main treasure of the system is an archive of recorded events. Many events and lectures were delivered as open for public, and now, there are more than thousand records available for free to learners and all internet users. This is a huge learning recourse which could be even more useful, if it would be accommodated with an appropriate metadata for easier retrieval and reusing. Widely acknowledged problem with metadata is that it is very difficult to convince authors of the learning recourses to fill in the metadata fields. That is why more and more recognition is getting unstructured metadata harvested automatically from the learning recourse, as well as users' generated metadata. It is typical on the internet to allow to users to write their own comments on various digital artefacts: articles, news, pictures, etc. These comments can provide useful information while searching for an artefact related to a specific topic, even if there is not any other metadata entered by the author. Another very important aspect

of such comments is the possibility to facilitate asynchronous communication and social interaction among users of the system.

It is clear that synchronous communication gives the best interaction possibilities but has limitations regarding flexibility of time and pace. Asynchronous distance learning gives most flexibility for an autonomous learner but has limited interactivity. In order to combine the advantages of both synchronous and asynchronous learning and to overcome time and space disadvantages, we need to develop the methods to simulate live interaction for learners who watch recorded events via the ViPS system. It is possible to develop a technique for recording interaction events (e.g. test or voting) during the live session and to reproduce it in the record review mode. Moreover, it would also be possible to provide such pseudo synchronous interaction with feedback by visualising an accumulated summary of all responses. The review mode could also be enriched with the previous learners' comments that might be synchronised with the presentation materials. In this way, those participants would get even more information than the ones who participated in live events, as there would be more user data accumulated afterwards. Of course, in recorded events, there is no possibility to provide instant spontaneous reaction to learners' questions, but it is still possible to set-up the automatic e-mailing of all posted questions to the instructor, so s/he could react in a timely manner.

3.3 CDK. Standardisation and Sharing of Learning Objects

The plan of CDK developers' has ambitions to further develop this tool and to evolve it to an advanced service for finding, reusing, adapting and sharing of existing material, as well as controlling sources and versions of e-learning material. The tool partially supports the LOM standard already and it is planned to develop it further for implementation of other learning technology standards such as IMS and SCORM. Compatibility with these standards would facilitate better integration of CDK with other repositories, tools and systems. In the framework of the EU's FP6 Information Society Technologies project iCamp⁴ KTU, as a partner institution, is committed to open up recourses of CDK and ViPS repositories by implementing Simple Query Interface (SQI) [12, 13] which would enable federation of repositories, brokerage and reusability of learning objects at an international level. Reusability of existing learning recourses is even more important at the national level and secondary schools are also moving into this direction. There are separate programmes for developing e-learning initiated for the secondary education system. Nevertheless, there are many teachers of secondary schools who are using CDK for development of e-learning content. Thus, it will be very important to set up links among repositories and e-learning environments developed for secondary and higher education.

4 Conclusions

The Lithuanian Distance Education Network together with other national programs successfully developed infrastructure and services that enable educational institutions

⁴ <http://www.iCamp-project.org>

to transform their academic and research activities into virtual online environments. De-facto it realises the main prerequisites of a virtual university.

The following development trends can be indicated as priority areas:

- to further develop the LieDM portal aiming at an open and service oriented architecture and integration of third parties' services;
- to develop an educational repository of learning objects and enriching it with content authoring tools and learning management systems aiming at compatibility with learning technology standards and maximum reusability;
- to measure and employ advantages of flexible, interactive, synchronous, and asynchronous communication tools in possible combinations and to develop integrated solutions with mixed possibilities;
- to initiate cross institutional collaboration at all educational levels in order to maximize employment of developed technology and infrastructure as well as reusability of existing learning resources.

