

Commentarii  
informaticae didacticae | 6

Ira Diethelm | Jannik Arndt | Malte Dünnebier |  
Jörn Syrbe (Eds.)

**Informatics in Schools:  
Local Proceedings of the  
6th International Conference  
ISSEP 2013  
– Selected Papers**

Oldenburg, Germany, February 26–March 2, 2013

Universitätsverlag Potsdam



## Commentarii informaticae didacticae (CID)



Ira Diethelm | Jannik Arndt |  
Malte Dünnebier | Jörn Syrbe (Eds.)

## **Informatics in Schools**

Local Proceedings of the 6th International Conference  
ISSEP 2013 – Selected Papers

Oldenburg, Germany, February 26–March 2, 2013

Bibliographic information published by the Deutsche Nationalbibliothek  
The Deutsche Nationalbibliothek lists this publication in the Deutsche  
Nationalbibliografie; detailed bibliographic data are available in the Internet  
at <http://dnb.dnb.de>.

Universitätsverlag Potsdam 2013  
<http://verlag.ub.uni-potsdam.de/>

Am Neuen Palais 10, 14469 Potsdam  
Tel.: +49 (0)331 977 2533 / Fax: 2292  
E-Mail: [verlag@uni-potsdam.de](mailto:verlag@uni-potsdam.de)

ISSN (print) 1868-0844  
ISSN (online) 2191-1940

The series *Commentarii informaticae didacticae* (CID) is edited by:  
Johannes Magenheimer, Universität Paderborn  
Sigrid Schubert, Universität Siegen  
Andreas Schwill, Universität Potsdam

Typesetting: Jannik Arndt  
Proofreading and copy-editing: Ursula Carr, textcompany-bremen  
This work is protected by copyright.

Published online at the Institutional Repository of the University of Potsdam:  
URL <http://pub.ub.uni-potsdam.de/volltexte/2013/6368/>  
URN <urn:nbn:de:kobv:517-opus-63688>  
<http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-63688>

Also published in print at Universitätsverlag Potsdam  
ISBN 978-3-86956-222-3

## Preface

Informatics in Schools — the name of this conference sounds quite clear but defining what the subject Informatics in schools is and what it should be is one of the goals of this conference series. Since the first ISSEP in 2005, almost eight years ago, several chairs and program committee members from all over the world have provided room for presentations and discussions to get answers to these questions. Quite often it was stated that the fast changing face of Informatics in everyday life and in schools makes it so difficult to define it. But now, having gone through the reviewing and publishing process myself as a chair for this conference, I think I discovered the real challenge of Informatics in schools in international discussions: It is not the *fast changing face* but the huge range of *different understandings* and implementations of what Informatics in schools is in every single nation.

Just look at the many different names our authors chose to describe Informatics in schools in their country: from common keywords such as “Computer Science”, “Informatics” or “Informatics Education” to “ICT” and “programming” and even introducing new terms like “Information study education”. It definitely is a challenge on its own to map a specific situation existing only in your home country to a language of another one. Other countries have different school systems and maybe even a different culture without an equivalent for your situation and, as a consequence, no word in their language that fits exactly. Therefore, I am confident that the “Darmstadt Model” by Peter Hubwieser and his working group presented in the first keynote of this conference will support our international joint efforts. This model aims at the comparability of different approaches and research results so that we can learn from each other even if we use different labels.

But as the names differ, so do the perceptions of the subject and its contents: We have a wide range of topics addressed in these proceeding’s papers: From programming in primary schools to NP-completeness in secondary education, from bilingual informatics education to the motivation for STEM, the efforts of word processing for mental models, computational thinking and last but not least the very important field of teacher education. The workshops reflect this variety as well. From my point of view the ISSEP is the conference with the widest range of papers focusing on Informatics in schools from the largest number of different countries. From this perspective the name of the conference could also be “Informatics in schools — the same but slightly different”.

At the ISSEP 2013 we are proud to offer papers by authors from 15 different countries. The conference proceedings are traditionally published in two books: 17 papers including the two keynotes appear in the Springer LNCS series [1] and another 14 papers and five workshop descriptions are collected in the “local proceedings”, where “local” refers to the fact that the publisher changes depending on the location of the conference. Publishing in the “commentarii informaticae didacticae”-series (latin for comments on Informatics education) provides the electronic access to these papers as well [2].

Finally, I would like to thank my co-chair Roland Mittermeir for his extraordinary support during the preparation process and especially for his help with the Springer LNCS proceedings. I would also like to thank Malte Dünnebier and Jörn Syrbe a lot; they did most of the work of bringing the conference to life in Oldenburg using their excellent organizational talents. For the local proceedings I have also had great support by Jannik Arndt and Ursula Carr during the editing process. Many acknowledgements also go to the PC members and additional reviewers who provided their precious expertise and time. But a conference is nothing without the authors, presenters and attendees. So, thank you all for writing, presenting, hearing, and reading parts of these proceedings and taking part in the workshops and discussions to move one or two steps forward towards a sustainable informatics education for pupils of all ages.

Oldenburg, January 2013

Ira Diethelm

[1] Diethelm, I., Mittermeir, R. T. (eds.): Informatics in Schools — Sustainable Informatics Education for Pupils of all Ages: 6th International Conference on Informatics in Schools: Situation, Evolution and Perspectives, ISSEP 2013, Oldenburg, Germany, February 26–March 2, 2013. LNCS, vol. 7780. Springer, Berlin (2013)

[2] <http://pub.ub.uni-potsdam.de/volltexte/2013/6368/>

## **Main Sponsors**

UGO – Universitätsgesellschaft Oldenburg e.V.

CEWE COLOR AG & Co. OHG, Meerweg 30–32, 26133 Oldenburg, Germany

IBM Deutschland GmbH, IBM-Allee 1, 71139 Ehningen, Germany

DFI – Department of Computing Science – Carl von Ossietzky University of Oldenburg



# Conference Organization

## Conference Chair

Ira Diethelm

## Program Committee

Ira Diethelm (U. Oldenburg, Chair)  
Torsten Brinda (U. Duisburg-Essen)  
Valentina Dagiene (Vilnius U.)  
Malte Dünnebier (U. Oldenburg)  
David Ginat (Tel-Aviv U.)  
Nataša Grgurina (U. Groningen)  
Peter Hubwieser (TU München)  
Juraj Hromkovič (ETH Zürich)  
Ivan Kalaš (Comenius U. Bratislava)  
Susumu Kanemune (Osaka Electro-Communication University)  
Maria Knobelsdorf (TU Dortmund)  
Robert McCartney (U. of Connecticut)  
Roland Mittermeir (U. Klagenfurt, Co-Chair)  
Viera K. Proulx (Northeastern University, Boston)  
Ralf Romeike (U. Potsdam)  
Sigrid Schubert (U. Siegen)  
Carsten Schulte (FU Berlin)  
Jenny Sendova (Bulgarian Academy of Sciences, Sofia)  
Simon (U. Newcastle)  
Maciej M. Sysło (Nicolaus Copernicus U., Torun)  
Jörn Syrbe (U. Oldenburg)  
Jan Vahrenhold (U. Münster)  
Tom Verhoeff (TU Eindhoven)

## Additional Reviewers

Ivan Derzhanski, Yayoi Hofuku, Martina Kabatova, Toshiyuki Kamada, Krassimir Manev, Neli Maneva, Yoshiaki Nakano, Tomohiro Nishida, Midori Nobe, Seiichi Tani, Renate Thies, Peter Tomcsányi, Michal Winczer

## Organizing Committee

Ira Diethelm (Chair), Jannik Arndt, Christian Borowski, Marion Bramkamp, Marius Dehé, Malte Dünnebier, Claudia Hildebrandt, Yuliya Meijer, Ana-Maria Mesaroş, Jörn Syrbe, Manuela Wüstefeld; all Carl von Ossietzky Universität Oldenburg, Dept. of Computing Science – Computer Science Education

# Table of Contents

## Towards Programming Competences

Why is programming difficult? — Proposal for learning programming in “small steps” and a prototype tool for detecting “gaps” . . . . .	13
<i>Yayoi Hofuku, Shinya Cho, Tomohiro Nishida and Susumu Kanemune</i>	

Test items for and misconceptions of competences in the domain of logic programming . . . . .	25
<i>Barbara Linck</i>	

Teaching NP completeness in secondary schools. . . . .	35
<i>Seungyon Kim and Seongbin Park</i>	

## Primary Education

Introducing Topics from Informatics into Primary School Curricula: How do teachers take it? . . . . .	41
<i>Jiří Vaníček</i>	

Environments for programming in primary education. . . . .	53
<i>Monika Gujberová and Peter Tomcsányi</i>	

## Towards National Curricula

A comparison of current trends within Computer Science teaching in school in Germany and the UK . . . . .	63
<i>Valentina Dagiene, Tatjana Jevsikova, Carsten Schulte, Sue Sentance and Neena Thota</i>	

Informatics Education in Turkey: National ICT Curriculum and Teacher Training at Elementary Level . . . . .	77
<i>Yasemin Gülbahar, Mustafa İlkhán, Selcan Kilis and Okan Arslan</i>	

The New Course of Study and a prospect of information studies education in Japan. . . . .	89
<i>Yoshiaki Nakano and Katsunobu Izutsu</i>	

## Computational Thinking

The Role of Algorithm in General Secondary Education Revisited. . . . .	99
<i>Daniel Lessner</i>	

An Epistemic Hypermedia to Learn Python as a Resource for an Introductory Course for Algorithmic in France. . . . . 111  
*Christophe Reffay, Mahdi Miled, Pascal Ortiz and Loïc Février*

Computational Thinking in Dutch Secondary Education . . . . . 119  
*Nataša Grgurina*

## **Different Views**

A Model for Teaching Informatics to German Secondary School Students in English-language Bilingual Education . . . . . 127  
*Martin Weise*

What you see is what you have in mind: constructing mental models for formatted text processing . . . . . 139  
*Carlo Bellettini, Violetta Lonati, Dario Malchiodi, Mattia Monga, Anna Morpurgo and Mauro Torelli*

## **Workshops**

Integrating School Practice in Austrian Teacher Education . . . . . 151  
*Lukas Planteu, Bernhard Standl, Wilfried Grossmann and Erich Neuwirth*

BubbleSort, SelectSort and InsertSort in Excel & Delphi – Learning the Concepts in a Constructionist Way . . . . . 153  
*Jan Benacka*

Problem-solving strategies must be taught implicitly . . . . . 155  
*Noa Ragonis*

.NET Gadgeteer Workshop . . . . . 159  
*Sue Sentance and Steve Hodges*

Using Teachers' TryScience to support educators and improve teaching. . . . . 161  
*Carol Berry and Peter Kusterer*

Author Index . . . . . 165



# **Towards Programming Competences**



# Why is programming difficult?

## Proposal for learning programming in “small steps” and a prototype tool for detecting “gaps”

Yayoi Hofuku<sup>1,4</sup>, Shinya Cho<sup>2</sup>, Tomohiro Nishida<sup>3</sup> and Susumu Kanemune<sup>4</sup>

<sup>1</sup> Sagami Kouyoukan High School

<sup>2</sup> Meisei University

<sup>3</sup> Osaka Gakuin University

<sup>4</sup> Osaka Electro-Communication University

**Abstract.** In this article, we propose a model for an understanding process that learners can use while studying programming. We focus on the “small step” method, in which students learn only a few concepts for one program to avoid having trouble with learning programming. We also analyze the difference in the description order between several C programming textbooks on the basis of the model. We developed a tool to detect “gaps” (a lot of concepts to be learned in a program) in programming textbooks.

## 1 Introduction

For students in the digital age it is important to understand that electronic equipment works with a computer and how it works. It is desirable for them to write programs so that they understand how a computer works according to the given instructions in a program. However, a lot of students drop out of introductory programming courses.

Why do students have difficulties with programming, although it is a creative activity and could be fun for them? In our class we often find that students have lost their way when they study particular contents that have many things to learn. That students are lost when they study a particular content with many things to learn. Therefore, we suppose that students are lost when they study a particular content with many things to learn (we say that such programs have “gaps”), so we developed a tool that detects new learning elements between the exercises in programming textbooks.

In this paper we will show the “small step” method which, by reducing the amount of new concepts, helps the students understand programming and the problems in existing textbooks.

We explain the tool and report on the results of applying the tool to C programming textbooks.

## 2 “Small steps” in programming learning

There is lots of research on the difficulties for novices in learning to program [1, 2]. Novice programmers often face difficulties already in basic courses [3]. There is a lot

of novices think that programming is difficult and requires too much knowledge and too many skills to be learned all at once and right from the beginning [4, 5].

Skinner [6] developed teaching machines and programmed instruction that presents material structured stepwise in a logical sequence. His work affects self-study materials for CAI, e-learning systems, and so on.

We focus on the principle that Skinner stated:

“In acquiring complex behavior the student must pass through a carefully designed sequence of steps, often of considerable length. Each step must be so small that it can always be taken, yet in taking it the student moves somewhat closer to fully competent behavior”. [6]

We call this principle “small steps”. This principle is efficient in studying mathematics and a foreign language.

Our goal is to also apply this principle to programming learning and help novice students to the learning of programming smoothly.

Our hypothesis is that if there are a lot of new syntax elements in one program, students may feel high gaps and find it difficult to understand further programs.

### 3 Problems in programming textbooks having “gaps”

Teachers or authors of programming textbooks tend to add lots of new concepts (we call them “gaps”) in each exercise because they know various concepts of programming. As a result, they make big “gaps” between exercises, and learners have difficulty solving them.

Fig. 1 shows an example of a big gap in a C-course exercise. First the teacher gives a simple example that displays a string constant. Next he/she gives an example that displays a variable number. She or he thinks it is an easy task because it contains only one concept “displays a number that is calculated from a variable”.

However, there are lots of new concepts in the second program.

- declaration of variable: `int n`
- data types: `int`
- substitute a constant for a variable: `n = 10`
- arithmetic operation: `n+20`
- formatting: `%d`
- multi argument of a function: `printf("%d", n+20)`

<pre>printf("Hello");</pre>	<pre>int n = 10; printf("%d", n+20);</pre>
-----------------------------	--

Fig. 1. Example of a big gap



It is not easy to detect new concepts even if we only show a two-line program. We focus on these gaps that teachers cannot identify. In particular, we focus on repetition (looping) structures with which lots of students have difficulty.

Repetition structures in C are much more complicated. Fig. 2 shows examples of two kinds of repetition: “for” and “while” structures.

<pre>int i; for (i = 0; i &lt; 10; i++) {     printf("Hello"); }</pre>	<pre>int i = 0; while (i &lt; 10) {     printf("Hello");     i++; }</pre>
--	---

**Fig. 2.** Example of repetition in C (for, while)

Both programs in Fig. 2 show that the instruction repeats “display a string Hello” 10 times. To understand these programs, learners should know “variables”, “data type (int)”, “variables initialization”, “assignment”, “compare operators”, “Boolean values” and “increment operator”. Many concepts are necessary to understand these simple structures.

General-purpose programming languages like C have multiple expressions to express repetition structures. For example “for”, “while” and “do-while”.

Therefore, teachers tend to give various examples and exercises to students, but these are too much to master, so learners may be confused when, for example, a “for” program contains “if”.

Moreover, a nested structure would bring confusion to the students.

Which should be the first structure when teaching repetition structures? Some teachers teach “for” first because a number of repetitions can be indicated clearly with the fixed numbers. However, as shown above, learners must know many concepts in order to understand “for(i=0 ; i<10 ; i++)”. Therefore they cannot understand it and rely on rote memorization. On the other hand, teachers wonder why learners forget it so fast.

The “small steps” teaching approach would resolve this problem in the following ways.

- **Changing order:** For example, we should teach “conditions”, “increment operator” and “if”, before we teach “while”. After students understand all of them, we can teach “for”.
- **Adding a new program:** For example, if the teacher shows students the program shown in Fig. 3, before he/she shows them the program on the right side of Fig. 1, they only need to learn an arithmetic operation, so the gap is lowered.

<pre>int n = 10; printf("%d", n);</pre>
---

**Fig. 3.** Example of adding a new program

## 4 Analysis of C language textbooks

To verify the hypothesis described above we analyzed programs that appear in C language textbooks.

### 4.1 Analyzing textbooks for introductory programming

As examples of textbooks for introductory programming we analyzed five textbooks [8–12] focused on the learning order of concepts. Table 1 shows the order of each textbook.

**Table 1.** Analysis of five textbooks

A	B	C	D	F
variables	variables	variables	variables	variables
...	types	...	assignments	types
assignments	assignments	assignments	types	assignments
expressions	arrays	(arithmetic)	(arithmetic)	arrays
...	strings	expressions	expressions	(arithmetic)
types	expressions	casts	casts	expressions
		(comparison)	...	(compound)
		expressions		expressions
if	if	if	for	if
switch	for	switch	while	while
do	while	for	do	do
while	do	while	if	for
for	switch	do	switch	switch

This result shows two problems with the order of learning control structures.

The first problem is “teaching everything”. Some textbooks show many concepts that do not need to be learned by novice students to explain language specifications. For example, textbook B explains arrays at the introductions of the types and all operators of C language.

The second problem is “disorder”. Some textbooks teach in an order that does not match with the structure of understanding. For example, textbooks B ,C and D introduce “while” statements after “for” statements. To understand “for” statements, the understanding of “if” statements (“condition”) and “while” statements (“repetition”) are needed, but students do not learn repetition at this time.

Textbook A introduces conditional expressions of “if” statements in a weird order; it starts with  $x \% 5$ . This example contains two concepts: “behavior of  $\&\& \text{if}$  statements” and “interpretation of integer values as Boolean values (yes or no)”. Table 2 shows the alternative order of conditional expressions. It would be easier if students start learning with a comparison expression. It is easy to grasp an image of boolean

values (comparison is answerable by yes or no). And only then students should learn the fact that “all integer values have the meanings yes or no (Boolean) in C language”.

**Table 2.** Order of comparison expression in textbook A and the alternative order

No.	Textbook A	Alternative order
3-1	if (x% 5)	if (x1 == x2)
3-2	if (x% 2)	if (x1 > x2)
3-3	if (x% 5) else	if (x1 == x2) else
3-4	if (x% 2) else	if (x != 0) else
3-5	if (x) else	if ((x%2) !=0)
3-6	if (x1 == x2) else	if (x%2) else

Taking these points into account when the teacher uses a textbook to teach programming, they can help students understand smoothly by changing the order of programs or adding supplemental programs that are not in the textbook.

The order of programs in a textbook reflects the philosophy of the author and is the standard order when using the textbook. We are interested in whether the standard order and the “ideal” order are the same. If students and teachers learn in the ideal order, they must feel that it is “easy to learn” and “easy to teach”.

## 5 Tools for detecting “gaps” in programming textbooks

As we described before, most textbooks have gaps. Students and teachers feel that gaps are difficult. We propose to change the order of programs in textbooks or provide other programs to fill these gaps. However, there are some problems.

- It takes a long time to find the gaps in textbooks.
- If a teacher tries to fill the gap, he/she is not sure whether the gap is really lowered.

Our hypothesis is that if there are a lot of new syntax elements in one program, students may feel high gaps and may find it difficult to understand further programs.

To solve these problems, we developed a tool named “De-gapper”, which detects new syntax elements of each program in programming textbooks.

### 5.1 Overview of the tool

De-gapper requires a set of program files as an input, arranged according to occurrence in the textbook. For each program file, De-gapper checks and reports if there are learning concepts appearing for the first time in the textbook.

Fig. 4 shows an example of input program files and output report of De-gapper.

If there are too many new learning concepts for one program, the program is expected to be too difficult to be understood by students.

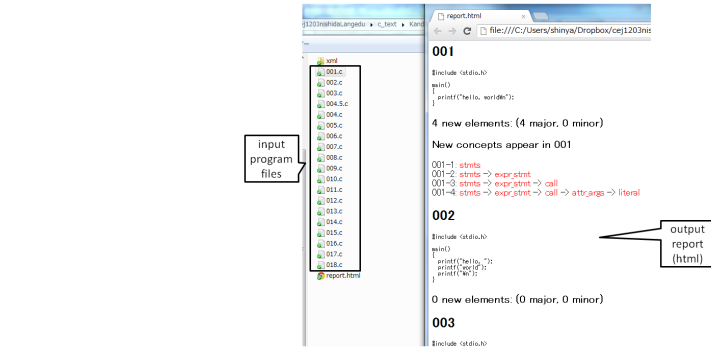


Fig. 4. An example of input and output of De-gapper

This tool is useful for a teacher teaching programming with a textbook. De-gapper tells him/her in advance which parts of the programs in the textbook are difficult, and he/she can prepare supplemental explanations for the difficult parts.

It is also useful for the author of the programming textbook. De-gapper lists all learning concepts that appear in the textbook. The author can check whether all learning concepts are explained in the textbook.

## 5.2 Implementation

De-gapper consists of a “parser” and a “checker”. The parser parses all program files in the textbook and outputs syntax trees as XML files (called “XML tree files”). Fig. 5 shows a sample input and output of the parser. The snippets parts of Fig. 5 contain tags for the entire program and function definitions. Since this paper will discuss programs that have just one function, we will omit these tags.

The checker scans all XML tree files in the order of occurrence in the textbook. For each XML tree file, the checker detects the paths of tag hierarchy. For example, the checker will detect the following paths in the XML tree file in Fig. 5:

```
list0101-1: stmts
list0101-2: stmts -> expr_stmts
list0101-3: stmts -> expr_stmts -> call
list0101-4: stmts -> expr_stmts -> call -> attr_args -> add
list0101-5: stmts -> expr_stmts -> call -> attr_args -> add -> num
list0101-6: stmts -> expr_stmts -> call -> attr_args -> add -> op
list0101-7: stmts -> expr_stmts -> call -> attr_args -> literal
```

De-gapper names these concepts “list0101-1” to “list0101-7”. Note that they are ordered by path string, not by occurrence in the program.

If there are paths that have not appeared yet in previously checked XML tree files, the paths are detected as “new learning concepts” of the program corresponding to the XML tree file.

Current implementation can parse C and Java program files. But it is easy to make it workable for other languages if we add parsers for these languages.

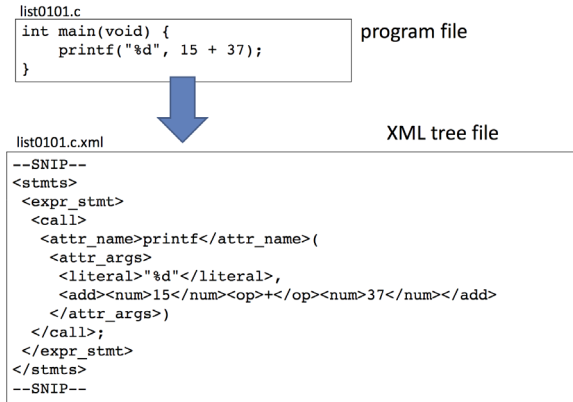


Fig. 5. Example of input and output of the parser

### 5.3 Examples

**Example 1 — Small Example:** Suppose that there is a textbook whose first program is `list0101.c`, shown in Fig. 5. De-gapper detects all paths of the program as new learning concepts. These can be explained as:

- `list0101-1`: The first example of a list of statements.
- `list0101-2`: The first example of an expression statement.
- `list0101-3`: The first example of a function call in a statement.
- `list0101-4`: The first example of an additive expression in a function call.
- `list0101-5`: The first example of a number in an additive expression.
- `list0101-6`: The first example of a `+` operator in an additive expression.
- `list0101-7`: The first example of a literal in a function call.

If the next program in the textbook is the one shown in Fig. 6, De-gapper detects 11 new learning concepts named `list0102-1` to `list0102-11`. These can be explained as:

- `list0102-1` to `list0103-3`: The first example of a variable declaration.
- `list0102-4`: The first example of a variable in a function call.
- `list0102-5`: The first example of an assignment.
- `list0102-6`: The first example of a variable in the left member of an assignment.
- `list0102-7`: The first example of an additive expression in the right member of an assignment. (at `vy=vx+10;`)
- `*list0102-8`: The first example of a number in an additive expression (at `vy=vx+10;`).
- `*list0102-9`: The first example of a `+` operator in an additive expression (at `vy=vx+10;`).
- `list0102-10`: The first example of a variable in an additive expression (at `vy=vx+10;`).

- list0102-11: The first example of an immediate number in the right member of an assignment (at vx=57;

```
list0102.c
#include <stdio.h>
int main(void) {
    int vx, vy;
    vx = 57;
    vy = vx + 10;
    printf("vx = %d\n", vx);
    printf("vy = %d\n", vy);
}

list0102-1: decls
list0102-2: decls -> decl
list0102-3: decls -> decl -> attr_declarators -> declarator
list0102-4: stmts -> expr_stmt -> call -> attr_args -> var
list0102-5: stmts -> expr_stmt -> let
list0102-6: stmts -> expr_stmt -> let -> attr_left -> var
list0102-7: stmts -> expr_stmt -> let -> attr_right -> add
*list0102-8: stmts -> expr_stmt -> let -> attr_right -> add -> num
*list0102-9: stmts -> expr_stmt -> let -> attr_right -> add -> op
list0102-10: stmts -> expr_stmt -> let -> attr_right -> add -> var
list0102-11: stmts -> expr_stmt -> let -> attr_right -> num
```

**Fig. 6.** Example of the program that contains variables first

In these concepts, list0102-8 and list0102-9, annotated with \*, are not completely “new” learning concepts for the following reasons:

- Students learn list0102-7, meaning that they can write additive expressions on the left member of an assignment.
- Additive expressions are already learned in list0101-4.
- They also learned list0101-5 and list0101-6. They already know that they can write + operators or numbers in additive expressions.
- Thus, they would imagine easily that they can write + operators or numbers in an additive expression on the left member of an assignment.

We will call such new concepts “minor” new concepts, and the rest “major” new concepts.

**Example 2 — The K & R book:** We will show another example. We applied De-gapper to “The C programming language” [7], from 1.1 to 1.6. The result is shown in Table 3.

De-gapper found that program No. 3 had extremely high gap with 26 major new concepts. Many supplemental programs are needed. However, we do not discuss this program in this section (This gap is apparent even if we do not use De-gapper). We will

**Table 3.** New concepts in the K & R book from 1.1 thru 1.6

No.	Section	# of new concepts (major/minor)	The first example of:
1	1.1	4/0	<code>printf("Hello, world\n");</code>
2	1.1	0/0	multiple <code>printf</code>
3	1.2	26/4	<code>int</code> variables and “while” statement
4	1.2	1/2	<code>float</code> variables
5	1.3	6/22	“for” statement
6	1.4	1/0	<code>#define</code>
7	1.5.1	2/0	<code>c=getchar();</code> (receiving return value of function call)
8	1.5.1	3/4	<code>c=getchar();</code> in loop condition of “while” statement
9	1.5.2	3/1	increment prefix <code>++</code>
10	1.5.2	3/1	“for” statement with an empty statement
11	1.5.3	3/5	“if” statement and character literal
12	1.5.4	10/24	<code>else</code> part and <code>  </code> operator
13	1.6	15/17	arrays and <code>&amp;&amp;</code> operator

focus on the programs that follow No. 3. From No. 4 to No. 11, most programs have quite low gaps, with 1 to 3 new concepts. Number 5 has a little bit of a high gap with 6 new concepts. Fig. 7 shows program No. 5 (also No. 4 to check gaps).

This is the first example of a “for” statement. Number 5 has the following major new concepts:

```
005-1:  stmts -> for
005-2:  stmts -> for -> attr_condition -> comp
005-3:  stmts -> for -> attr_condition -> comp -> num
005-6:  stmts -> for -> attr_initializer -> let
005-12: stmts -> for -> attr_loop -> expr_stmt -> call
        -> attr_args -> mul
005-23: stmts -> for -> attr_increment -> let
```

When students learn the “for” statement, the following concepts are mandatory:

- 005-1: the “for” statement itself
- 005-6: the initializer part of statement
- 005-2: the condition part of statement
- 005-23: the increment part of statement

We focus on the remaining parts: 005-3 and 005-12. These are interpreted as:

- 005-3: The first example of an immediate number in a comparison expression:  
`fahr <= 300 .`
- 005-12: The first example of an arithmetic expression in an argument of a function call:  
`printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));`

These concepts are not the essential parts of a “for” statement. Although the authors may intend to write program No. 5 as a transformation of No. 4 (“while” statement), it

```

#include <stdio.h>
/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    float lower, upper, step;

    lower = 0; /* lower limit of temperature scale */
    upper = 300; /* upper limit */
    step = 20; /* step size */

    fahr = lower;

    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
}

```

No. 4

No. 5

```

#include <stdio.h>
/* print Fahrenheit-Celsius table */
main()
{
    int fahr;
    for (fahr = 0; fahr <= 300; fahr = fahr + 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}

```

**Fig. 7.** First example of “for” statement in K & R book, in comparison with the previous “while” statement program.

is not a pure transformation; No. 4 contains the variables `lower`, `upper`, `step` and `celsius` to define ranges or keep a calculation result, while No. 5 does not contain them. The pure transformation would be like Fig. 8 named “No. 4.5”. If there was this program between No. 4 and No. 5, the number of the new concept would be 5:

```

004.5-1: stmts -> for
004.5-2: stmts -> for -> attr_condition -> comp
004.5-5: stmts -> for -> attr_increment -> let
004.5-10: stmts -> for -> attr_initializer -> let
004.5-13: stmts -> for -> attr_loop -> compound_statement

```

004.5-13 is a new concept that does not appear in No. 5, which contains no compound statement in the “for” statement. Aside from 004.5-13, the remaining four new concepts correspond to the mandatory concepts of “for” statements.

This example shows that the program in which De-gapper indicates many learning concepts does actually have difficult points.

Through these examples, De-gapper is expected to have the following benefits:

- De-gapper can find gaps from many programs automatically.
- De-gapper helps teachers find the gaps hidden in textbooks and gives them chances to “degap” (to prepare supplemental programs between gaps).
- After inserting supplemental programs, De-gapper can re-evaluate the gaps to check if the gaps are lowered.



```

#include <stdio.h>
/* print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300; floating-point version */
main()
{
    float fahr, celsius;
    float lower, upper, step;

    lower = 0; /* lower limit of temperature scale */
    upper = 300; /* upper limit */
    step = 20; /* step size */

    for (fahr = lower ; fahr <= upper; fahr = fahr + step) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
    }
}

```

**Fig. 8.** A proposal to prepare a supplemental program as the first example of a “for” statement in K & R book.

## 6 Summary and future work

We proposed a “small step” teaching method for programming learning and a tool for detecting “gaps”. We plan to develop the method and tool in the following ways:

*Weighting new concepts* The number of new concepts in a program is not always proportional to the “height” of gaps, i.e., how difficult the program appears to the students? We will have some experimental classes to measure gaps that De-gapper has calculated and the actual gaps that students felt.

*Correspondence between new concepts and program parts* In current implementation, the contents of new concepts are indicated by XML paths. They should be interpreted into understandable natural language. In addition, current implementation does not show which part of the program corresponds to the paths.

We plan to add a feature that automatically highlights which part of a program are new concepts or shows corresponds parts by hovering the mouse cursor over XML paths.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 22500828.

## References

1. McCracken, M., Almstrum, V., Diaz, D. et al.: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM, SIGCSE Bulletin*, Vol. 33, No. 4, pp. 125–180 (2001)
2. Gross, P., Powers, K.: Evaluating assessments of novice programming environments. *ACM, ICER'05*, pp. 99–110 (2005)
3. Lahtinen, E., Ala-Mutka, K., Jarvinen, H.-M.: A study of the difficulties of novice programmers. *ACM, SIGCSE Bulletin*, Vol.37, Issue 3, pp. 14–18 (2005)
4. Shuhidan, S. M., Hamilton, M., D'Souza, D.: Understanding novice programmer difficulties via guided learning. *ACM, ITiCSE '11*, pp. 213–217 (2011)
5. Spohrer, J., Soloway, E.: Novice mistakes: Are the folk wisdoms correct? *ACM, CACM*, Vol. 29, No. 7, pp. 624–632 (1986)
6. Skinner, B. F.: *Teaching Machines*. Science, Vol. 128, pp. 969–977 (1958)
7. Kernighan, B., Ritchie, D.: *C Programming Language* (2nd Edition). Prentice Hall PTR (1988)
8. Shibata, BY.: *Meikai C Gengo* 2nd ed, Nyumonhen. SoftBank Creative (2004) (in Japanese)
9. Hayashi, H.: *Shin C Gengo Nyumon* 2nd ed, Beginner Hen. SoftBank Creative (2003) (in Japanese)
10. Takahashi, M.: *Yasashii C* 2nd ed. SoftBank Publishing (2003) (in Japanese)
11. Kochan, S.: *Programming in C* 3rd ed. Sams Publishing (2005)
12. Perry, G.: *Absolute beginner's guide to C* 2nd ed. Sams Publishing (1994)

# Test items for and misconceptions of competences in the domain of logic programming

Barbara Linck

Didactics of Informatics and E-Learning, University of Siegen, Siegen, Germany

`barbara.linck@uni-siegen.de`

**Abstract.** Development of competence-oriented curricula is still an important theme in informatics education. Unfortunately informatics curricula, which include the domain of logic programming, are still input-orientated or lack detailed competence descriptions. Therefore, the development of competence model and of learning outcomes' descriptions is essential for the learning process in this domain. A prior research developed both. The next research step is to formulate test items to measure the described learning outcomes. This article describes this procedure and exemplifies test items. It also relates a test in school to the items and shows which misconceptions and typical errors are important to discuss in class. The test result can also confirm or disprove the competence model. Therefore, this school test is important for theoretical research as well as for the concrete planning of lessons. Quantitative analysis in school is important for evaluation and improvement of informatics education.

**Keywords:** competence, test items, misconceptions, informatics in upper secondary education, logic programming

## 1 Introduction

Development of competence-oriented curricula is still an important theme in informatics education. E.g., the IFIP Working Group 3.2 analyzed various curricula for informatics. They concluded that it is important for curricula to be well designed and to allow a "constructive approach with an open space for interpretation and effective knowledge transfer." [1, p. 189-190]. Hence, the competence orientation in informatics curricula is fundamental. Some frameworks refer to learning outcomes for informatics education e.g., [2-3] explain one of them. However, they do not include competences in logic programming. Logic programming is a theme of ten informatics curricula for upper secondary school in Germany. Unfortunately, these curricula are still input-oriented or lack detailed competence descriptions. For teachers the learning outcomes in informatics curricula are essential. They are the basis for defining which competences the students are supposed to learn. Especially, „During the preparation process the teacher might examine, for example, a section of a GCSE syllabus and consider exactly what are the key concepts and skills to be taught." [4, p. 245] There-

fore, the development of learning outcomes' descriptions is important for the learning process in the domain of logic programming.

A prior research established a competence model for the domain of logic programming [5]. Based on this model, competence descriptions were formulated [6]. The next research step is to develop test items. These items are necessary to measure the described learning outcomes. In addition, this measurement can confirm or disprove the competence model and the competence descriptions. This article describes the method of the items' development, exemplifies chosen test items, and relates the results of a test in school to the items. Furthermore, the answers of the test show which misconceptions and typical errors are important to discuss in class.

## **2 Test items and misconceptions**

Regarding to [7, p. 41] a framework, which describes the extent of the domain, is supposed to be a basis for test measurement. This framework delineates which aspects of the content, skills, processes etc. of the domain should be measured. Therefore, the developed competence model with competence descriptions for the domain of logic programming is the basis for the measurement. Moreover, the test format and the target group have to be taken into account while constructing the test. The following sections describe the test format, examples of the test and results out of it.

### **2.1 Method to develop test items**

There is a wide variety of test formats. E.g., multiple-choice questions, short answers, coding exercises, essays and filling-the-blank questions are possible. According to Bacon multiple-choice (MC) answers are as reliable as short-answer tests, "but an MC exam may be completed quickly." [8, p. 35] Haladyna et al. [9] presents a revised taxonomy of writing multiple-choice items. Among other things it is important to base each item on important content and to avoid trivial content. Moreover, characteristics of a well-written test are that the content of each item is independent from contents of other items, that the vocabulary is simple for the group of students and that the length of the choices is equal. Furthermore, it is important to minimize the amount of reading, to develop as many effective choices as possible - but three are adequate - and to use typical errors of students to write distractors. Kuechler and Simkin analyzed that multiple-choice questions "enable test takers to better guess correct answers compared to constructed response test" [10, p. 394] for the domain of programming. However, they showed in another article "that carefully constructed MC questions on the topic of basic computer language programming can test a broad range of levels of understanding of that subject." [11, p. 79] Therefore, it is very important to choose the distractors wisely. Furthermore, the students are supposed to decide whether the answers are chosen or not chosen. This allows a distinction between not chosen answers and missing statements. In addition, the test should include open answers as well. A described context, in which the learning outcome is supposed to be applied, is important for the competence orientation [12].

With regard to these demands and to the taxonomy of Anderson and Krathwohl [13] test items were developed and tested in a pretest in school. In addition, the competence descriptions were a basis for the test items. With respect to the pretest, in some cases it was necessary to modify the phrasing of the item. Moreover, appropriate items were chosen out of the collection of the items. This modified test was tested in school again and is described in this article.

## 2.2 Test group and test conditions

Informatics is often a facultative subject in Germany. Therefore, in the most cases only a few students take this course. The test items were tested in three different classes. Three students were in class A, six students in class B and 16 students in class C. Classes A and B were a basic informatics course with three 45-minute lessons per week. Class C was an advanced informatics course with five 45-minute lessons per week.