References

1. Information Technologies for Science and Higher Education. Program for the period of 2001-2006. Lithuanian Ministry for Education and Science (2001). Retrieved from <http://www.itmis.lt> on April 4, 2006
2. Aoki, K., Pogroszewski, D.: Virtual University Reference Model: A Guide to Delivering Education and Support Services to the Distance Learner. In: Online Journal of Distance Learning Administration. Vol. 1, n. 3. State University of West Georgia, Distance Education Center. (1998) Retrieved at <http://pebbles.westga.edu/~distance/aoki13.html> on April 12, 2006
3. Whittaker, S., O'Conaill, B.: The role of vision in face-to-face and mediated communication. In: Finn, K., Sellen, A., Wilbur, S. (eds.): Video-mediated communication. Mahwah, NJ: Lawrence Erlbaum Associates, Inc. (1997) 23-49
4. Carvalho, C.: Modernizing and Globalizing the Learning Environment: Video-Conferencing in Education. In: Distance Education in Small States, 2000 Conference Proceedings. The University of the West Indies/The Commonwealth of Learning. (2000) 299-309
5. Süß, Ch., Freitag, B., Brössler, P.: Metamodeling for Web-Based Teachware Management. In: Chen, P.P., Embley, D.W., Kouloumdjian, J., Liddle, S.W., Roddick, J.F. (eds.): Proc. Intl. WWWCM'99 Workshop on the World-Wide Web and Conceptual Modeling in conjunction with ER'99, Nov. 15-18 1999, Paris, France. Lecture Notes in Computer Science Vol. 1727, Springer-Verlag, Berlin (1999) 360-377
6. Blinco, K., Mason, J., McLean, N., Wilson, S.: Trends and Issues in E-learning Infrastructure Development. A White Paper for Alt-I-lab 2004 (Advanced Learning Technology Interoperability). Prepared on behalf of DEST (Australia) and JISC-CETIS (UK). (2004)
7. Wilson, S., Blinco, K., Rehak, D.: Service-Oriented Frameworks: Modelling the infrastructure for the next generation of e-Learning Systems. Prepared on behalf of DEST (Australia), JISC-CETIS (UK) and Industry Canada. (2004). Retrieved from: http://www.jisc.ac.uk/uploaded_documents/AltI-labServiceOrientedFrameworks.pdf on April 12, 2006

8. Dodds, P. (ed.): Advanced Distributed Learning Sharable Content Object Reference Model. The SCORM Overview. Advanced Distributed Learning. (2001)
9. Littlejohn, A., Buckingham Shum, S. (eds.): Reusing Online Resources: A Sustainable Approach to eLearning (Special Issue) *Journal of Interactive Media in Education*, Vol 1. (2003). ISSN:1365-893X
10. Duval, E.: Learning Technology Standardization: Making Sense of it All. In: *ComSIS*, 1(1): (2004) 33–43
11. Marshall, S.: E-learning standards: Open enablers of learning or compliance strait jackets?. In: Atkinson, R., McBeath, C., Jonas-Dwyer, D., Phillips, R. (eds.): *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference*. Perth (2004) 596-605
12. Aguirre, S., Brantner, S., Huber, G., Markus, S., Miklós, Z., Mozo, A., Olmedilla, D., Salvachua, J., Simon, B., Sobernig, S., Zillinger, T.: Corner Stones of Semantic Interoperability Demonstrated in a Smart Space for Learning. In: Decker, S., Stuckenschmidt, H. (eds.): *Poster Proceedings of the European Semantic Web Conference*, Heraklion, Greece (2005)
13. Simon, B., Massart, D., van Assche, F., Ternier, S., Duval, E., Brantner, S., Olmedilla, D., Miklós, Z.: A Simple Query Interface for Interoperable Learning Repositories. In: Olmedilla, D., Saito, N., Simon, B. (eds.): *Proceedings of the 1st International Workshop on Interoperability of Web-based Educational Systems*, Chiba, Japan (2005).

Localization and Internationalization of Web-Based Learning Environment

Tatjana Jevsikova

Institute of Mathematics and Informatics, Akademijos str. 4,
LT-08663 Vilnius, Lithuania
tatjanaj@ktl.mii.lt

Abstract. Internationalization and localization of web-based learning environments is problematic for two reasons: 1) communication of software in two environments: browser at the client side and virtual learning environment at server side, and 2) educational necessity for full compatibility of language and culture elements between learning content and the enclosing web-based learning environment. The localization process of open source virtual learning environment and web browsers is discussed. A list of the main unresolved internationalization issues observed in an open source web-based learning environment is presented. The importance of internationalization of original software is discussed as a means for cost saving of localization and for enhancement of product quality towards achieving localized software quality such that it looks and feels as if it would have been made for the target language and culture.

1 Introduction

General requirements to localized software are exactly expressed by the statement that *localized software must look and feel as if it would have been made for the target language and culture* which is known as localizers' folklore [8]. This is especially important for educational software as it indirectly influences school student's language habits and culture orientation.