The test is quantitative, took 45 minutes and was handed out by the teacher. The following sections present and analyze two different test items. Both sections exemplify the test items, the analysis of the test results and the identification of misconceptions as well as typical errors.

## 2.3 Multiple-choice questions: selection of implementation

As shown in [6], one competence is to analyze a given problem and to implement it into the knowledge base. The competence description is: "The learners are able to analyze a given problem. They can implement it into facts and rules. Therefore, they are able to create different possible implementations. In addition, they can compare various implementations and select the best one." [6, p. 8] Item number two tests the comparison and selection of different implementations. The context of the item is that four friends talk about their holidays and in which countries each of them has been on holiday. Four possible implementations (see Table 1, answer 2a-2d) of this situation are given. The students are supposed to select the implementations, which can be used to answer the following two questions:

- Who has been to the United Kingdom?
- How many countries has Florian (one of the friends) been to?

In addition, the students are not supposed to use system predicates e.g. `findall/3` to formulate the queries. Especially the students in the classes A and B do not know these system predicates.

The right answer for this task is to choose the last implementation 2d. In Prolog – the logic program that is used in the three classes – the matching of the variables (at the head of the rule to the asked query) requires the knowledge of the position of the variables. In the given implementation two and three, the same country, e.g. the United Kingdom, has different positions. Therefore, they are not possible knowledge bases for answering the two questions. The first implementation 2a can be used to an-

swer the first question. Nevertheless, without the system predicate findall/3 it is not possible for the students to answer the second question.

**Table 1.** Possible implementations

<i>chosen</i>	<i>not chosen</i>		
		holiday(katrin, united_kingdom). holiday(katrin, france). etc.	2a
		holiday(katrin, united_kingdom, france, spain). holiday(florian, portugal, united_kingdom, netherlands). etc.	2b
		holiday_katrin(united_kingdom, france, spain). holiday_florian(portugual, united_kingdom, netherlands). etc.	2c
		holiday(katrin,[united_kingdom, france, spain]). holiday(florian,[portugual, united_kingdom, netherlands]). etc.	2d

The misconception of implementation 2a is that Prolog knows how many facts and rules belong to one predicate. Table 2, 3 and 4 show the distribution of the answers. The students who chose “false” give the right answer since it is not a possible implementation as already mentioned.

With regard to these tables this misconception should be discussed in class. Only 56 % of all students knew that this implementation could not be used to answer the two given questions. The students of the advanced informatics course were only slightly better with 56.25 % than the students of the basic informatics course with 55.56 %. Therefore, this misconception is very important for both courses.

**Table 2.** Answer 2a, all students

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	14	56	70
right	1	6	24	30
missing statements	99	5	20	missing statements
	total	25	100	100

**Table 3.** Answer 2a, basic informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	5	55.56	100
right	1	0	0	0
missing statements	99	4	44.44	missing statements
	total	9	100	100

**Table 4.** Answer 2a, advanced informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	9	56.25	60
right	1	6	37.50	40
missing statements	99	1	6.25	missing statements
	total	16	100	100

Moreover, this part of the test item can be used in a following test again. Data analysis presents that the answers 2a and 2d as well as 2c are not too similar and do not have to be changed. E.g., data shows that 24 % of the students chose answer 2a. Therefore, students do not skip the item immediately. This confirms that this item does avoid trivial content and includes typical errors as a distractor. Both demands are important for the development of test items [9]. 40 % of the students, who knew that answer 2d is a possible implementation, did not choose 2a as a right implementation. Hence, the misconception of 2a does not depend on answer 2d. In addition, 48 % of the students, who chose 2b as a wrong answer, did not choose 2a as a right answer. The same occurs with answers 2c and 2a. Therefore, the misconceptions of 2b and 2c are independent of the misconception of 2a. But only 8 % of the students, who chose answer 2c as a wrong answer, did not choose 2b as a wrong answer. As a result, answers 2b and 2c do not show enough reliability. For a repetition of the test one of these two answers should be erased or replaced by another misconception.

#### 2.4 Open answer of implementation: connecting predicates with each other

For the implementation of a given problem in facts and rules it is also necessary to use system predicates. Another important competence is to implement a rule, which accesses different predicates. The following item tests both. The described situation is a pizzeria, which has certain pizzas, pastas, desserts, a soup, two salads and a meat dish on its menu. The task is to implement a rule, which creates a menu with an appe-

tizer, a main dish and a dessert. In addition, the rule is supposed to display the price of the menu. This task demands an open answer. Table 5, 6, and 7 show how many students were able to implement a rule, which accesses the three predicates *appetizer*, *main dish* and *dessert*.

**Table 5.** Several predicates, all students

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	1	4	4.17
right	1	23	92	95.83
missing statements	99	1	4	missing statements
	total	25	100	100

**Table 6.** Several predicates, basic informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	1	11.11	12.5
right	1	7	77.78	87.5
missing statements	99	1	11.11	missing statements
	total	9	100	100

**Table 7.** Several predicates, advanced informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	0	0	0
right	1	16	100	100
missing statements	99	0	0	missing statements
	total	16	100	100

This analysis shows that 95.83 % of the students understood the described situation as well as the formulation of the tasks and gave the right answer. Therefore, this task does not need to be rephrased. Furthermore, a performance difference between the basic and the advanced informatics course is obvious. All students of the advanced course gained this competence, whereas 87.5 % of the basic course gave the right answer. The latter percentage still confirms the test item. As already mentioned, this



item also includes the competence to use system predicates. Table 8, 9 and 10 show how many students are able to implement the system predicate *is*.

**Table 8.** System predicate, all students

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	18	72	75
right	1	6	24	25
missing statements	99	1	4	missing statements
	total	25	100	100

**Table 9.** System predicate, basic informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	5	55.56	62.5
right	1	3	33.33	37.5
missing statements	99	1	11.11	missing statements
	total	9	100	100

**Table 10.** System predicate, advanced informatics course

<i>value label</i>	<i>value</i>	<i>frequency</i>	<i>percent</i>	<i>valid percent</i>
false	0	13	81.25	81.25
right	1	3	18.75	18.75
missing statements	99	0	0	missing statements
	total	16	100	100

According to these tables only 25 % of the students used the system predicate *is* in the right way. 37.5 % of the students in the basic informatics course achieved this competence. Compared to this, only 18.75 % of the students in the advanced informatics course gave the right answer. The test shows that most of the students in the latter course, who gave a wrong answer, mistook the system predicate = for the system predicate *is*. Thus, the misconception between both system predicates should be discussed with more examples in this course.

The result of the test is that the students did not use the system predicate *is* in the correct way. However, this does not imply that these students did not achieve the

learning outcome, which was supposed to be tested. Especially, the output orientation demands to have a closer look at what is tested. This test item shows that these students did not remember the system predicate *is*. The conclusion is not that they cannot use any system predicate correctly. Therefore, another test item should be added to the test. It would be interesting to analyze if the students are able to use a given system predicate in the correct way and, in addition, if they are able to remember as well as use another system predicate.

## 2.5 Test result: misconceptions

As shown in 2.3 the test items use misconceptions as distractors. Furthermore, the open answers in the test include typical errors and can be analyzed as well. 2.4 exemplifies this. With regard to all test items, the students avoided or made the following misconceptions and typical errors (see Table 11). The percentage refers to the valid percentage.

**Table 11.** Misconceptions of logic programming and test results

<i>misconception/typical error</i>	<i>shown by the students</i>
Only one of the arguments at the head of the rule has to match the query.	0 %
Wrong structure of the clause tree.	4 %
The position of the argument at the head is not important.	5 %
The system answers of $rule(A1,A2)$ is $rule(a1,a2)$ instead of $A1=a1, A2=a2$ .	12 %
Wrong order of the answers (first system answer as first answer in the test).	29 %
Anonyms variable ("_") can give an answer with a concrete instantiation.	30 %
Mistake the meaning of predicate and clauses.	44 %
No point at the end of each fact or rule.	44 %
Mistake the meaning of <i>is</i> and =.	46 %
Atoms can be written with a big initial letter.	68 %
Prolog knows how many facts and rules belong to one predicate.	70 %

Therefore, the test result can be used to improve the teaching of logic programming. The identified misconceptions and typical errors should be discussed in school.

Another interesting test result is that the students of the advanced informatics course were able to transfer their knowledge. The test included a system predicate to

modify the knowledge base dynamically. Although, the students did not learn the dynamic modification yet, 81.25 % of the students of the advanced course answered this question. 100 % of those students got the structure of the system predicate and the used arguments right. 61.54 % of them knew the system predicate. Most of them used the predicate *delete*, some used the predicate *retract*. It can be assumed that the students transferred the predicate *delete* from another programming language and matched it to the structure of Prolog. This hypothesis would be interesting to test in a further research.

### 3 Conclusion

In summary, this article shows that the distractors of the developed test items are well chosen. Only slight changes should be done for a next test. In some cases one distractor should be erased or replaced. In other cases additional test items can support the testing of the competences. Therefore, this test result can be used to conclude first findings referring to the competence model of logic programming. This will be a next research step.

Based on the test result, this article identifies also misconceptions and typical errors of logic programming. These misconceptions and errors are essential for the learning progress. Students should interact with them critically. This is fundamental for a deeper understanding of the subject. Hence, they are also important for the design of the lessons. The more often a misconception appears, the more it should be discussed in school. Even if a student does not make the typical error her- or himself, the understanding of it supports e.g., the understanding of the structure of facts and rules.

Therefore, this school test is important for theoretical research about competences in logic programming as well as for the concrete planning of lessons. Quantitative analysis in school also referring to other subjects is important for evaluation and improvement of informatics education.

### 4 References

1. Veen, M. van, Mulder, F., Lemmen, K.: What is Lacking in Curriculum Schemes for Computing/Informatics. In: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, pp. 186–190, ACM, New York (2004)
2. Antonitsch, P.: On Competence-Oriented and Learning Informatics. In: Proceedings of 5th International Conference ISSEP, CD-ROM, Bratislava (2011)
3. Micheuz, P.: A Competence-Oriented Approach to Basic Informatics Education in Austria. In: Proceedings of 5th International Conference on Informatics in Schools, Springer, Heidelberg, pp. 43–55 (2011)
4. Webb, M.: Pedagogical Reasoning: Issues and Solutions for the Teaching and Learning of ICT in Secondary Schools, In: Education and Information Technologies 7(3), pp. 237–255, (2002)
5. Linck, B., Schubert, S.: Logic programming in informatics secondary education. In: Proceedings of 5th International Conference ISSEP, CD-ROM, Bratislava (2011)

6. Linck, B.: Competence Descriptions for Informatics Education – using the example of logic programming. In: Proceedings of IFIP-Conference “Addressing educational challenges: the role of ICT (AECRICT)”, Manchester Metropolitan University (2012)
7. American Educational Research Association, American Psychological Association, National Council on Measurement in Education: Standards for educational and psychological testing. American Psychological Association, Washington DC (1999)
8. Bacon, D.: Assessing Learning Outcomes: A Comparison of Multiple-Choice and Short-Answer Questions in a Marketing Context. In: *Journal of Marketing Education* 25(31), pp. 31–36 (2003)
9. Haladyna, T., Downing, S., Rodriguez, M.: A review of Multiple-Choice Item-Writing Guidelines for Classroom Assessment. In: *Applied measurement in education* 15(3), pp. 309–334, Lawrence Erlbaum Associates, Inc. (2002)
10. Kuechler, W., Simkin, M.: How Well Do Multiple choice Tests Evaluate Student Understanding in Computing Programming classes? In: *Journal of Information Systems Education* 14(4), pp. 389–399 (2003)
11. Simkin, M., Kuechler, W.: Multiple-Choice Tests and Student Understanding: What Is the Connection? In: *Decision Science Journal of Innovative Education* 3(1), pp. 73–97 (2005)
12. Weinert, F.E.: Concept of Competence: A Conceptual Clarification. In: Rychen, D., Salganik, L. (eds.): *Defining and Selecting Key Competencies*. Seattle, (2001)
13. Anderson, L., Krathwohl, D.: *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives*. Longman, New York (2001)

# Teaching NP completeness in secondary schools

Seungyon Kim and Seongbin Park\*

Korea University, Seoul, Korea

{seungyon\_kim, hyperspace}@korea.ac.kr

**Abstract.** In this paper, we show how the theory of NP completeness can be introduced to students in secondary schools. The motivation of this research is that although there are difficult issues that require technical backgrounds, students are already familiar with demanding computational problems through games such as Sudoku or Tetris. Our intention is to bring together important concepts in the theory of NP completeness in such a way that students in secondary schools can easily understand them. This is part of our ongoing research about how to teach fundamental issues in Computer Science in secondary schools. We discuss what needs to be taught in which sequence in order to introduce ideas behind NP completeness to students without technical backgrounds.

## 1 Introduction

Informatics education in secondary schools typically includes programming as a way to solve computational problems. A programming course can consist of lectures on object-oriented languages such as Java or procedural languages such as C. Once students learn how to program, they may ask whether all computational problems are solvable or whether there exist more efficient solutions (or programs) for given problems than their own programs. These are issues of computability or computational complexity and it is difficult to teach these issues in secondary school level informatics classes because there are many technical terms that students need to understand beforehand.

In this paper, we propose a way by which NP completeness and other relevant issues can be taught to secondary school students who understand what algorithms are. We also discuss why teaching NP completeness is important in secondary school informatics classes. The motivation of this research is as follows. First of all, many students are familiar with games such as Tetris or Sudoku, but few students are aware of the fact that they are NP-complete problems [1, 2]. If they cannot find correct solutions for Sudoku and they do not know of the existence of NP-complete problems, they may just think the reason is that it is difficult to solve. In other words, instead of seeing the structural properties of the problem, they may simply think that it is difficult to solve. However, if we introduce the idea of NP completeness to students, they can see the problems from a general perspective and can understand why some specific instances of Tetris or Sudoku can be hard to solve. Secondly, students who can solve quadratic equations of the form  $ax^2 + bx + c = d$  can easily solve linear equations of the form  $ex + d = f$ , but they may not know that similar transformations (or reductions) exist among different

---

\* Corresponding author

computational problems as well. If students understand reductions used in NP completeness, they can classify problems that they know and possibly learn how to analyze an arbitrary computational problem.

## 2 What to teach

In this section, we explain how we can introduce the concept of NP completeness to secondary school students. To this end, we identify candidate topics that most students at secondary schools are familiar with. These can serve as starting points to learn the concept of NP completeness.

Typical topics that secondary school students learn include graphs of inequalities, number sequences, quadratic equations etc. When students learn these topics, they study how to solve different types of problems, but often they are not aware of whether they deal with problem instances or the problem itself. For example, when students learn quadratic equations, they study what quadratic equations are, how solutions for these equations could be found, and they work with some problem instances so that they feel they understand how to solve arbitrary quadratic equations. However, since NP completeness is a property of decision problems and it is important to understand the relationship among the set of problems which are NP-complete, we need to teach students how to relate one problem to another. This requires a clear understanding of the difference between a computational problem and its instances because a transformation between two computational problems can be defined in terms of their instances.

Once students understand the difference between a problem and a problem instance, they can understand the role of an algorithm precisely; i.e., an algorithm is a well-defined computational procedure [3] which takes the input of a problem (i.e. a problem instance) and returns a correct output in a finite number of steps. At this point we can explain why a computational problem that asks whether a program, downloaded by a student from the Internet, is a computer virus or not, is not computable [4] by mentioning that there is no consistent mechanism (i.e. algorithm) which can always give a correct answer for an arbitrary problem instance. We can even introduce a 3 CNF Satisfiability problem which exhibits a phase transition as the ratio of the number of clauses to the number of variables varies [5]. Since students are already familiar with the notion of phase transition from chemistry class, this phenomenon may sound exciting and they may be motivated.

To introduce the concept of reductions used in NP completeness, we can point out that two computational problems that look similar on the surface may have different structural properties. This can be explained using Hamiltonian cycle problem and Eulerian cycle problem [6]. We can mention that their structures are different because they ask to satisfy different constraints; i.e., one asks to visit every vertex once and the other asks to visit every edge once. This can naturally lead to a discussion about a class of problems that share certain common denominators in terms of their structures. For example, we can use a diagram about the web of reduction [7] which shows how different problems can be connected in terms of a special relation, called reducibility. From the figure, students can see some connection between the Satisfiability problem and the Hamiltonian cycle problem. Though it may be difficult to understand how the reduction

really works, students can at least realize that constraints existing in the Satisfiability problem are structurally similar to those in the Hamiltonian cycle problem. As a simple example of reduction, we can explain the Clique problem and show that a satisfying assignment for the Satisfiability problem can get us the information about a clique in a graph that is created based on the constraints in the Satisfiability problem.

To introduce properties shared by problems in NP, we can first explain problems that belong to a class called P. This can be done by recalling that the running time of an algorithm is typically given as a function of the size of input for the problem under consideration [8]. Students are familiar with how to count the number of steps of an algorithm that can be represented as a flowchart. We then point out that certain decision problems admit of efficient solutions which means they can be solvable in time proportional to some polynomial function of input size. We can also mention that there are decision problems whose solutions can be efficiently verified once we are given candidate solutions, but it is not proved yet whether they also admit efficient solutions. At this point, we can define class P as the set of decision problems which admit of efficient solutions and class NP as the set of decision problems which admit of efficient verification.

Once students understand properties of problems in P and NP, we can continue explaining a different way to define NP. That is, we explain that a decision problem belongs to NP if there is a nondeterministic algorithm which solves the problem efficiently [9]. But in order to understand this definition, students need to understand what a nondeterministic algorithm is. We can explain the concept by showing how a two-phase algorithm that consists of guessing and verifying solves a graph coloring problem [10]. We can also point out that an algorithm which we talk about in general is essentially a special type of nondeterministic algorithm that does not contain the first guessing step. In addition, we can explain that it is the reason why P is included in NP; i.e., a deterministic algorithm is a special type of a nondeterministic algorithm.

### 3 Conclusions

NP completeness is an important topic in the theory of computations and other areas in computer science. Understanding NP completeness is difficult because it requires a clear understanding of various terminologies and mathematical backgrounds. In this paper, we argue that the basic idea of NP completeness can be taught at secondary school. Our expectation is that if students learn the ideas behind NP completeness, they can broaden their horizons with regard to the set of computable problems and problem solving techniques.

### References

1. Breukelaar, R., Hoogeboom, H. J., Kusters, W. A.: Tetris is hard, made easy, Technical report, Leiden Institute of Advanced Computer Science, Universiteit Leiden (2003)
2. Aaronson, L.: Sudoku Science, IEEE Spectrum (2005)
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to Algorithms, second edition, The MIT Press

4. Lowther, J. L., Shene, C. K.: Toward an Intuitive and Interesting Theory Course: The First Step of a Road Map, *Journal of Computing Sciences in Colleges* (2004)
5. Hayes, B.: Can't Get No Satisfaction, *American Scientist* (1997)
6. Hayes, B.: Accidental Algorithms, *American Scientist* (2008)
7. Arora, S., Barak, B.: *Computational Complexity: A modern approach*, Cambridge University Press (2009)
8. Hromkovič, J.: *Algorithmic Adventures From Knowledge to Magic*, Springer Berlin Heidelberg (2009)
9. [http://en.wikipedia.org/wiki/NP\\_\(complexity\)](http://en.wikipedia.org/wiki/NP_(complexity)) (last checked: 1/31/2013)
10. Basse, S., Van Gelder, A.: *Computer Algorithms*, Addison Wesley (2000)



## **Primary Education**



# Introducing Topics from Informatics into Primary School Curricula: How do teachers take it?

Jiří Vaníček

University of South Bohemia in České Budějovice, Czech Republic  
vanicek@pf.jcu.cz

**Abstract.** The process of introducing compulsory ICT education at primary school level in the Czech Republic should be completed next year. Programming and Information, two topics from the basics of computer science have been included in a new textbook. The question is whether the new chapters of the textbook are comprehensible for primary school teachers, who have undergone no training in computer science. The paper reports on a pilot verification project in which pre-service primary school teachers were trained to teach these informatics topics.

**Keywords:** primary school, informatics curricula, teacher education

## 1 ICT as a compulsory subject in the Czech Republic

The new Education Act of 2004 brought about some major changes in the school curricula in the Czech Republic. The Framework Educational Programme (FEP) define the limits of compulsory education. The orientation of education is set on key competencies and favours outcome-oriented teaching. Information and communication technologies have become one of the nine educational areas for primary and lower secondary school level [1]. The subject Information and Communication Technologies (ICT) is compulsory from primary school level. The school reform is now entering its 5th year. This means that all children leaving primary school level should now have the skills and basic competencies to work with computers and with information.

The reality, however, is far from positive and does not correspond with the expectations. Although the one-year ICT course should be taught in any one year between the 1st and the 5th grade, most schools leave the subject for the last possible grade, i.e. the 5th. The introduction of this new subject meets with a number of difficulties:

- this subject has never before been taught at primary school level, there is no tradition of or experience with teaching it;
- the concurrent transition to new ways of creating school educational programmes and at the same time the shift of responsibility for creating and realizing curricula to the teachers and the schools made the introduction of this new subject more difficult;

- primary school teachers neither have been trained nor are being trained, they lack the necessary experience, there are no lifelong education courses organized by the Ministry of Education;
- teachers do not have methodological materials, pupils do not have textbooks, there has been no research in this area;
- primary school teachers have received no training and are still not receiving any training in the area of didactics of information technologies during their undergraduate studies.

The introduction of the subject Information and Communication Technologies has been and still is uncontrolled.

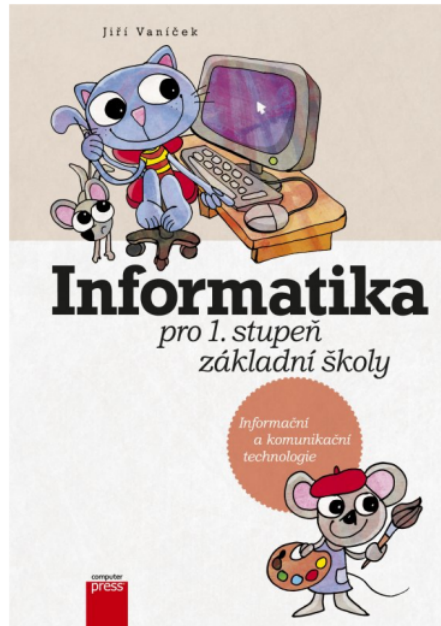
## 2 Basics of computer science in ICT curricula

Simultaneously with the process described above other changes that have an impact on the school curricula could be observed over the past few years. There is a trend of reintroducing into the teaching of work with computers not only topics of digital literacy, but also topics from the basics of computer science. Already in 2003 K-12 Curriculum Committee of The Association for Computing Machinery in the USA stated that the “goals of a K-12 computer science curriculum are to introduce the fundamental concepts of computer science to all students, beginning at the elementary school level”. [2] In the same document they also demand that “upon completion of grade 5, students should develop a simple understanding of an algorithm.”

Not only Czech statistics show a decline in the teenagers’ interest in technical subjects. For example the 2006 OECD document states “that between 1990–2005 the numbers of science and technology students have been decreasing in relative terms.” [3]. This results in increased attempts to endorse sciences in schools. The attempts to reverse the decline of interest in sciences are connected with the effort to establish computer science as a true science discipline. The initiative Computing at School defines “Computer Science as the fourth science” because it lets us “understand the natural world in a new way, and is rapidly invading other disciplines, not merely as a way to do calculations, but as a whole new way of thinking”. If we accept these arguments, then the basics of computer science may, in combination with motivation through digital technologies, develop scientific thinking in all pupils, analogous to other sciences.

The question when to start teaching the basics of computer science can be answered as follows: “There is a strong analogy with the other sciences. We take it for granted that every student should learn the elementary concepts of (say) Physics at primary school and Key Stage 3, ... Exactly the same pattern should apply to Computer Science.” [4] However, the basics of computer science are not included in the official curricula documents for primary school education in the Czech Republic. E.g. algorithms are mostly included in mathematics, in ICT the documents only state that “the pupil should use algorithmic thinking in his/her interaction with a computer” [1]. The approach in the UK is very different: “the Government is now encouraging every good school to offer Computer Science as part of their curriculum, from primary

school onwards.” [4]. Topics from computer science are much more solidly anchored in Slovak primary schools as well, since “informatics at primary ... schools consists of the following five collective thematic units: 1. Information around us, 2. Communication through digital technologies, 3. Procedures, problem solving, algorithmic thinking, 4. Principles of the functioning of digital technologies, 5. The information society.” [5] Two of these topics focus on the basics of computer science.



**Fig. 1.** The cover of the new textbook Informatics for primary school

One of the important international activities promoting a higher share of computer science at primary and secondary school levels outside the school curricula is the Bebras contest (Beaver of Informatics) for primary and secondary school pupils [6]. The history of this contest dates back to 2004 and it has been held in the Czech Republic since 2008. The topicality of this contest is clearly manifested by the growing number of children and young people as well as countries participating in the contest. And this despite the fact that the contest does not focus on information technologies and user approaches. It prefers questions and topics from the basics of computer science, i.e. algorithmization, understanding information and its representations, and also mathematical and logical foundations of computer science. The problems which are compulsory for each state have not included questions about the use and role of ICT in everyday life during the last few years. The contest is gradually spreading to younger children. Slovakia opened the category for 8-year-old pupils in 2011. The Czech Republic opened the contest to primary school pupils in 2012.

### 3 Innovation in ICT teaching at primary school level

With the aim of accommodating both trends described above, namely the extension of teaching computer science to lower age groups and the emphasis on computer science topics, we wrote a textbook and a teachers' book and prepared a set of teaching materials for ICT education at primary school level (see Fig. 1). [7] This textbook includes topics not only of the basics of computer use, but also topics of propaedeutics to computer sciences connected with algorithmization and the understanding of information. The reason for dealing with these topics was to try and influence the teachers and to broaden their perspectives in the discipline. We bore in mind Rohann's assertion that "Primary school teachers, who are educated to teach a wide variety of subjects, will therefore need a thorough understanding of the subject matter of technology to know which topics to address and how to address them in their technology lessons." [8].

The textbook of Informatics for primary schools includes 8 chapters:

- 4 chapters from the area of ICT on running applications, drawing, writing and Internet (including communication)
- 2 chapters from the area of introduction to computer science (programming, information)
- an introductory chapter for very young pupils, manipulation with mouse, work on keyboard etc. (see Fig. 2)
- a concluding chapter on computer- assisted learning and on projects with the use of technologies.

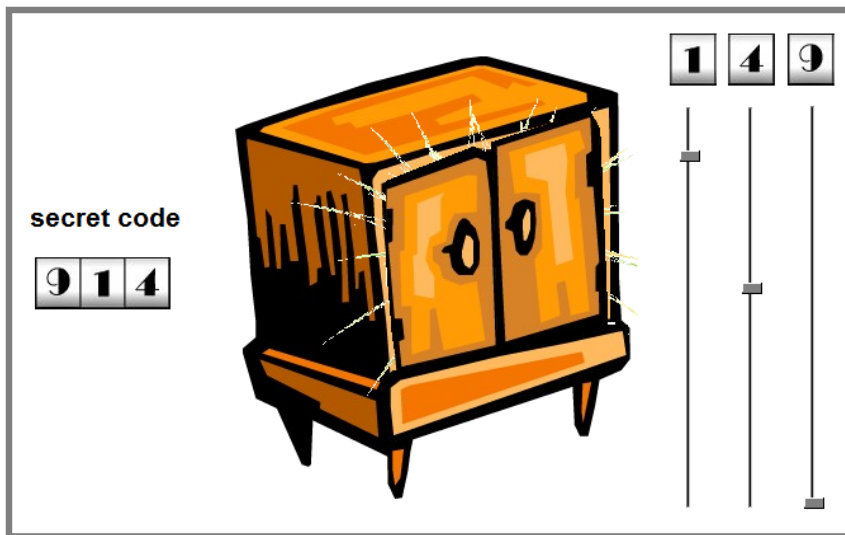
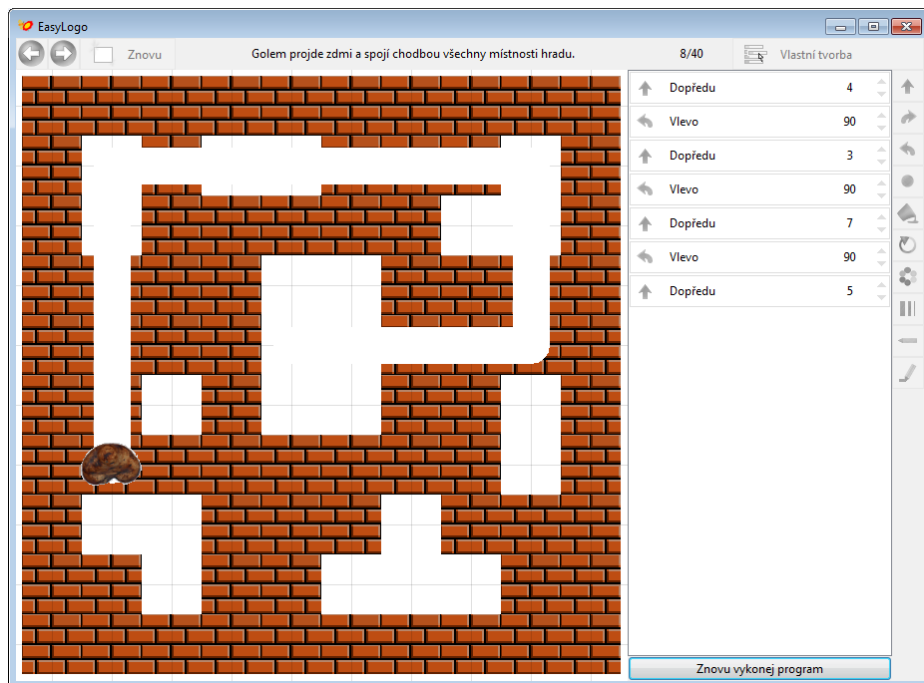


Fig. 2. "Set the input code to the treasure by dragging the sliders", the activity for beginners.

Let us now introduce the two "computer science" chapters in more detail. The chapter on programming uses the application EasyLogo, whose author is Ľubomír Salanci [9],

a simple microworld based on Logo, in which the user controls a figure that, analogously to the “logo-like turtle”, moves and draws. Apart from giving direct control the application offers a system of creating programmes as a sequence of commands with integer parameters, a loop of a known number of repetitions, a simple application of procedures.



**Fig. 3.** Programming activity for beginners in EasyLogo, creating tunnel in castle walls to join all chambers using turtle drawing commands

The main advantage of this environment is a set of programming exercises for the pupil, implemented into the environment of the application. Each task has a goal and tools for reaching it, which makes it possible to guide the pupil methodologically from simpler to more difficult algorithmic problems. The pupils may work at their own pace and the computer gives them full feedback. They learn new concepts in accordance with the constructivist paradigm, in creation, manipulation, in activities appropriate for their personality, temperament and age. The teacher is in the role of manager, an assistant rather than the person assigning the problems or explaining the subject matter. The closed learning environment of these tasks gives the teacher a feeling of security, and this, according to Slovak teachers, is much appreciated.

We created 40 problems for the environment EasyLogo (see Fig. 3, 4), some of which are based on the Slovak version of the application. Using the turtle graphics pupils draw pictures, move figures, solve problems, read and correct already written programmes. The chapter on programming is complemented by activities in which

pupils create commands for their classmates to carry out some task and by algorithmic questions with multiple-choice answers.

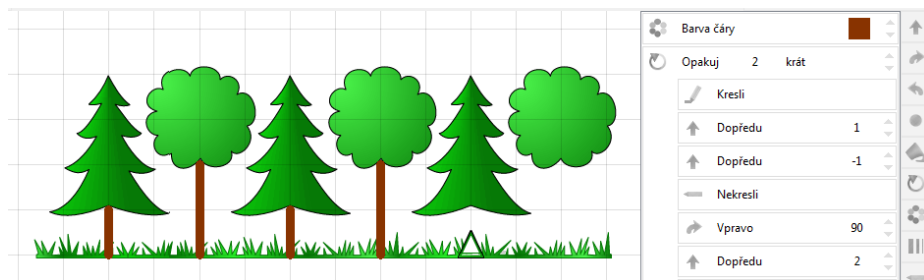


Fig. 4. Advanced programming activity in EasyLogo, drawing tree trunks using a loop

The chapter on information deals with information from the point of view of its content (what do the things around us tell us), representation (signals, symbols) and types of digital media (text, audio, video). The chapter also focuses on representation of information in the form of lists, tables and graphs. Apart from user competencies (e.g. writing into a table, formatting the list, creating graphs from tables) pupils train competencies to understand these structures. They learn to distinguish between a numbered and an unnumbered list, to choose the format of a table for recording specific data, to choose the suitable type of list, to read information from graphs and to make a graph “manually” (see Fig. 5). When reading a graph, pupils are taught to pay attention to the graph’s information value. Pupils then make use of tables, charts and graphs with simple data processing in their own research (measuring, questionnaires, surveys). The chapter has been supplemented by web application Grafy online (Graphs online) [10] for simple conversion of data from a table or chart into a graph, and software activities in which a pupil creates graphs from tables “manually”.

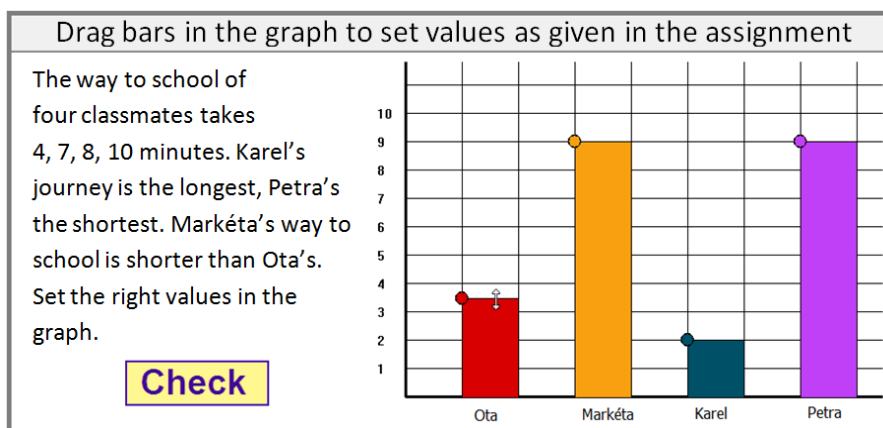


Fig. 5. A learning activity in which the pupil “manually” sets the values on the graph with the mouse to make them correspond with the text on the left (translated).



## **4 Verification among pre-service teachers**

Simultaneously with the process of writing the new textbook we started piloting the course Didactics of Information Technologies for pre-service primary school teachers at the Faculty of Education, University of South Bohemia. The one semester course consisted of a one-lesson lecture and a one-lesson lab-course a week. In the lectures the students were introduced to the different topics as described in the textbook, in the seminars they worked with the materials described in the paragraphs above.

We were interested in the following questions: how can computer science be made more familiar to pre-service teachers, who have no training in the field and how do they grasp topics in this discipline? Other questions were whether the tasks and problems prepared for the new textbook were comprehensible, whether students would grasp both the assignment and the informatics background.

Feedback from the students was gained by the method of participant observation, by analysis of students' work and by interviewing some of the students after the course was completed. The group participating in the research consisted of 42 students in the 6th semester of a long-cycle Master study programme for primary school teachers, 2 men and 40 women, at an average age of 22 with no prior teaching experience. The students had already been tested in information literacy (corresponding roughly to the level ECDL-START) and had finished the course Technology in Education, in which they had worked with audio, video and interactive board.

The teaching in the course was very practical. In the lab-course the students were presented with the more difficult problems from the textbook, in other words with problems of the level their future pupils would be expected to master. The lectures were devoted to the different topics from the textbook. The students, who were in the position of their prospective pupils, were presented with the teaching methods used in the textbook. In most cases the various methodological approaches were documented or illustrated on problems and activities from the textbook, in some cases on activities the students had the chance of trying out in the role of pupils and getting hands-on experience. The students' task at the end of the semester was to create their own set of problems and activities for teaching one of the presented topics.

## **5 Findings**

### **5.1 General attitude**

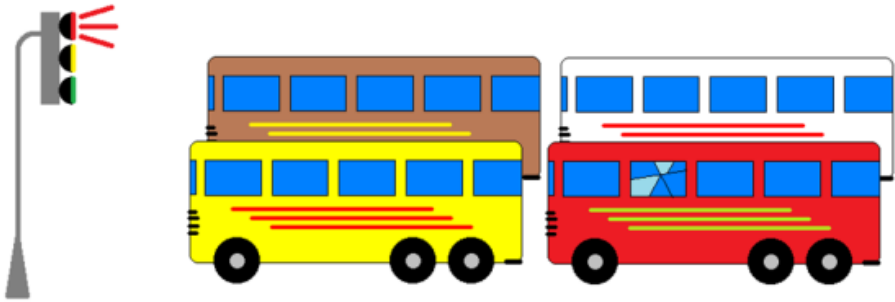
The students grew markedly more interested when they found out that this had become a compulsory subject in school curricula that they would have to teach as well.

In general, the students can be said to have regarded the presented teaching methods as beneficial, they were drawn into the playful, practical applications and problems from real life. In these cases everything seemed more comprehensible to them than in theoretical passages. As soon as the lecture brought some theoretical topic, the students tended to learn it off by heart with the aim of reproducing it, not of using it.