According to licensing and distribution policy, software may be divided into two main parts: proprietary and open source. Open source software can be obtained free of charge; the source code is freely available and can be modified to fit one's needs; the localization negotiations are less complicated. However the level of internationalization of open source software usually is lower than that of proprietary software. The reason is in the goals of software production. Proprietary software is produced for the market. In order to sell it across the border it is necessary to localize it and to do localization easier by means of internationalization. Open source software is produced by volunteers without profit considerations and sometimes without sufficient knowledge about the subject of localization. Thus localization of such software in many cases is problematic. So, we restrict our discussion in this paper to open source software.

The problem can be solved at software production time, making it language- and culture-neutral and suitable for localization (this process is called internationalization), or at localization time, modifying the original code [9]. More accurate and less expensive is internationalization simply by the fact that work may be done once and forever in the original product versus to solve the problem for every localization and migrating solution from version to version. So, we emphasize internationalization here to identify problems and make them known to this audience of educators in the hope that it will reach educational open source software developers.

Teaching and learning by internet becomes more and more popular. Web-based learning environment consists of two main communicating parts: client side and server side. Virtual learning environments (VLEs), such as Moodle, ATutor, and Claroline, are several of the most popular components on the server side. Access to the server is provided by a web browser at the client side. Learning content is placed inside the VLE. The server software with its own settings and a web browser form the environment of the VLE. The operating system is the ultimate environment of all those. Thus we have hierarchy of environments (Fig. 1). Interoperability between those parts is a source of additional problems for localization.

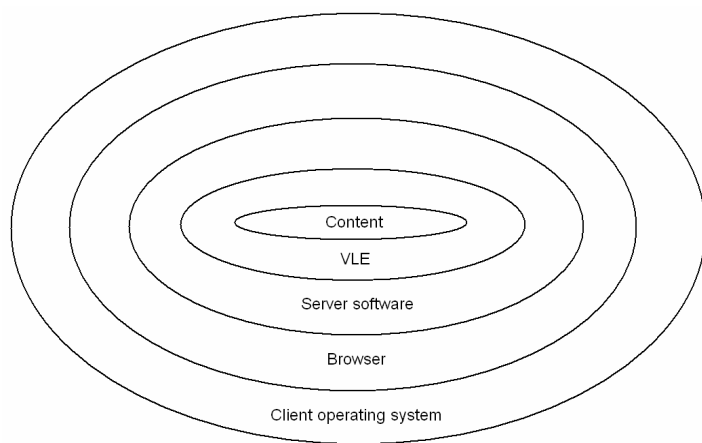


Fig. 1. Hierarchy of environments

The closest environment for learning content is the VLE. It must not contradict the learning content. E.g. if the same string "3,141" in original or properly prepared (localized) content denotes the real number π (decimal fraction), but in the VLE interface denotes an integer number (one thousand one hundred and forty one), then it is a serious conflict.

General localization problems are discussed and some solutions of the problem can be found in [2, 7]. Here we concentrate on specific problems concerned with enhanced requirements concerning the quality of web-based educational software.

2 Server-Side Localization and Internationalization: Virtual Learning Environments

During last decades a lot of open source and proprietary VLEs have been created to support the newest ideas of e-learning. Open source VLEs gain more and more popularity within learning communities because of the reasons mentioned above.

We select three of the most popular open source VLEs for our analysis: Moodle, ATutor, and Claroline. The main selection criterion is a number of languages they are localized to. All of the selected VLEs are translated into many languages. The VLE Moodle (developed in Australia) is being translated to 67 languages, ATutor (developed in Canada) – to 56 languages and Claroline (developed in Belgium) – to 33 languages. Some results (e.g. internationalization issues discussed above) are suitable for other VLEs either.

One of the localization work issues is translation of the strings and documentation. However, full 100% translation has been reached only for a small number of languages (Fig. 2). For most of the other languages translation is still in progress.