## 5.2 Programming

The students had no problem with the term programming, there was no apparent fear of programming. When solving programming problems the students perceived their activity as real programming. This particular group of students thus did not confirm the general prejudice against the term programming.

It can be said that roughly one half of the students were evidently enjoying their work in the environment EasyLogo. We could witness emotional reactions to the user-friendly environment and to situations when the students managed to solve some more difficult problem. In the final testing all the students were able to solve more difficult algorithmic problems using a loop (e.g. make a programme in which the turtle would finish drawing all tree trunks – see Fig. 4).



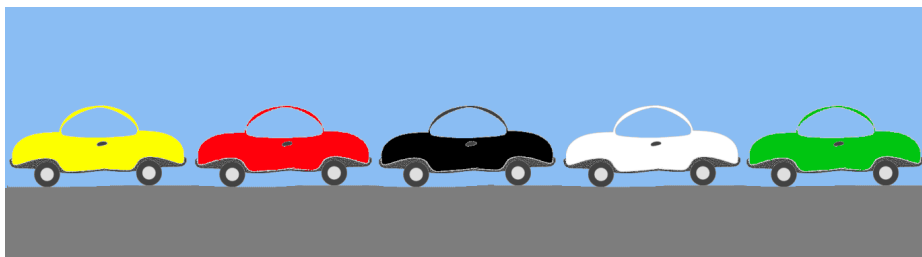
**Fig. 6.** Problems with spatial orientation in the problem assignment – is the bus with a broken window to the left or in front of the white bus?

The students were not very enthusiastic about using the loop, the structure of the command Repeat in case there were more commands included. It was an additional condition that the length of the programme must not exceed 15 lines in order to fit the programme into the window of the application that forced the students to use the command Repeat. The same applied to embedded procedures which were used only if the assignment clearly stated that it was compulsory to use pre-prepared subprocedures.

## 5.3 Problem of spatial orientation

The students' problems with spatial orientation – what is in the front, what is on the left etc – were striking. For example, the task which was originally formulated as “buses parked in a garage” (see Fig. 6) that were to be coloured according to the instruction based on their position was too difficult for some of the students who were not even able to determine whether (e.g.) the red bus was in front of the yellow bus or to the left of it. This experience made us reformulate the problem in the textbook to moving buses and to add traffic lights. The direction in which the bus is going defines the direction “in front” and “behind” more clearly.

A similar situation could be observed with a problem in which the students were to determine in a picture of a line of cars (that seemed exactly the same at the front and at the back) which car was driving in front of the black car (see Fig. 7). Some students insisted that “in front” is always on the right. This probably comes from the position of words in a written text. No arguments could persuade them that the cars could also be heading to the left. It seems that these problems are of a geometrical nature and have no connection to informatics. However, these unexpected problems may disqualify many problems from real life when aiming at ordering and classification.



**Fig. 7.** A “problem” for students: which car is in front of the black one?

#### **5.4 Work with information, graphs**

When creating graphs with the help of the application Grafy online, some students had problems when trying to enter the data correctly into the table from which the graph was generated. This appears to be user competence but is based on the understanding of a table as a data structure. The students entered the data into the wrong column designed for description of the items, and when this generated a wrong graph they were at a loss how to correct it.

#### **5.5 Interdisciplinarity, creativity**

In general, the students preferred problem-solving tasks, even in programming, to tasks in which they were required to create something new. The students’ own project work was not very imaginative and was often a mere variation to problems solved in the seminars.

The students did not differentiate between informatics and other school subjects. The problems posed by the students often involved knowledge from other disciplines. For example in word processing more attention was paid to grammar mistakes than to typographic errors, in drawing more attention was paid to the right use of drawing tools from the artistic rather than the ICT point of view. The students did not convince us that they would be able to create teaching objects for use in their teaching of informatics.

## 6 Conclusion

We tried to answer the question how to train didactically pre-service primary school teachers with no former training in informatics. Twyford and Järvinen claim that “In a qualitative study the ‘hands-on’ approach was again advocated for primary school pupils, but the researchers added that pupils’ understanding of technology can be best achieved when the presented problems are authentic” [11]. Our experience supports this claim. The useful approach seems to be to introduce pre-service primary school teachers to the basics of informatics by including topics and problems close to their pupils, in other words by including manipulative, practical problems accompanied by explanatory commentaries showing the relation between the problem and informatics.

Our sample of pre-service primary school teachers did not have troubles with algorithmic problems if they could understand the problems as controlling the movement of objects. Analogically, they easily and naturally accept the topic of understanding information and its representations if the particular problem seems generally beneficial to their pupils’ education. It can be concluded that they perceive the informatics part of education much more in the context of education as a whole than in the context of computer science.

If we want primary school teachers to include the basics of computer science in their curriculum of ICT correctly, they will have to have methodological guidance, either from teachers’ textbooks or from in-service seminars. In the offered course pre-service teachers paid more attention to attractiveness, playfulness, manipulative nature, usefulness of the activities than to disciplinary purity and correctness. These findings make us believe that additional training of primary school teachers in computer science would bring no significant effects in the direction of improving their teaching.

## References

1. Rámcový vzdělávací program pro základní vzdělávání. (Frame educational programme for primary and lower secondary education) [online], VÚP, Praha (2007) [http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPZV\\_2007-07.pdf](http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPZV_2007-07.pdf). (last checked 1/31/2013)
2. Tucker, A.: A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee. The Association for Computing Machinery, New York (2003).
3. OECD. Evolution of student interest in science and technology studies. Organisation for Economic Co-operation and Development, Global Science Forum, Paris (2006)
4. CAS: Computer Science as a school subject. Seizing the opportunity. Computing at School Working Group, (2012) <http://www.computingatschool.org.uk/data/uploads/Case%20for%20Computing.pdf> (last checked 1/31/2013)
5. Blaho, A., Salanci, E.: Informatics in Primary School: Principles and Experience. In Kalaš, I. and Mittermeir, R.T. (eds.) ISSEP 2011, pp. 129–142 (2011)
6. Dagienė, V.: Sustaining Informatics Education by Contests. In Hromkovič, J., Královič, R. and Vahrenhold, J. (eds.) ISSEP 2010, pp. 1–12 (2010)

7. Vaniček, J.: Informatika pro 1. stupeň ZŠ (Informatics for Primary School), textbook. Brno: Computer Press, 2012, 88, ISBN 978-80-251-3749-9
8. Rohaan, E.J., Taconis, R., Jochems, M.G.: Reviewing the relations between teachers' knowledge and pupils' attitude in the field of primary technology education. *International Journal of Technology and Design Education*, Vol. 20, No. 1, pp. 15–26 (2010)
9. Salanci, E.: EasyLogo – discovering basic programming concepts in a constructive manner. In Clayson, J. E., Kalaš I. (eds.) *Constructionist approaches to creative learning, thinking and education: lessons for the 21st century*. Bratislava: Library and publishing centre Comenius University (2010) ISBN 978-80-89186-65-5
10. Šimandl, V.: Grafy online, educational software [online]. České Budějovice: Jihočeská univerzita (2012) <http://graf.asp2.cz> (last checked 1/31/2013)
11. Twyford, J., Järvinen, E.M.: The formation of children's technology concepts: A study of what it means to do technology from a child's perspective. *Journal of Technology Education*, Vol. 12, No. 1, pp. 32–48 (2000)



# Environments for programming in primary education

Monika Gujberová and Peter Tomcsányi

Department of Informatics Education, Faculty of Mathematics, Physics and Informatics,  
Comenius University, 842 48 Bratislava, Slovakia

{gujberova, tomcsanyi}@fmph.uniba.sk

**Abstract.** The aim of our article is to collect and present information about contemporary programming environments that are suitable for primary education. We studied the ways they implement (or do not implement) some programming concepts, the ways programs are represented and built in order to support young and novice programmers, as well as their suitability to allow different forms of sharing the results of pupils' work. We present not only a short description of each considered environment and the taxonomy in the form of a table, but also our understanding and opinions on how and why the environments implement the same concepts and ideas in different ways and which concepts and ideas seem to be important to the creators of such environments.

**Keywords:** Primary informatics, Programming environments for children, Comparing programming environments

## 1 Introduction

At the beginning of our work there was an idea to look at the programming environments used in Slovak schools from a technical point of view, from the point of view of a developer of such environments rather than from the point of view of their users. Our interest was based on our previous experience with developing such environments. We wanted to know the ways they implement (or do not implement) some programming concepts, the ways programs are represented and built in order to support young and novice programmers, as well as their ability to enable different forms of sharing the results of pupils' work.

The second section shortly describes the considered environments that were chosen based mainly on our knowledge of their widespread use in Slovak schools.

The third section shortly describes how we chose our initial set of criteria and how we iteratively enhanced them. Then it presents our final list of features as well as the knowledge that we gained in the process of studying the environments. Finally we present a grid of features for the selected ten environments in a form of a table.

## 2 The Environments

For our study, we chose at first eight environments, knowing from our contact with schools and our in-service courses for primary teachers that they are actually wide-

spread in Slovak schools (sections 2.1 and 2.2). Later we added two environments that are not yet commonly used in Slovakia, but we consider them worth comparing to the other environments as well as to present them to our teachers (section 2.3).

Our choice includes not only worldwide well-known environments (like Scratch or Alice), but also less known ones (like Karel or Baltie) and others specific to our country (like The Jumper or Phillip the Ant).

## 2.1 Logo descendants

Logo has been developed not only for children, but also for teachers [1]. First implementations were text-based. The turtle came later, at first as a physical robot turtle, later as the on-screen light turtle [2].

**Imagine Logo** is a full implementation of Logo developed in Slovakia and used in several countries [3]. It supports pure textual programming, but parts of the programs (especially parameters of standard procedures) can be constructed by selecting from lists, and its visual objects can be constructed by direct manipulation.

**Scratch** is a well-known modern descendant of Logo created by the Lifelong Kindergarten Group at the MIT Media Lab [4]. It has been localised to Slovak [5]. It uses puzzle-like construction of programs. The program is constructed from jigsaw puzzle-like pieces containing commands and parameters. The pieces either fit together or do not fit together depending on their shape.

**EasyLogo (Izy Logo** in Slovak language) [6] is a simplified turtle graphics program that has been implemented in a form accessible to pupils in primary education. It limits the turns to multiples of 45 degrees and redefines the metrics of the screen so that `fd 1` always moves to the next grid point. The programs are constructed by dragging the icons of actions into the program. The parameters are set by choosing them from lists. It allows defining one's own new commands without parameters.

**Lively picture** is another simplified version of Logo. It has been implemented in Imagine Logo. It focuses on a widespread simple Logo activity – constructing *lively pictures*, which are stories or animations. They contain static or moving graphical objects. Some of them may react to user interaction (clicking or dragging by mouse). Its programming language is purely iconic, only the parameters of some commands are expressed in numbers.

## 2.2 Programming specialised robots

**Karel 3D for Win32** is the most widespread implementation in Slovakia. Karel was originally developed as an introductory programming language for university students [7]. The robot can move and sense walls. It can put and collect marks (beepers), in newer implementations it can also put and collect bricks. The programs are written in text form, it includes procedures. It has no variables. Karel became popular in Czechoslovakia in the late 80s and it has been re-implemented for several kinds of computers.

**Baltie (Baltík** in Czech and Slovak) is an iconic programming language that originated in Czech Republic and is quite popular in the Czech Republic as well as in Slo-



vakia. The main character is a young magician Baltie, who can move around a 2D scene and conjure objects (square pictures). There are three modes of operation: creating the scene, direct mode conjuring and programming.

**The Jumper** [8] can jump up and down, step left and right and can repeat a list of statements. The pupils construct programs that lead the Jumper from its starting position to the doors leading to the next level by clicking on icons representing commands. The program must first be fully constructed and only then can it be run to see whether it is correct. The number of statements is limited so that the pupil must recognise repeated actions.

**Phillip the Ant** has to find a path through a maze, collect candies, avoid spider nets, collect other useful things and move things to their correct places. Phillip can be controlled in three modes: direct mode, arrows mode and iconic mode. The environment has been created in Imagine Logo as part of a master thesis [9], later extended for the use in in-service training courses in Slovakia [8].

### 2.3 Advanced environments

**Microsoft Kodu Game Lab (Kodu)** [10] has been designed for creating and playing one's own games. The whole program is event-based – it is a sequence of *When something happens do something* statements. The conditions and commands are expressed by icons selected from menus (repositories). The Web page of Kodu contains many tutorials that help to learn its handling.

**Alice 3D** is a programming environment that allows creating animations, interactive stories, videos and games. Originally it has been developed for teaching the basics of programming especially targeting pupils with no pre-knowledge in programming or little interest in maths. Another goal was to attract more girls to programming [11].

## 3 The features

We started to observe the ten environments using the criteria derived from the extensive work of Kelleher and Pausch [12], even if the goal of their work has been somewhat different from ours. Then we iterated these three steps: (1) studying the environments and coding their properties according to the existing set of criteria, (2) gaining knowledge about the properties of the environments, and (3) reflecting the gained knowledge by modifying the set of criteria to better meet our specific goals. This section presents the final set of criteria. We describe them in detail and give examples of the environments that meet or do not meet the specific criteria. Table 1 presents the complete grid of criteria and environments.

### 3.1 General vs. Specialised

Some of the considered environments are strictly *specialised* microworlds, like The Jumper and Phillip the Ant. Other environments include *general* programming lan-

guages like Imagine Logo, Scratch, Kodu and Alice. In primary education both these types of environment are useful for the teacher even if they serve different purposes.

Specialised environments usually include some number of prepared tasks and it is possible for the teacher (or even for pupils) to prepare their own tasks within the bounds of the environment. Sometimes it is possible to do this inside the environment in an interactive and visual way. We coded such environments as *internal* in the row *creating tasks*. Sometimes an external program is needed (usually a plain text editor), coded as *external*.

### 3.2 Programming style

For our purposes the programming style is the way of thinking while programming. The considered environments use three programming styles: *procedural*, *object-oriented* and *event-based*. The procedural style uses lists of statements that are executed from the beginning to the end. Most of them also allow the users to define their own procedures and/or functions. The *object-oriented* style uses the abstraction of object which is very common in professional programming environments. The environments for children often include partially implemented object-orientation, which we code as the *level of object orientation* in a next criterion.

The *event-based* style allows defining snippets of code that run when an event happens. Typical events are mouse clicks and drags, keyboard key pushes/releases or collisions of objects.

### 3.3 Level of object orientation

Blaho and Kalaš [3] defined a hierarchy of several levels of using object-oriented features. For our purposes we slightly modified their hierarchy as follows:

- *objects* – existence of objects as entities that encapsulate data (internal state) and procedures (methods) that can change the data
- *cloning* – a command or a user interface action that allows the user to create an identical copy of a complete object
- *own variables* – the users can define their own state variables in objects
- *own methods* – the users can define their own methods in objects that may override the standard behaviour of the object
- *inheritance* – one object can serve as a parent of another object that inherits all the behaviour of its parent (and may override some behaviour by defining its own methods)
- *classes* – special entities that serve only as parents (or prototypes) for creating objects (instances of the class) and themselves are not valid objects
- *multiple inheritance* – one object can inherit behaviour from several parent objects or classes.

One more criterion is outside the hierarchy – whether the programming language allows to *programmatically create and destroy* objects or instances of classes. Kodu and Imagine allow it while Alice and Scratch do not.

### 3.4 Programming Constructs

- *conditional* (if statements)
- *count loop* – a loop with a known number of repetitions but without an explicit loop variable (like *repeat* in Logo without using the *repcount* function)
- *for* loop
- *while* loop
- *variables* (global, local or object variables)
- *own procedures or methods*
- *parameters* in own procedures or methods

### 3.5 Code Representation

We consider the representation being *textual* only if it can be fully edited letter by letter. The other two representations are *card or puzzle-like* and *iconic*. The former still uses text to represent the program, but parts of the text are written on either rectangular cards or puzzle-like pieces, while the latter uses a purely graphical way of expressing the commands (even if sometimes it still needs numbers).

Some environments use two representations. EasyLogo represents the basic actions (commands) as icons that the user must drag into the program, but after one such icon is dragged, it changes into a textual card that shows the name of the command.

### 3.6 Project Construction

Modern programming environments for children tend to minimise typing or try to abandon it completely. However, an interesting research study suggests that it is not easy to predict whether these kinds of simplification will really make the learning of programming easier or not [13]. We distinguish six types of project construction:

- *typing* code
- *selecting* pieces of code from menus or similar user interface elements
- assembling code from *rectangular cards*
- assembling from *puzzle-like pieces*
- assembling from *icons*
- *direct manipulation* for creating (visual) objects

Imagine Logo allows typing, but specific parts of the code can also be selected from menus and special helper dialogs (choosers). Additionally objects (e. g. turtles) can be created by direct manipulation, too. Karel allows constructing the program only by typing, but the environment of the robot can be created by direct manipulation. Scratch and Alice construct code by assembling from pieces.

### 3.7 Preventing (syntax) Errors

Alternative ways of program representation and/or construction can usually prevent syntax errors. Sometimes also semantic errors can be avoided, e.g. Scratch does not allow the user to place a number in a place where a Boolean expression belongs.

We distinguish three approaches: *shape matching* (the shapes of pieces must match, otherwise a part of the program cannot be constructed), *selection from valid options* (a list of valid options is presented as a menu, pop-up window, a pile of selectable cards etc.), and *syntax directed editing*.

Syntax directed editing recognises the structure of the program and allows the user to drag/select only complete structures (e. g. complete *if-then-else* command or *while* loop), thus not allowing the user to make errors by forgetting a part of the command.

### 3.8 Saving and Exporting

Being able to show the results of their work to others is an important point for children. Some environments are able to export the whole project or at least its result in a form that can be run or at least shown, using only standard software with no need to install the environment itself. Sharing the result of one's work on the Web is another important feature. We distinguish the following features of the environments:

- *own format* – whether it is possible to save the project into a file that can be later loaded by the same environment and used further
- *result* – whether the environment can save only the resulting static picture or series of pictures (video or animation) that can be shown later using a player of a standard file format without the ability to continue the work on the project
- *standalone project* – whether it is possible to create a standalone executable file that can be run without the need of having the whole environment available
- *Web site* – whether the creators of the program provide a Web site to share the programs created by pupils
- *save for Web* – whether it is possible to save the resulting program in a form that can be run inside a browser, in which case we list the additional software that is needed (except the browser itself) to run it.

### 3.9 Localization

In primary schools it is very important to present the computer programs to pupils using their native language. Several environments on our list originate from Slovakia, other environments can be localised *by the end-users* (Scratch and Kodu), and for some others several localisations already exist. In our table we also present the *number of localizations* that we know about.

**Table 1.** The resulting grid of features and environments

		Imagine Logo	EasyLogo	Lively picture	Scratch	Karel 3D for Win32	Balite 3	The Jumper	Flip the Ant	Kodu	Alice 2.2
General vs. Specialised	specialised way of defining the tasks		x	x		x	x	x	x		
			ext			int	int	ext	int		
Programming style	procedural	x	x			x	x	x	x		
	object-oriented	x								x	x
	event-based	x		x	x					x	x
Level of Object Orientation	objects	x	x	x	x					x	x
	cloning	x		x	x					x	
	own variables	x			x						
	own methods	x								x**	x
	inheritance	x									x
	classes	x								x	
	multiple inheritance	x									
Programming Constructs	programmatically create/destroy	x								x	
	conditional	x			x	x	x			x	x
	count loop	x	x	x	x	x	x	x			x
	for loop	x			x	x	x				x
	while loop	x			x	x	x				x
	variables	x			x	x	x			x	x
Code Representation	procedures/methods	x	x		x***	x	x				x
	parameters	x	x								x
	textual	x				x					
Project Construction	card or puzzle-like		x		x			x			x
	iconic		x	x			x	x	x	x	
	typing	x			x****	x					
	selecting	x	x	x	x			x		x	x
	rectangular cards										x
Preventing (syntax) Errors	puzzle-like pieces				x						
	icons		x	x			x	x	x	x	
	direct manipulation	x			x	x				x	
	shape matching				x						
Saving and Exporting	selection from valid options		x	x	x		x	x	x	x	x
	syntax directed editing		x	x	x					x	x
	own format	x	x	x	x	x	x		x	x	x
Localisation	result	anim. gif gif, jpg	wmf		gif						mov jpeg
	standalone project	exe									
	Web site				x					x	
	save for Web	plugin									Java 3D
Localisation	by the end user				x					x	
	number of localisations		7	4	1	50	4	2	1	1	7
<b>Legend:</b>											
* Applies to Alice 2.3											
** Kodu has no real methods, but object can define their own reactions to events											
*** Only with an extension called BYOB (Bring your own blocks)											
**** Typing is very limited in Scratch – e.g. number and text data											

## 4 Conclusion

Our work started as a trial to make a systematic list of environments useful for primary education in our country. After a few iterations we found the result interesting not only for us, but also for a broader audience. The list itself shows the variability of the environments from specialised ones to general ones, from those inspired by Logo to others allowing the user to create complex videos, from textual programming through puzzle-like programming to iconic programming.

We believe that the information is valuable both to developers of such environments and their users. The developers get a new overview of all possible approaches used, while the teachers get acquainted with environments they did not yet know about or get more information about environments that they have already heard about.

## References

1. Feuerzeig, W. et al.: Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project: Bolt, Beranek and Newman, Inc., Cambridge, MA (1969)
2. Papert, S. et al.: Logo Philosophy and Implementation, LCSl (1999)
3. Blaho, A., Kalaš, I.: Object Metaphor Helps Create Simple Logo Projects. In Proceedings of EuroLogo 2001. Edited by G. Futschek. Linz, August. pp. 39–43 (2001)
4. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch Programming Language and Environment. In ACM Transactions on Computing Education, Vol. 10, No. 4, Article 16 (2010)
5. Drahošová, M.: Úvod do programovania v prostredí Scratch. Master thesis FMFI UK, Bratislava, Slovakia, (2010) [Introduction to programming in Scratch. Master thesis, Comenius University, Bratislava. In Slovak language]
6. Salanci, L.: EasyLogo – discovering basic programming concepts in a constructive manner. In: Proceedings Constructionism 2010, Paris (2010)
7. Pattis, R. E.: Karel – the robot, a gentle introduction to the art of programming. Wiley, London (1981)
8. Tomcsányiová, M. a kol.: Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika – Riešenie problémov a základy programovania 1. 1. Vydanie (2009) [Problem solving and programming basics, Textbook for an in-service teacher training course. In Slovak language]
9. Ondková, J.: Detský programovací jazyk Mravec pre 1. stupeň ZŠ. Diplomová práca. Bratislava: FMFI UK (2006) [Children's programming language The Ant for the primary school. Master thesis, Comenius University, Bratislava. In Slovak language]
10. [http://kodu.blob.core.windows.net/kodu/Curriculum\\_MARS/Level%20%20-%20Search%20and%20Explore%20Mars.pdf](http://kodu.blob.core.windows.net/kodu/Curriculum_MARS/Level%20%20-%20Search%20and%20Explore%20Mars.pdf) (last checked 1/31/2013)
11. Utting, I. et al.: Alice, greenfoot and scratch – A discussion. [online] ACM Transactions on Computing Education, Vol. 10, No. 4, Article 17 (2010) <http://www.cs.kent.ac.uk/pubs/2010/3071/content.pdf> (last checked 1/31/2013)
12. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37(2), pp. 88–137, (2005)
13. Lewis, C. M., “How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch”, Proc. SIGCSE’10, ACM Press, WI, USA, (2010)

## **Towards National Curricula**





# A comparison of current trends within Computer Science teaching in school in Germany and the UK

Valentina Dagiene<sup>1</sup>, Tatjana Jevsikova<sup>1</sup>, Carsten Schulte<sup>2</sup>,  
Sue Sentance<sup>3</sup> and Neena Thota<sup>4</sup>

<sup>1</sup>Vilnius University, Naugarduko Str. 24, 03225 Vilnius, Lithuania  
{valentina.dagiene, tatjana.jevsikova}@mii.vu.lt

<sup>2</sup>Freie Universität Berlin, Königin-Luise Str. 24, 14195 Berlin, Germany  
schulte@inf.fu-berlin.de

<sup>3</sup>Anglia Ruskin University, Department of Education, Chelmsford, Essex CM1 1SQ, UK  
sue.sentance@anglia.ac.uk

<sup>4</sup>UpCERG, Uppsala University, Uppsala, Sweden. University of Saint Joseph, Macau, S.A.R.  
neenathota@usj.edu.mo

**Abstract.** In the last two years, CS as a school subject has gained a lot of attention worldwide, although different countries have differing approaches to and experiences of introducing CS in schools. This paper reports on a study comparing current trends in CS at school, with a major focus on two countries, Germany and UK. A survey was carried out of a number of teaching professionals and experts from the UK and Germany with regard to the content and delivery of CS in school. An analysis of the quantitative data reveals a difference in foci in the two countries; putting this into the context of curricular developments we are able to offer interpretations of these trends and suggest ways in which curricula in CS at school should be moving forward.

**Keywords:** CS Ed Research, ICT, CS at school, CS curriculum, topics, international comparison, international study

## 1 Introduction

The second decade of the new millennium has started with growing attention to Computer Science (CS), Informatics or Computing in school education. Many countries are becoming inspired by the challenges posed by the Computer Science Teacher Association in USA [17] and Computing At School in the UK [3]. Readers can gain an impression of this recent attention by just observing the titles of articles of the third volume of the “ACM Inroads” 2012 publication: ‘Transforming High School Computing’; ‘Reforming K-12 Computer Science Education...what will your story be?’. All of these papers, and there are more, are devoted to the questions of CS education in schools.

In 2011, the annual international Koli Calling conference on computing education research held a workshop group on teaching computing at school and computing teacher education. Before the workshop, a group of researchers created an online

survey, and gathered information about the state of the art and current activities regarding teaching CS at high school and CS teacher education in multiple countries. The aim of this research was to get an overview of CS promotion at general education, to develop a vision and/or identify trends, and to offer suggestions for future work. Experts and teaching professionals were asked for their opinions on what was happening in their own country at that time, given that the pace of change is fast [15].

It is necessary to mention briefly the terminology used in different countries. Many European countries have titled the subject Informatics as equivalent to CS in USA. Recently the Royal Society in the UK has identified that the term ‘Computing’ should be broader than the ICT course and include CS: “*Every child should have the opportunity to learn Computing at school, including exposure to Computer Science as a rigorous academic discipline*” ([18], p. 6). Michal Armoni wrote: “*CS should be taught as another different stand-alone subject (under the title CS, computing, informatics or any other similar in nature) with a stand-alone curriculum that introduces CS as a discipline of its own*” [1] (p. 20). In this paper we will use the term Computer Science (CS) to refer to this academic discipline and it is equivalent to Informatics (Informatik) used in Germany.

## 2 Methodology

The data presented here is a subset of the data obtained by an international working group on the situation of CS as subject in schools. Experts from the international community of CS education and teaching professionals worldwide, were asked to answer an online questionnaire [15].

The call to fill in the survey was sent by personal email to experts known from relevant conferences. In addition, the call was sent to the mailing lists of the ACM SIGCSE (Special Interest Group on CS Education), the Computing at School forum (CAS) in UK, the IOI (International Olympiad in Informatics), and to the Computing Education Researchers (FG DDI) list in Germany.

Participants were instructed to answer from a national perspective, e.g. describing situations, problems and developments of CS education at school on a national or at least state-wide or province level. For all questions and scales, respondents were able to leave some questions unanswered. In the introduction to the questionnaire, participants were informed that they could leave out questions that did not refer to their personal knowledge or experience. This was done in order to avoid participants having to provide superficial information or referring to a local situation instead of the national one. We expected many experts to be aware of some aspects of the national system, but probably not about e.g. each type of school/age level. The results reflect this assumption, as the data has many unanswered questions.

Experts from 22 countries answered the questionnaire (Table 1). There were primarily up to 3 responses from most countries; however, from the UK, Germany and USA there were more respondents.

Part of the questionnaire addressed topics and goals covered in CS Education in three age ranges (primary education, lower secondary education, and upper secondary

education), as well as teaching methods. In each of these parts the participants were asked to rate the importance of different aspects of CS at school on a 5-point Likert scale. It is the responses to this section of the questionnaire, and what it reveals about the differences between nations, that we are addressing in this paper.

**Table 1.** Number of respondents\*

Nation	AUS	AUT	BGR	CAN	CHE	CZE	DEU	DNK	FIN	FRA	GBR
No	1	3	2	3	2	1	12	1	3	1	20
Nation	ISR	LTU	LVA	NLD	NZL	POL	PRT	SVN	SWE	UKR	USA
No	3	2	3	5	2	2	5	1	2	1	8

\* We use three-letter country codes defined in ISO 3166-1.

The questions used for goals, topics and methods were designed in a small group, and then sent out to a group of experts in several countries. In this pilot, only some changes were needed. The most disagreement was on age ranges, which were finally defined as follows: 1) Primary School (Elementary School; grades 1-4/7 students are typically 5 to 10 years old); 2) Lower Secondary (Middle School; grades 5-9/10, students are typically 10 to 15 years old); 3) Upper Secondary (Secondary School; grades 9/10 to 12/13, students are typically 15 to 18 years old).

### 3 Data analysis

In this section we present data from the questions described above from two countries, UK and Germany. As can be seen in table 1, there are also other countries with some more answers. E.g. USA, but here we got only one answer for the quantitative questions. Similar was the situation for the Netherlands.

Due to the large number of unanswered questions, we use a qualitative description of the visualized quantitative data, assuming that the few experts from one country indeed were able to rate the situation of the country on the given scale. The data was visualized using bar charts showing mean values. Thus, differences in these ratings are considered as indicators for possible differences in the educational approaches between the countries. We do not consider these indications as the final results, but as themes we should take into account when discussing the current situation in these countries, as they might reveal different possibilities, developmental stages or just local/national differences.

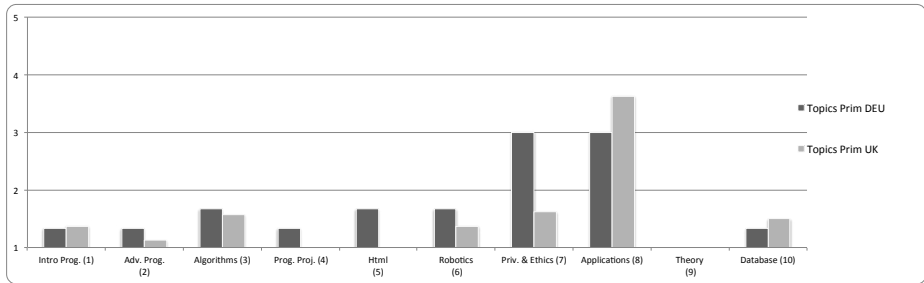
For more results on the study, see [15].

### 3.1 Topics

Questions were asked about the coverage of the following topics in schools:

1. Introductory programming (merely introduction to concepts, language, tools, etc.)	5. HTML
2. Advanced programming (merely programming in order to solve problems)	6. Robotics
3. Algorithms	7. Privacy & ethics
4. Programming project (full lifecycle projects, with e.g. requirements analysis, etc.)	8. Applications (e.g. text processing)
	9. Theory (e.g. automata)
	10. Database

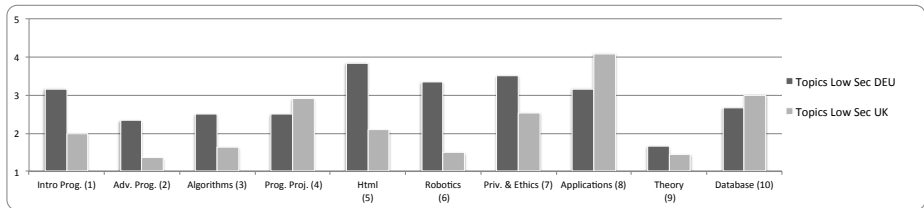
In primary schools, it can be seen (Fig. 2) that Germany and UK are covering broadly similar topics, with UK covering more with regard to applications than anything else. In Germany there is a strong focus on privacy and ethics, much higher than in the UK. Programming, HTML and robotics are also rated somewhat higher.



**Fig. 1.** Topics covered at primary school (3, DEU and 8, UK experts answered)

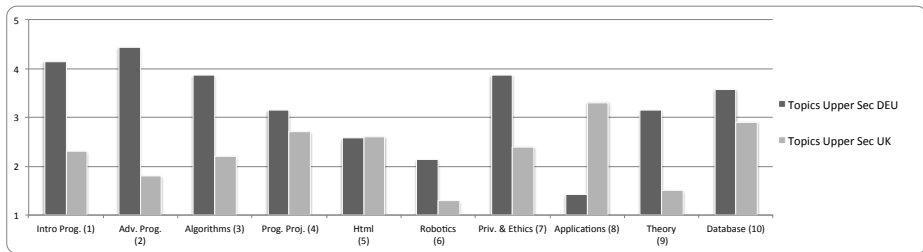
By *lower secondary* more of all these topics are covered in both countries (Fig. 2). However, the differences between the countries are more pronounced as we can see that Germany is including more programming and HTML, while UK still focuses mostly on applications, including now databases and projects.

In Germany, there is still a high focus on privacy and ethics, and also robotics. There is also a relatively high emphasis on applications.



**Fig. 2.** Topics covered at lower secondary school (6 and 11 experts answered)

By *upper secondary*, more of each topic is covered in both countries (Fig. 3). There is no difference with respect to the amount of HTML scripting covered, but Germany is showing a greater emphasis on computer programming and computational theory, whereas the UK still has its greatest emphasis on software applications, with databases the next greatest. In Germany, the focus on privacy and ethics again is very high, and there is some focus on this in the UK.



**Fig. 3.** Topics covered at upper secondary school (7 and 10 experts answered)

Results from other countries in our questionnaire reveal some differences with Germany and the UK data. For example, in Israeli primary schools there is more focus on Applications, HTML and Robotics. This difference is not observed in lower secondary school, except for Applications that are still slightly more focused on. In Israeli upper secondary school considerably more attention than in Germany is paid to Algorithms, Programming projects and Theory, and slightly more focus is on Advanced programming. This may be explained by the fact that Israeli upper secondary school has developed and implemented a strong CS curriculum with five courses: 1) Fundamentals (2 units, introducing the central concepts of solving algorithmic problems and teaching how to apply them in a programming language); 2) Software design (data structures and design of complete system); 3) Second paradigm (introduction to a second programming paradigm, logic programming, functional programming and system-level programming are three of the current possibilities); 4) Applications (focuses on a particular application, emphasizing both theory and practice; and 5) Theory (exposes to selected topics in theoretical CS, e.g. models of computation, finite automata) [10].

### 3.2 Teaching Methods

We asked questions about the choice of the following teaching methods:

1. Pupils work individually on projects	8. Using editors
2. Pupils work in small groups on projects	9. Programming
3. Pupils work individually or small groups on small tasks at the computer	10. Using integrated development environments (IDE)
4. Classroom based teaching	11. Projects
5. Discussions	12. Role plays
6. Reading	13. Lectures
7. Using standard applications (text processing, mail)	

Looking at the average responses with respect to teaching methods (Fig. 4) there are only slight differences, with more individual work being carried out by UK students and more group work in Germany. Students in Germany attended slightly more lectures and worked on programming tasks using IDEs and editors, whereas UK students used more standard applications. This reiterates the difference in topics alluded to above.

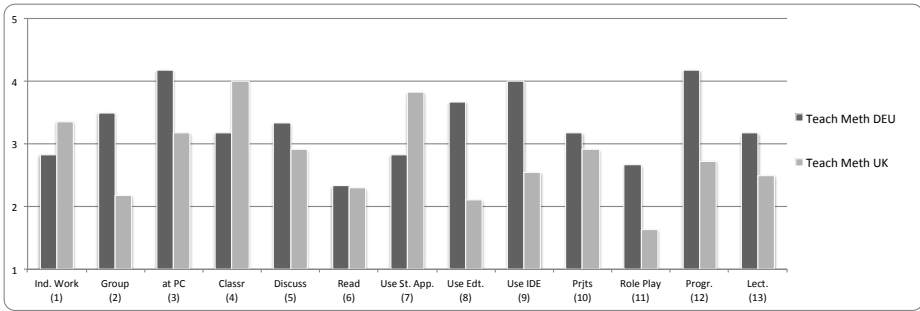


Fig. 4. Teaching methods used in different countries (6 and 11 experts answered)

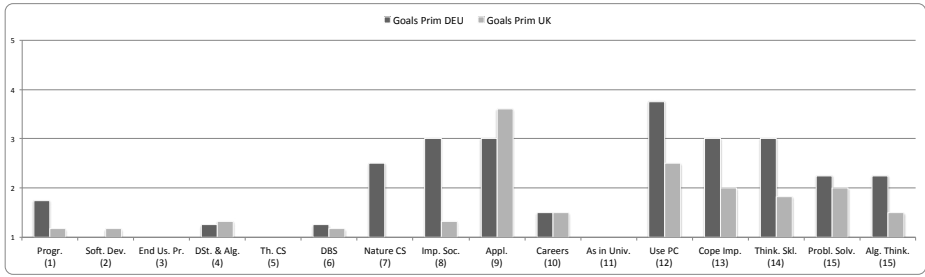
### 3.3 Goals

We asked questions about the goals of CS programs in schools:

1. Learning programming in the small	9. Mastering the important applications
2. Learning programming and software development process	10. Knowing careers and opportunities in CS
3. Learning end user programming (Macros, etc.)	11. Introducing CS as it is presented and conceptualized in Universities
4. Data structures and algorithms	12. Preparation of learners to use computers / digital technologies
5. Aspects of theoretical CS (e.g. halting problem; complexity)	13. Preparation of learners to cope with the impact of CS on everyday lives (e.g. political issues like privacy, e-Democracy)
6. Databases: design and queries	14. Developing thinking skills (logical reasoning, abstraction, etc.)
7. Understanding the nature of computer science	15. Problem solving skills
8. Understanding the impact / relationship of CS and the society	16. Algorithmic thinking

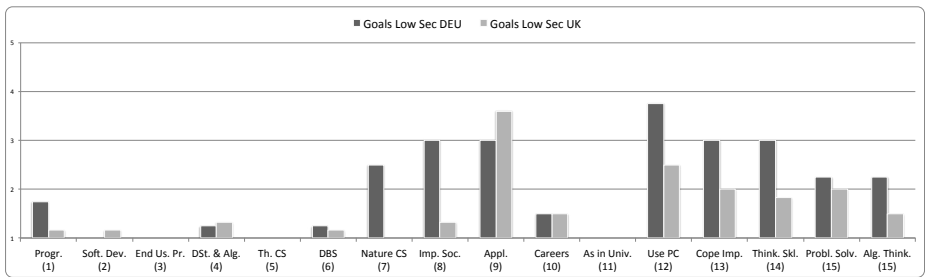
The goals that schools were working towards in terms of CS education in primary were again broadly similar, as for topics. There were some exceptions: in Germany, there was more emphasis on the nature of CS, its impact on society, personal computer use, and thinking skills (Fig. 5).