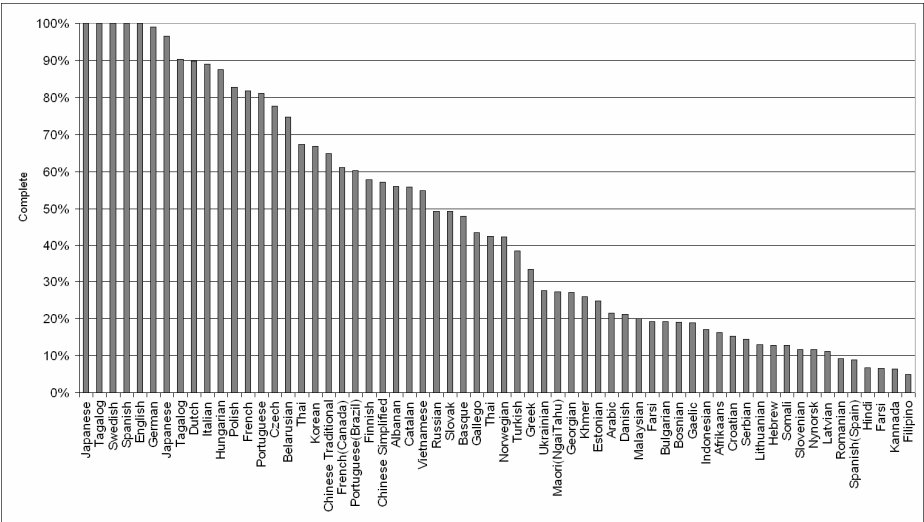


Fig. 2. VLE Moodle 1.5.3 translation progress (March, 2006)

The same situation is common with many open source programs, especially the larger ones. It can be explained that translation of the last strings is more time consuming than translation of the same number of the first strings [1].

Packaged distribution is common for all localizations Packages contain resource files for all languages. Installation programs are usually not localized or partly localized. In the general case such localizations are considered not complete. However, this is acceptable for programs running on servers: program installation on a server is usually the privilege of the server’s administrator (not a teacher or student), whose knowledge of technical English is necessary.

Some documentation files are written for server administrators or program developers. They are translated to some languages, to other they are left untranslated (e. g. Moodle in German), but supplemented with a preamble written in the target language and presenting a summary of the files contents and explaining the reason, why it is left in English. This should not be confused with help files. Those are written for users and must be translated and adapted.

Localization is not the translation of strings only, but also adaptation of the product and its components to the particular cultural environment – achieving full fitness to the locale of the target language (character encoding, number and date/time formats, colors, examples, etc.). Foresight of such adaptations and possibilities to accomplish them during localization should be reasonably applied during development of the original product, i.e., to internationalize it [2].

The values of some frequently used items are included in the settings of the operating system (character encoding, decimal separator, thousand separator, date and time formats, currency format, etc). An application may detect and use them. Otherwise their setting must be defined during localization. In this case they must be included in a localizable part of program.

Lack of some internationalization setting may be considered as internationalization defects or errors (bugs). Here we present the list of internationalization errors we have found analyzing VLE originals and their localizations into European languages using Latin, Cyrillic and Greek alphabets. The list of issues based on general local issues, defined in international standard [5], and VLE-specific internationalization issues, observed during VLEs analysis.

Character encoding. To represent a language's characters VLEs should use an encoding, suitable for that language. Unicode (UTF-8) has an advantage compared to 8-bit encoding as it can be used to represent multilingual characters (for example, for implementing courses for learning of foreign languages). In our examples, Moodle and ATutor use UTF-8, Claroline currently does not.

Letters in login name. Most of the VLEs do not support usage of international characters in user (login) name (one of the components of authorization data): only underscores, numbers, and letters from the basic Latin alphabet are usually accepted. Some VLEs (e.g., ATutor) use the login name not only for internal identification but also for addressing the user in the system (the user name becomes visible to the user himself and other users, it is used during communications between user and system, as well as between users). So, it is natural that user should be allowed to choose a name, composed from his/her native language alphabet letters (not only ASCII).

Letters in password. Password is another component of authorization data, and usually is composed from letters and digits. It is natural, that all letters of the alphabet may be freely used in a password. There are many programs which restrict the set of letters to the ASCII alphabet. It is unnatural for languages whose alphabets do not coincide with the ASCII alphabet to restrict using some or all own letters and offer to use foreign letters instead. Restriction of the character set available for a password reduces its security.

Letters in personal name. Virtually every VLE has user profile area, where personal data are collected and stored. It is natural that all correct characters of a person's first and last name (surname) should be accepted (a user shouldn't change or misspell

his/her real name to register in the VLE). Our analysis of the selected VLEs has shown that usage of national characters depends on server's settings: if they are correct for the locale (cultural environment), then all the characters of that locale can be used.

Order of first name and last name. The order and punctuation of a person's first and second names is different in different cultures (e.g. "First Last", "Last First", "Last, First"). In VLEs such pairs of names usually appear in greeting when starting the session, as well as the external identifier of the user in the system, visible to other users. So, the order of name and surname should be included in the localizable part of source either as a separate parameter indicating the order, or via including parameters that have different names in other strings. For example, the string "%s %s has been successfully registered to the course", where the first parameter %s stands for the first name, and the second parameter %s stands for the last name, is incorrect from internationalization point of view, as a localizer can not choose the correct order.

Matching of plural and singular forms. VLEs (and other applications) usually have dialogs where next to a number (either entered by the user, or presented by the VLE) an appropriate object name is shown. However, applications rarely choose the correct form of object name according to the number, for example, "1 object", "2 objects", "3 objects". In English two forms of noun are used (plural and singular); in other languages three or more forms are used (in Lithuanian, Polish, Russian and other languages there are three forms: one singular and two plural). So applications should include the number and separate strings for all plural and singular forms in localizable sources and dynamically match them.

Grammatical name forms. In inflective languages (Lithuanian, Finnish, Polish, etc.) names in dialog windows may appear in various cases. It is not reasonable to require that generators of all forms for arbitrary names of all languages be included into original software. However, a placeholder for such generator should be foreseen. Vocative case is practically required for names in all logins to VLE. E. g, English 'Hello, Jonas' will be 'Sveikas, Jonai' in Lithuanian. The name may be first name, second name or a combination of both as well as the user name.

Parameterized strings. A lot of problems during localization usually cause sentences or phrases, composed of several separate strings, as well as some parameters used inside the strings. Such a parameterization can be suitable for one language, but totally unacceptable for other languages, as strings order, word forms, usage of capital letters is different among the languages and cultures. The usage of parameter should be commented via localization comments, so that localizer could use it in correct place.

Date format. There are short and long date representation formats, which are different in different countries, e.g. mm/dd/yy, yyyy.mm.dd, yyyy-mm-dd for short date formats. Date presentation in VLE usually depends on server settings and VLE settings. Some VLEs cannot properly show date in the correct format, even if the server's date settings are correct for the particular cultural environment. There should be a possibility either to use formats according to international standard [4], or specify in the localizable source desirable default date format, date components (year, month, day) separators and the order of components.

Time format. There are 12 and 24 hours time formats and various separators for hour and minutes (e.g. full stop, colon). For example, in the UK the 12 hours format is used

(e.g., 11:30 AM, 5:40 PM), in many other European countries a 24 hours format is used (e.g., 19:45). Time presentation in VLE usually depends on server settings and VLE settings. Some VLEs cannot properly show time in the correct format, even if the server's time format settings are correct for the particular locale. There should be a possibility either to use formats according to the international standard [4], or to specify in the localizable source a desirable default time format and separators.

First day of the week. The first day of the week differs among countries, e.g. in USA it is Sunday, while in Lithuania, Poland, Russia and many other countries it is Monday. Almost all VLEs have a calendar component for tracking learning events, where the right beginning of the week must be shown by default.

Decimal separator. The international standard [3] defines two decimal separators: comma and point (full stop). In USA the point is used as decimal separator, in most European countries the comma. An incorrect decimal separator in a particular language may be confused with the thousand separator and lead to wrong understanding of number values. There are programs which detect it from operating system settings. For others it must be chosen during localization. Unfortunately, there are programs with hardcoded decimal separator as point and this requires considerable amount of extra programmer's work during localization. In VLEs, decimal separators usually appear in counters of downloaded/uploaded material, assignment scores, or file size.

Thousand separator. In USA the thousand separator is represented as point, in most European countries as comma or blank space. An incorrect thousand separator may be confused with a decimal separator and lead to wrong understanding of numeric values. There are programs which detect it from operating system settings. For others it must be chosen during localization. Unfortunately there are programs with hardcoded thousand separator as comma and this requires considerable amount of extra programmer's work during localization. In VLEs, the problem usually appears in content's files sizes, scores, etc.

Telephone number format. The telephone number format, accepted in different countries, can be different. In VLE it appears mostly in user profile data. If the telephone number is composed of several separate fields, there should be a possibility to change fields order and their number. Another way to avoid format incompatibilities is to use only one field.