Interestingly, we found somewhat reversed ratings for Using PC and Applications. In addition, Impact on society as a goal the somewhat reversed ratings for Using PC and Applications. In addition to Impact on society as goal, also understanding Nature of CS was rated very high in Germany.



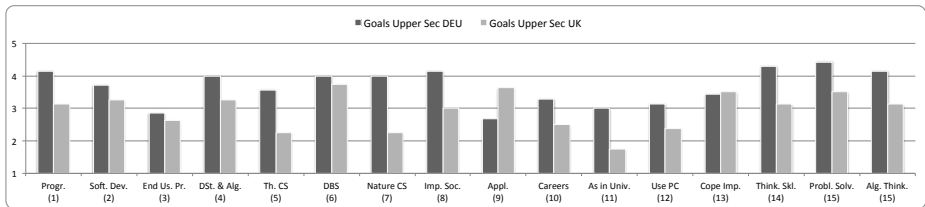
**Fig. 5.** Goals of lower primary education in CS (in UK and Germany) (4 and 6 experts answered)

By lower secondary, these differences are more marked (Fig. 6).



**Fig. 6.** Goals of lower secondary education in CS (4 and 6 experts answered)

By upper secondary, more UK teachers report that more software applications are taught in schools than German teachers (Fig. 7). However, Germany records more curriculum goals around problem solving, thinking skills, programming, and the nature of Computer Science.



**Fig. 7.** Goals of upper secondary education in CS (7 and 8 experts answered)

## 4 Differences in curricula

### 4.1 UK perspective

The data gathered very much reflects the situation in England and Wales in the middle of 2011, when the survey was carried out. A short history of UK teaching of ICT and CS will demonstrate emerging trends.

Prior to the 1980s, programming had been taught in schools and in the age of the first small personal computers such as the BBC Micro, all students learned about hardware, binary arithmetic, and computer programming. The UK was seen as a leading force in the development of personal computing in schools and many children were inspired and engaged. A qualification was available to 14–16 year olds in Computer Studies. However, in the 1980s, CS essentially disappeared as a curriculum subject for under-16 year olds, being replaced by Information & Communication Technology (ICT) [4]. The National Curriculum was introduced into England and Wales, which focused on developing skills in software packages as well as other ICT soft skills. The most recent version of the National Curriculum for ICT (2007), which was a compulsory subject in schools up to age 16, had a focus on using ICT appropriately with the right tools and being able to evaluate effective use of ICT; Scotland, however, managed to retain Computer Studies as a topic in schools at this time. CS was retained as a subject in its own right post-16, but the number of students choosing to take this was diminishing due to the lack of understanding of the difference between ICT and CS [3].

Although it had been seen as important to give students in the UK digital literacy skills through the 1990s and the 2000s, this was obviously at the expense of CS topics. By around 2008, there was a realisation, through the development of the Computing at School group (CAS) in the UK that this was unsatisfactory and could suggest a reason for declining recruitment figures to study CS in higher education. Concerns were expressed that ICT lessons were “boring” and did not engage or enthuse students as well as focusing too much on basic digital literacy. The Royal Society thus launched a consultation on the matter in 2010.

The Royal Society released its report entitled “Shut Down or Restart: The way forward for Computing in UK schools” at the end of its 18 month consultation period, describing the teaching of CS in many schools as “highly unsatisfactory” [18]. The recommendations included increasing the number of teachers trained to teach CS, improving in-service training, and providing more technical resources for schools. The Secretary of State for Education has subsequently announced that the National Curriculum for ICT is to be discontinued from September 2012. It will remain compulsory to teach ICT from ages 5 to 16, but schools no longer have a prescriptive program of study and are encouraged to include more CS.

The media, as well as industry and academics, have been quick to react to these comments, and there is now a rapid pace of change within England and Wales to a broader, more academic, curriculum, combining ICT and CS. There is an increased focus on reducing the amount of time spent learning to use software applications and bringing in introductory programming into the lower secondary stage. This includes



learning Scratch, Kodu and simple Python programming, in addition to the development of multimedia and animation skills that were formerly part of the lower secondary curriculum. We have now seen the introduction of several qualifications at the upper secondary stage that students can study from ages 14–16 in CS. The Computing at School group has had a very significant role to play in both advocating and implementing these developments.

## 4.2 Germany perspective

In order to understand what current developments imply, we need to discuss the history of CS education in Germany. CS started very early in Germany (in the 1970s). The first two decades were dominated by a succession of four didactical approaches to teaching CS at the school level (for a more in depth discussion see [6]).

The initial approach, with virtually no practical impact, was called hardware orientation [8], and based on a cybernetic philosophy, grounded in the understanding that digital technology is completing the technological evolution of mankind by entering the third and last stage. The first stage was the invention of tools to augment limbs (or: extremities) with tools (like a hammer). The second approach was the invention of machines to augment physical strength. Digital technologies are seen as the final stage, as they are augmenting intellectual power, by a process called *objectivation*, the segmentation of problems in a digital representation. Therefore, it was seen as necessary to teach basic principles of the hardware, and not to start using a higher order programming language, because using e.g. loops would prevent from focusing on and understanding the fundamental basic process of digitalization. At this point, a counter movement started and proposed algorithms as the central point of interest, and the need to start with a higher order programming language in order to teach algorithmic problem solving, and to introduce basic algorithms and algorithmic strategies (sorting, searching, abstract data types, recursion). Programming projects were introduced as a means to foster these learning goals.

The third approach at this point in history criticised the algorithmic approach for being somewhat detached from the use of CS in everyday life and from appropriately teaching the impact of CS. It was termed ‘use orientation’. Programming projects should show the impact or usefulness in real life, like digitizing the library, or the grocery store. Essential (but in practice often neglected) was a reflection on the impact of such a project on the users or society in general.

The last of the so-called classical approaches in CS school education in Germany strengthened this focus on users, and suggested to shorten programming, in order to gain time for discussing impact; because in the meantime (mid 80s) the personal computer was invented and a lot of applications were developed so that problem solving with computers was also possible by using software applications.

In 1986, national regulations introduced a mandatory subject for teaching these topics, but focusing on use of IT (ICT) in lower secondary school [7, 11]. An additional and optional subject called Informatics was also introduced for upper secondary school (and in lower secondary as well in some parts of Germany). In a way, the last approach of the classic views described above became ICT, whereas Informatics was

oriented towards algorithm-oriented approach and also to the third, more critical, phase.

In late 80s, Informatics was well established in school, but came into a crisis in the 90s [13], due to lower enrolments, lack of clarity of educational goals, and demands for change due to new programming paradigms.

There are now several newer approaches for conceptualizing CS as a subject in school in Germany, see e.g. [2, 5, 12, 16].

Current developments in Germany are focusing on improving the situation in lower secondary education, for which educational Standards (Bildungsstandards 2008) have been developed by the Gesellschaft für Informatik (GI), the German Computing Society. While most issues could be solved easily during development of standards, some issues have been the main focus of discussion [14]: A) Should there be a programming standard? Or should it be modelling, or modelling and implementing? Is programming a central topic (a content standards), or merely a practice, and thus a process standard? In the end, it became a process standard named Modelling and implementing. B) Is the topic of CS, man and society one associated topic (one content standard) or merely something that permeates every part of the curriculum? Or, when included as a process, does it face the danger of becoming implicit or even being left out of everyday teaching practice? In the end, it became a content standard.

Contextualized teaching is proposed as a model to foster the development of teaching units that implement and help to disseminate the standards (examples are teaching units on Chatbots as introduction to artificial intelligence, email as introduction to cryptography, or mobile phones as introduction to privacy and understanding information and data). Other examples currently being developed are focusing on including hardware, using some of the current and increasingly-popular hardware gadgets like Arduino, Scratch boards, Raspberry Pi or .Net Gadgeteer.

## 5 Discussion

It is apparent that the responses showed some differences in the topics taught at school between Germany and the UK. It would appear that both countries have a similar focus in primary school, but that the difference is quite significant by the time students reach age 14. In primary school, most teaching focuses on the use of ICT and software applications, although there is a strong focus on ethics also in primary school in Germany. There are other themes that emerge from the data, where we can look at differences and similarities in relation to the way CS is taught in school. We will discuss these themes in more detail in the following sub-sections.

**Ethics in Computing.** In Germany, there is a focus on privacy and ethics from primary to upper secondary as this has been a focus within the curriculum for several decades. In the UK, ethical issues are discussed with pupils at all levels of their education, but mostly within the context of internet safety and areas such as data protection. This appears within the ICT curriculum so it is unusual that the UK experts did not feel that this was a significant topic that had been covered.

**ICT versus CS.** When looking at the full range of topics from primary to upper secondary, there is some progression from using computers and application – ICT – to programming, database, algorithms, to CS.

In the UK, throughout the stages, there is a focus on application, hence ICT, is maintained. In Israel, in contrast, the upper secondary conceptualization seems very closely related to CS as taught at the university level.

It can be seen from our data that coverage of ICT and CS topics differs in different countries, according to the national perspective. We surmise that different countries can learn from the good practices of each other.

**The role of programming.** Again, there are differences that can be noted between countries. In Israel, the curriculum contains a strong focus on programming, advanced programming, data structures and algorithms, and on programming projects, according both to our data and the literature. In Israel, programming projects are likely to be done by individual work [9], whereas in Germany, programming projects are almost always done in groups, or within the whole class. The UK seems to report less involvement with programming projects until reaching the upper secondary stage.

The current trend for primary and lower secondary education, in many countries, is to introduce visual environments such as Scratch and Kodu that make programming fun and accessible to younger children. This is appearing within the curriculum in both UK and Germany, but probably this trend occurred earlier in Germany. However in some countries, there still is the problem that there are not enough teachers who are fully trained to teach CS. The danger might be that the teacher does not have the understanding of the programming concepts underlying the environment to be able to facilitate the development of basic programming principles and might still perceive using such tools as learning to use an application as opposed to learning the principles of programming.

**Understanding the nature of CS.** This topic is essential if we are going to give children a head start in CS. However, the data does not yet give us the evidence that this is uniformly introduced to children in school. In upper secondary there is more teaching of understanding the nature of CS according to the data, but this may be in non-compulsory courses that small numbers of children sign up for (we do not have data for this).

**Identifying trends in CS at school.** A longitudinal study would give firm evidence of trends in the teaching of CS in schools. We will be able to further our understanding of the way that CS is delivered in schools in different countries by repeating this data collection exercise at a later date. However, the evidence from published curricula and professional associations indicates that there are trends towards teaching more CS in school and having less focus on digital literacy. If these trends really do exist, can we suppose that in a few years' time all countries may be covering the same topics within the broad area of CS? Although we have noted differences between the UK and Germany throughout the study, the changing curricula suggests that these differences are already diminishing.

On the other hand, we might see different traditions or cultures with regard to CS education: In Israel, we see a strong focus on CS as it is conceptualized and taught in academia. In Germany, this focus is true for the algorithmic oriented approach from the 80s, but was widened afterwards. Now the focus is – according to our data – on the societal impact of CS, and on ‘nature of CS’ – teaching a suitable conceptualization of CS – which might become necessary due to a weaker link to the academic image of CS. In the UK, the focus is on applications, probably due to the history where ICT was the central subject and was only recently supplemented by CS. It will be interesting to see how the development proceeds: Towards the Israeli model, or towards the German model – or towards a unique integration of ICT (and applications) into a CS curriculum.

Further work will include using the questionnaire to collect data from school teachers and experts in Asia about the content of CS education there.

## 6 Conclusion

The data suggests that at the time of the survey Germany has a broader focus on Computer Science in its schools than the UK. The UK data shows that many topics are covered, but there is an increased emphasis on teaching students to use software applications. This disparity is starting to be addressed within the UK. Although Germany has managed to have a continuity of teaching Computer Science (as Informatics) within the school curriculum throughout the history of digital technologies, the UK is starting to catch up.

The situation in the UK is rapidly changing. Teachers are now trying to adapt rapidly to these changes and in-service and appropriate pre-service training of CS teachers for England, Wales and Northern Ireland is needed (Scotland already has CS within the curriculum). The intention is that establishing a broad and balanced curriculum, we can use experiences from a range of other countries. In addition, through these experiences, we can develop teacher education programs and new curricula that will be of wide interest.

## 7 References

1. Armoni, M.: The nature of CS in K–12 curricula: the roots of confusion. *ACM Inroads* 2, pp. 19–20 (2011)
2. Breier, N, Hubwieser, P.: An information-oriented approach to informatical education. *Informatics in Education*, vol. 1, pp. 31–42 (2002)
3. CAS Working Group: A Curriculum Framework for Computer Science and Information Technology, vol. 8 (2012)
4. Crick, T., Sentance, S.: Computing at school: stimulating computing education in the UK. In: *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. ACM, New York, NY, USA, pp. 122–123 (2011)
5. Euler, D.: *Didaktik einer sozio-informationstechnischen Bildung*. Botermann & Botermann, Köln (1994)

6. Forneck, H-J.: Bildung im informationstechnischen Zeitalter: Untersuchung der fachdidaktischen Entwicklung der informationstechnischen Bildung. Sauerländer, Aarau (1992)
7. Forschungsförderung B-L-K für B und Gesamtkonzept für die informationstechnische Bildung. Springer, Bonn (1987)
8. Frank, HG., Meyer, I.: Rechnerkunde: Elemente der digitalen Nachrichtenverarbeitung und ihrer Fachdidaktik. In: Frank H., Meder B.S. (eds.) *Kybernetische Pädagogik Schriften 1958-1972. Band 5* Kohlhammer, Stuttgart, pp. 585–774 (1974)
9. Haberman, B., Cohen, A.: A high-school programme in software engineering. *International J of Engineering Education*, vol. 23, no. 1, pp. 15–23 (2007)
10. Hazzan, O., Gal-Ezer, J., Blum, L.: A model for high school computer science education: the four key elements that make it! *SIGCSE Bull* 40, pp. 281–285 (2008)
11. van Lück, W.: Informations- und kommunikationstechnische Grundbildung und der Computer als Medium im Fachunterricht. Soester Verlagskontor, Soest (1986)
12. Magenheimer, J., Schulte, C.: Social, ethical and technical issues in informatics – An integrated approach. *Education and Information Technologies* 11, pp. 319–339 (2006)
13. Peschke, R.: Die Krise des Informatikunterrichts in den neunziger Jahren. In: Stetter F., Brauer W. (eds.) *Zukunftsperspektiven der Informatik für Schule und Ausbildung: GI-Fachtagung "Informatik und Schule" 1989*. Springer, Berlin, Heidelberg u.a., pp. 89–98 (1989)
14. Schulte, C., Saeli, M.: Applying standards to computer science education. In: Kadjevich D., Angeli C., Schulte C. (eds.) *Improving Computer Science Education*. Routledge, pp. 117–131 (2013)
15. Schulte, C., Dagiene, V., et al.: Computer Science at school/CS teacher education. In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. ACM, New York, NY, USA, pp. 29–38 (2012)
16. Schwill, A.: Fundamental Ideas: Rethinking Computer Science Education. *Learning & Leading with Technology*, vol. 25, no. 1, pp. 28–31 (1997)
17. Seehorn, D., Carey, S., Fuschetto, B., et al: *CSTA K–12 Computer Science Standards Revised 2011*. 65 (2011)
18. The Royal Society. Shut down or restart? The way forward for computing in UK schools. The Royal Society (2012)



# Informatics Education in Turkey: National ICT Curriculum and Teacher Training at Elementary Level

Yasemin Gülbahar<sup>1</sup>, Mustafa İlkhani<sup>2</sup>, Selcan Kilis<sup>3</sup> and Okan Arslan<sup>3</sup>

<sup>1</sup> Ankara University, Department of Informatics, Ankara, Turkey  
gulbahar@ankara.edu.tr

<sup>2</sup> Ministry of Education, General Directorate of Innovation and Educational Technologies,  
Ankara, Turkey

milkhani@gmail.com

<sup>3</sup> Middle East Technical University, Department of Computer Education and Instructional  
Technologies, Ankara, Turkey  
{skilis, okana}@metu.edu.tr

**Abstract.** This article is a summary of the work carried out by the Ministry of Education in Turkey, in terms of the development of a new ICT Curriculum, together with the e-Training of teachers who will play an important role in the forthcoming pilot study. Based on recent literature on the topic, the article starts by introducing the “F@tih Project”, a national project that aims to effectively integrate technology into schools. After assessing teachers’ and students’ ICT competencies, as defined internationally, the review continues with the proposed model for the e-training of teachers. Summarizing the process of development of the new ICT curriculum, researchers underline key points of the curriculum such as dimensions, levels and competencies. Then teachers’ e-training approaches, together with selected tools, are explained in line with the importance and stages of action research that will be used throughout the pilot implementation of the curriculum and e-training process.

**Keywords:** informatics education, ICT curriculum, teacher training

## 1 Introduction

The term of “informatics”, inferred from “information” and “automatics” has its origin in Europe, but it is not in common use in other countries of the world. ICT is the mutual term in most European countries; in Austria it is called “informatics” [23]. If ICT is one side of the medal, informatics is the other side. As discussed at the ISSEP conference in March 2005 in Klagenfurt (<http://issep.uni-klu.ac.at>), the topic of “Does ICT eat or feed informatics” did not lead to a clear conclusion. As [23] stated, the borderlines cannot be drawn exactly, but they are fairly indistinct and floating. In our country the term ICT Education is perceived as Informatics Education, which aims to develop students’ computer literacy, technology awareness, computer usage and problem solving skills.