Dialog fields and controls order. VLEs usually use separate fields or controls for entering culture-sensitive data, e.g., an address dialog may have a "state" field unsuitable for many countries in the user profile, or the agenda dialog may have an order of date fields (e.g., day, month, and year) that do not fit some locales. Ideally, there should be an option to change the order through localizable source (it shouldn't be hardcoded).

Colors and graphics. Colors and graphics are widely used in VLEs. It is known that the same color can have different meaning in different locales (e.g., white in Europe and Japan). Sometimes, text is used within graphics. That's why there should be a possibility to modify some graphics elements (they should be included in localizable source) and to change color scheme (this is usually done by means of CSS).

An evaluation presented below (Table 1) was based on European locales. We do not include here hieroglyphic, right-to-left direction languages. If we find an incompatibility with one of these locales on a particular internationalization issue, we

can state, that the issue is not internationalized. Plus (+) means that the issue is internationalized (adapted for its localization), and minus (-) means that the issue is not fully internationalized. However, server software (Web server software being used, PHP environment, database software, etc.) and its settings could make some impact on evaluation of server-side localization.

Table 1. An implementation of internationalization issues

Internationalization issue	Moodle	ATutor	Claroline
Character encoding	+	+	+
National characters in login name	-	-	+
National characters in password	-	-	+
National characters in user's first and last names	+	+	+
Order of user's first name and last name	+	+	-
Matching of plural and singular forms	-	-	-
Grammatical name forms	-	-	-
Composition of several strings	-	-	-
Date format	-	+	+
Time format	+	+	-
First day of the week	-	-	+
Decimal separator	-	-	-
Thousands separator	-	-	-
Telephone number format	+	+	+
Dialog fields (controls) order	-	-	-
Color scheme and graphics	+	+	+

3 Client-Side Localization and Internationalization: Web Browsers

The most popular open source internet browsers have come from Mozilla: Mozilla itself and its derivatives (SeaMonkey and Mozilla Firefox). Mozilla originated from the proprietary browser Netscape. It has its distinct resource file structure. Each localization is implemented as a separate distribution package. Thus the installation program should be localized ensuring full localization of all packages. For client side software this is very important since installation of client side software is usually performed by the user (e.g., student, teacher).

We have localized Mozilla into Lithuanian having in mind using it in schools as internet browser [6]. Localization was supplemented by original bookmarks, Lithuanian spell checker; all texts were edited in the same style as textbooks. A thousand packages were distributed to schools in 2004. In September 2004 a questionnaire was performed among schools about open source software. Answers to the question "Why do you choose Mozilla" were as follows: it is free - 35%, it is localized - 33%, it equipped with default Lithuanian bookmarks - 18%, Lithuanian code tables were set as defaults in e-mail component - 14%.

At the end of 2004 the first stable version of Mozilla Firefox appeared and trademarks and logotype policy was introduced. This policy imposed some restrictions

into localizations. Possibility to have full localization was granted by permission to escape restrictions by developing independent localization named Firefox Community Edition. We have made both variants (Table 2).

Table 2. Comparison of localization properties of Mozilla Firefox and Firefox Community Edition

No	Item	Firefox Community Edition	Mozilla Firefox
1	Default home page	Lithuanian original page, edited	Partially translated Mozilla Community page, not edited
2	Default bookmarks	About 200 Lithuanian sites including state and EU offices, communities, science, education, culture, press, transport timetables, telephone books, etc.	10 sites of international Mozilla and Firefox communities
3	Default search engines, suitable for Lithuanian locale	Google, Yahoo, Netsprint, Eurovok (Lithuanian Dictionary)	Google, Yahoo, Wikipedia
4	Help	Translated and adapted	Translated
5	Install setup component	Localized	Partly localized
6	Uninstall component	Localized	Partly localized
7	Graphics (e.g., text in themes and extensions preview pictures)	Localized	Not localized

Comparison is in favor of Firefox Community Edition, especially for using it at schools.

Both variants have yet unsolved bugs originating from an insufficient level of original internationalization. Let us list the most important ones:

1. The decimal separator is hard coded as point. It appears in page format settings. This bug is registered on 2002-08-05 (assigned number 161116) in Mozilla bug tracking system Bugzilla.

2. Matching singular and plural forms was not implemented properly. The situation was corrected only partly using abbreviations of words.

The latter defect was solved for the Calendar program as add-on for Mozilla [7]. It waits the similar solution for Firefox.

4 Conclusions

The analysis of web-based learning environments revealed the following issues.

1. Localization process of open source web-based learning software runs irregularly. At the moment many localizations are underway, but only a few are completed.

2. The list of internationalization bugs, specific to virtual learning environments was compiled. It may be used for testing purposes of such environments.
3. Internationalization of virtual learning environments is not sufficient for effective localization and do not ensure high quality of localizations.
4. Some internationalization defects of internet software increase costs of localization and prevent having full and proper localizations. Investigation and publication of internationalization defects and impact on authors of original programs are desirable.

References

1. Dagiene, V., Grigas, G.: Quantitative evaluation of the process of open source software localization. *Informatica*, Vol. 17, No 1. Institute of Mathematics and Informatics, Lithuanian Academy of Sciences, Vilnius (2006) 3–12
2. Esselink, B.: A practical guide to localization. John Benjamins B.V. (2000)
3. ISO 31-1: Quantities and units – Part 1: Space and time (1992)
4. ISO 8601: Data elements and interchange formats - Information interchange - Representation of dates and times (2004)
5. ISO/IEC 15897: Information technology - Procedures for registration of cultural elements (1999)
6. Jevsikova T., Dagiene V., Grigas, G: Mozilla Internet application suite: developing for education. 2nd International Conference on Information Technology: Research and Education. Proceedings, London, 96–100 (2004)
7. Jevsikova, T.: Programu adaptavimas lietuviškai lokalei (Software adaptation to Lithuanian locale). *Informacines technologijos 2003, Konferencijos pranešimų medžiaga* (Conference proceedings), ISBN 9955-09-335-8, Technologija, Kaunas (2003) I-(8–14) (in Lithuanian)
8. Schäler, R.: The cultural dimensions in software localization. *Localisation Focus*, Vol. 1, Issue 2. Localization Research Centre, Limerick (2002)
9. Uren, E., Howard, R., Perinotti, T.: Software internationalization and localization. Van Nostrand Reinhold (1993)

Author Index

- Allegra, Mario 290
Antonitsch, Peter K. 59

Blonskis, Jonas 220
Boyle, Roger D. 83
Butler, Deirdre 255

Cha, SeungEun 199
Chiazzese, Giuseppe 290
Chifari, Antonella 290
Choi, SookKyong 199
Cibulskis, Gytis 299
Clark, Martyn A.C. 83
Cormack, Gordon 230

Dagienė, Valentina 1, 220
Dzemyda, Gintautas 1

Futschek, Gerald 159

Ginat, David 127
Goodwin, Neville 242

Haberman, Bruria 38, 94
Han, HeeSeop 199
Hartnell-Young, Elizabeth 279
Hoyles, Celia 267
Hromkovič, Juraj 25
Hubwieser, Peter 104

Jang, HyeSun 199
Jevsikova, Tatjana 310
Jones, Duncan 267

Kahn, Ken 267
Kalas, Ivan 13
Kamada, Toshiyuki 138
Kanemune, Susumu 138
Kemkes, Graeme 230
Kim, JongHye 199
Kim, Yong 199
Kurebayashi, Shuji 138
Kwiatkowska, Anna Beata 209
Kwon, DaiYoung 199

Laucius, Rimgaudas 169
Lee, HwanCheol 199
Lee, WonGyu 199

Markauskaite, Lina 242
Martin, Fred 255
Merlo, Gianluca 290
Micheuz, Peter 189

Noss, Richard 267

Ottaviano, Simona 290

Park, JongEun 199

Reid, David 242
Reimann, Peter 242

Salanci, Lubomir 179
Sapagovas, Mifodijus 1
Sendova, Evgenia 71
Seta, Luciano 290
Shin, EunMi 199
Song, JaeShin 199
Strohecker, Carol 255
Syslo, Maciej M. 209
Szlávi, Péter 48

Targamadzé, Aleksandras 299
Todaro, Giovanni 290

Vasiga, Troy 230
Verhoeff, Tom 150

Weigend, Michael 117

Yehezkel, Cecile 38
Yeum, YongChul 199
Yoo, SeungWook 199

Zsakó, László 48