National development depends on investment in human resources, and a nation’s future correlates with the education system training these resources. The F@tih Project<sup>4</sup> is

<sup>4</sup> F@tih Project: <http://fatihprojesi.meb.gov.tr>

a transformation movement at a national level, having various and important outcomes, particularly the ICT curriculum and teachers training in ICT competencies. These are the two burning topics in terms of the project's success, as defined at a recent project evaluation workshop. Existing curricula, e-content and teaching approaches are inadequate in terms of what the project aims to achieve. What is more, not only for the F@tih project, but also to help Turkey strive towards reaching the goal of an information society, the ICT curriculum should be redesigned. In the information society, each student should be provided with the opportunity of taking advantage of technology [3]. ICT will be used as a tool for learning and for shaping the future of each student if the students are provided with the appropriate technologies and approaches at the right age.

## **2 Literature Review**

### **2.1 About the F@tih Project and EBA**

In the F@tih project, ICT tools and resources will be used to address inadequacies in the learning and teaching processes for 620,000 primary and secondary-education students. The aim is to increase learning and teaching opportunities and to enhance the schools by using ICT tools and resources. For all classrooms LCD interactive smart boards and tablet PCs will be provided as well as a purpose-designed network infrastructure. In-service training will be provided for teachers to enable them to use and adapt ICT tools and resources efficiently within their teaching and learning processes. The five main components of the project are:

1. Providing hardware and software infrastructure
2. Providing and managing educational e-content
3. Ensuring the effective use of ICT in the curriculum
4. In-service teacher training
5. Deliberative, safe, manageable and measurable use of ICT

The project is being carried out by the Ministry of National Education, supported by the Ministry of Transport, and is planned to be completed in 5 years.

For supporting the project, a web portal named EBA (Eğitim Bilişim Ağı (tr) – Education and ICT Network) is designed in order to address both teachers' and students' needs. In EBA, which is in test use now, there are eight main services, namely: (1) News, (2) e-Content, (3) e-Book, (4) Video, (5) Audio, (6) Images, (7) Forum, and (8) Map. In the News section information about project process, new progresses and some announcements is provided. The e-Content section contains shared work of universities and schools. In the e-Book section sample interactive course books are provided. The Video section offers videos about course contents, documentaries, and interesting scientific topics. The Audio section does not only cover courses, but also social life, including poems and stories, in order to help students improve their competence. The Archive of the Ministry of Education provides a big variety of photographs on course components to students and teachers. In the Forum part students, teachers and other stakeholders of the project can ask each other questions and carry out discussions. In the Map section there is a world map. Students can use it to find a city, region or country they want to learn about. Although a great variety of electronic content is available to students and teachers, EBA is the main umbrella that provides trusted shared content.



## 2.2 ICT Competencies for Students and Teachers

With the continuous emergence of new technological developments and innovations in the 21<sup>st</sup> century, today's teachers need to be better prepared and consider new approaches in order to bring information and communication technology skills into their classrooms. According to the UNESCO ICT Competency Standards for Teachers, they should acquire the skills and standards to empower their students with the benefits technology offers, should teach the subject effectively integrating technology and support the personal development of students [33]. In addition, teachers should have a vision of integrating technology into teaching and learning, they should develop an understanding of and capability with ICT, understand how to detect and develop students' ICT capabilities, teach effectively using technology and manage learning environments safely [2].

Not only teachers, but also students should increase their ICT capabilities. In general, ICT competencies for students can be summarized under main headings as: creativity and innovation, communication and collaboration, research and information fluency, critical thinking, problem solving, decision making, digital citizenship, and technology operations and concepts respectively [35]. For information literacy learners should assess the information process and learn to use tools like online catalogues and keyword search strategies; in terms of communication and collaboration they should work on a collaborative project, use different media to collect information; for authoring, they should publish a piece of original work and comply with copyright conventions; for organizational tools they should use the features of a network and work with spread sheets and databases; finally for presentation and visual display, they should develop and deliver a presentation and a display [6].

## 2.3 Teacher Training

In order to enhance knowledge about digital technology in teacher training and to enable teachers to integrate information and communication technology into educational settings and to become conversant with the use of the Internet and ethical issues concerning the web, a collaborate approach is adopted and online education is preferred in most countries in the world [21]. Most of the training programs, as in Korea, are designed as self-directed and self-paced [19]. For more efficient training, smaller-scale training program sessions were held in smaller regions, as in 25 areas throughout Taiwan [5]. To support this online education via website, sometimes face-to-face meetings and conferences are held. In general, video conferencing, Web 2.0 tools, webinars and online discussions are the tools preferred most [28]. As [18] stated, online discussions help to build a community of practice and also reduce the sense of loneliness.

In teacher training projects across many countries teachers are able to send, upload and share files, photos, etc. with their colleagues. In order to motivate teachers and increase participation in training, some reinforcement techniques are used such as issuing certificates or offering training at graduate schools. Using the module of EBA, sharing and collaboration address the needs of teachers in terms of experience, good examples, administrative issues etc. The Distance Education Center (DEC), which is a part of EBA, has been established to reach teachers indifferent regions of Turkey. In order to

train teachers, three different perspectives of adult learning are being used according to teachers' needs and status. To keep teachers updated all the time, a live broadcast called "ICT Talks" is put into practice. In this broadcast different burning topics, such as internet and ethics, Web 2.0 tools and technology integration strategies are discussed briefly once a week. In addition to this, webinars about trends will be held in order to enhance teachers' knowledge about technology integration. Briefly, it became obvious that teachers are more interested and generally more active in these online training programs. On the other hand, problems are frequently encountered, such as technical issues, a lack of equipment or badly managed forum discussions. Periodically, feedback from the participants is collected, results are collated and evaluated and the training programs are adjusted accordingly [36].

### **3 National ICT Curriculum**

The existing ICT curriculum, which is used to facilitate information and communication technologies in schools, is limited as to the technologies and software provided to learners. Today, however, we have unlimited choices and resources available for meeting our expectations. ICT should be used effectively and efficiently in order to overcome possible obstacles in teaching-learning processes and add value to societies' existing structures and cultures [27]. From a technological point of view learning in digital environments, augmented reality, mobile learning and social networks are visions of BECTA [3] as future dimensions in technology.

Hence, a new curriculum should consist of important concepts such as digital literacy, technology use, ethical considerations, security, privacy, programming concepts and cybercrimes from the perspective of both the individual and society. In other words, the main goal of the new curriculum is to build up a "new culture" about the use of technology rather than teach the usage of specific software based on a constructivist and cognitivist point of view with a learner-centered approach. Moreover, technology usage should be integrated into all other courses [9].

In order to meet the demands of the information age it was a necessity to create a new ICT curriculum. Defining ICT standards expected from learners, as knowledge, skills and values, is of great importance for defining teacher competencies and learners achievement levels [31]. Creating standards is an effective and important way of learning, since expectations are defined clearly [24, 29]. Hence, a standards-based curriculum approach was preferred and a framework was established based on the international standards of ICT [25, 26, 30], which was composed of four dimensions: (1) Digital literacy, (2) Communication, Knowledge Sharing and Self-Expression via ICT, (3) Research, Knowledge Construction and Collaboration and (4) Problem Solving, Programming and Development of Authentic Materials. The general goal of the standards-based curriculum was formulated as: "learners are expected to use information and communication technologies effectively, efficiently and in parallel with the ethical values". The official name of the course is specified as "Information and Communication Technologies and Software".

On the other hand, it is important to consider both cognitive and technical competencies in the curriculum development process. Digital literacy is achieved through

the use of digital technologies, communication tools and social networks in the process of accessing, managing, designing, evaluating and creating information by means of cognitive and technical knowledge, skills and values through hands-on experience [8]. Moreover, based on the fact that learners will have different prior knowledge on the topic, each learner should be provided with an individual instructional design according to her/his background knowledge.

Mainly three levels were formed with two dimensions in each level, based on some taxonomies and levels defined by various researchers as can be seen in Table 1 [14, 32, 34].

**Table 1.** ICT levels for learners used as a guide while specifying learning outcomes

<b>Level</b>	<b>Explanation</b>
Basic I	Understanding ICT
Basic II	Accessing and Evaluating information
Intermediate I	Managing information
Intermediate II	Transforming information
Advanced I	Creating information
Advanced II	Sharing information

While implementing the curriculum, teachers will find out the level of each learner and will try to improve it for each learner individually or for the group. During the implementation of the program, teachers are free to decide what they will teach and how. The key point for teachers here is to choose up-to-date topics and activities. The main goal is to create a culture about ethical usage of technology and to enhance the competence of each student. Students completing all the levels will be directed to bigger projects and competitions.

As for the teaching-learning process, measurement and evaluation within the course will also be learner-centered. Hence, alternative assessment approaches are preferred for evaluating the process and product from an authentic point of view. For providing teachers with the tool of evaluation, fulfilling the premise of learner-centeredness together with constructivist and cognitivist approaches, e-portfolio assessment is chosen as the evaluation method for this course [16]. This approach will not only enhance contributions to a national content development process by the use of national products like EBA, Kirk Ambar etc., but will also make learners learn by doing.

#### **4 Training Teachers at a Distance: Expected Competencies and e-Learning Approach**

Effective use of technology during the implementation of the F@tih Project and the implementation of the new ICT curriculum require teachers to learn and master different competencies. To this aim, teachers' ICT competencies were basically grouped

under headings as: skills and practice, knowledge and understanding, and values and attributes. To achieve these competencies teachers need training, and the only way to reach all the teachers in a short space of time is to use the Internet and web-based technologies, i.e. through e-learning. Keeping in mind the differences that exist in schools, in terms of technical infrastructure, prior knowledge, learner competencies and expectations, sustainable e-training of the teachers seems to be the best solution. Gaible and Burns [15] also recommended to give teachers enough training and support for an effective integration of the technology, to make them adopt applications in learning environments quickly and to ensure sustainability of the training process by considering three different approaches: (1) Content Delivery System (for gaining expertise and pedagogical knowledge), (2) Application (for creating opportunities to experience the technology) and (3) Diffusion of new innovations (disseminating successful samples and learner-centered teaching approaches). Hence, an open-source learning management system together with the products for sharing and communicating like Web 2.0 tools were used to form an “e-Training Portal” for teachers.

It is imperative to use information and communication technologies for teacher training in this information age when the advantages are considered. Applicable approaches in this teaching-learning process can be grouped as: (1) Individual learning, (2) e-Content and (3) Communities of practice [15]. Hence, any e-Training portal should at least be providing these three approaches for enriching the learning process and addressing possible individual differences.

Keeping all these facts in mind, an “e-Training Portal for Teachers” has been designed and developed (Fig. 1).

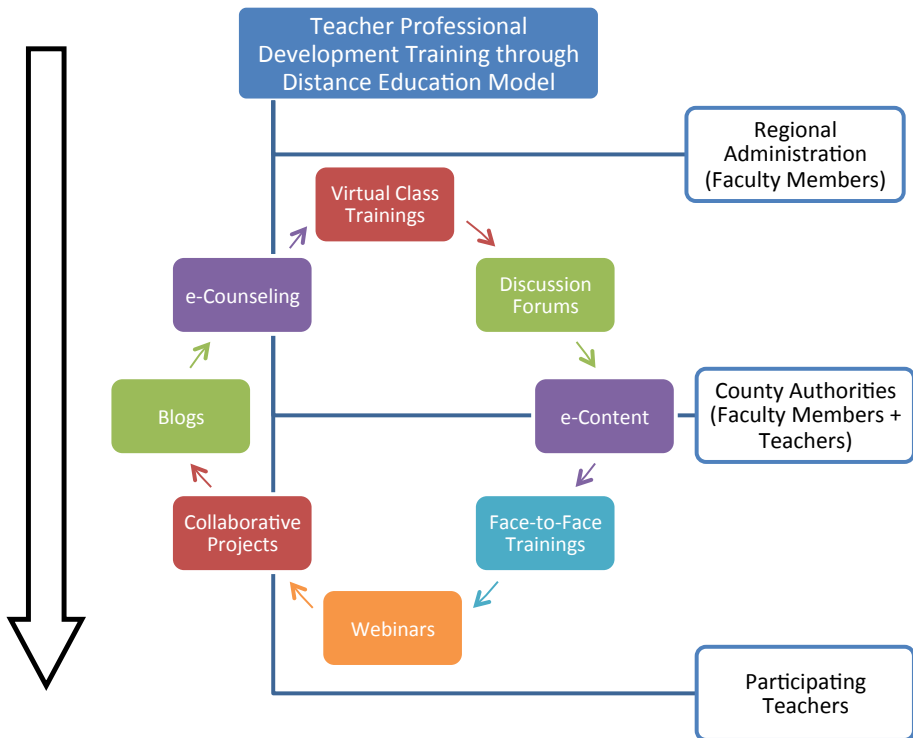
## **5 Collaboration and Research Model for Enhancing Quality**

Action research is a collaborative activity among colleagues searching for solutions to real life problems experienced in schools, or seeking for ways to enhance instruction and increase student achievement [13]. The aim of action research is to enhance and improve the professional development and the application of the user in daily life environments [7]. In action research the process teacher can search for ways to improve students’ learning or focus on his/her own method that he/she uses [17]. From this point of view action research is a special research method that lets teachers and other educators enhance and improve teaching practices [7]. Generally action research

1. helps teachers to recognize the problems about education issues and solve them,
2. may create web site based communities for practice and
3. helps to enhance and improve teacher capacity in terms of collaboration and support.

As [20] (1998) stated that action research is constructed by eight phases which are based on three main key questions. These phases can be seen in Fig. 2.

1. Define current situation (Phase 1, 2, 3,4)
2. Decide what to change and implement (Phase 5)
3. Search on effects of changes (Phase 6, 7, 8)

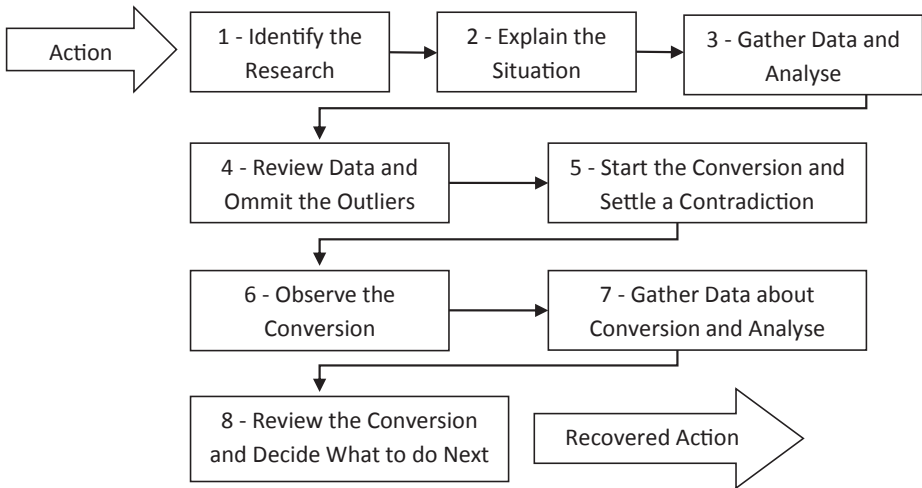


**Fig. 1.** Teacher Professional Development Training through Distance Education Model

In light of this information it is determined that action research is the most suitable method for a collaboration and research model. Within the scope of action research, the project is planned to be evaluated in terms of planning and coordination, in-service training, teachers' and students' issues in different phases of the project. Information and the thoughts of stakeholders, who are at different levels of the project, will be gathered by action research, which encompasses both qualitative and quantitative methods. The action research will be cyclical, repeated after each semester. After phase eight of an action research, according to the previous cycle's findings, it will be repeated in terms of project efficiency and success in the following semester.

### 5.1 Planning & Coordination

Activity reports, questionnaires about in-service training and teachers, interviews with project executives and managers are the basic instruments of collecting data. Comparing plans and outcomes of the project, identifying the possible mistakes and finding or suggesting solutions according to the situations are the fundamental aims of the planning and coordination phase in order to address project success and sustainability.



**Fig. 2.** Eight Phases of Action Research [20]

## 5.2 In-service Training

In-service training is one of the most important key points of the project. Plans, implementation and quality of the in-service training are the main issues of this phase in terms of the project's success. Thanks to technology integration, teachers are supposed to not only improve pedagogically, but also improve their high order skills in using ICT tools and sources. In order to accomplish this phase, teachers may be surveyed about their satisfaction and ICT usage and its integration into education. In addition, at the end of the semester, teachers' lesson plans could be analyzed in order to understand whether or not the information and skills from the in-service training have been put into action.

## 5.3 Teachers

Teachers, who are one of the largest stakeholders of the F@tih project, are in need of high order technology and pedagogical information such as how to implement ICT in education and how to address their students' needs. A survey could be conducted in order to evaluate teachers regarding ICT usage in education, different types of instructional methods, their attitudes to foreseen or real problems and problem-solving strategies.

## 5.4 Students

To understand whether or not the F@tih project succeeds for students, the grades of students both before and after the project could be compared. Besides, based on the objectives of the new curriculum, portfolios which will be prepared by students during the F@tih project could be analyzed in order to understand whether or not students gain from the objectives.

## 5.5 Discussion and Conclusion

Although there are many projects around the world regarding technology integration into education, Turkey has decided to start by equipping her schools with the appropriate technology apparatus. For this purpose, a variety of e-content software, tutorials, encyclopedias, animations, simulations, games etc. has been either developed or purchased for students, for use both within and outside of the classroom. Moreover, for teachers to become used to smart boards, a lot of e-content has been developed to enrich the interactive use of the devices. On the other hand, on-going face-to-face training is being conducted with the users of these new technological devices. In general, students are never given training but teachers are provided with short periods of training several times, but are still needing more. E-training modules and portals have been developed to help maximize the experience of teachers acquiring these competences.

In the autumn semester, all of these new ideas and approaches will be implemented in schools selected as pilot institutions. After that, based on action research results, the necessary improvements and re-implementations will continue.

It is believed that in the presentation the data for the first phase of research, at least, will be shared with the participants. The research results are expected to reveal both the successes of the e-training program and the ICT curriculum. Findings will be used to redesign or improve the existing structures for reaching the goal of effective integration.

## References

1. Baron, G.L.: ICT competencies, for Students and Teachers: Dilemmas, Paradoxes and Perspectives – The French Case (n.d.)
2. BECTA: 21st Century Teacher: Are you Ready to Meet the Challenge? (2010) <http://unesdoc.unesco.org/images/0015/001562/156207e.pdf> (last checked 1/31/2013)
3. BECTA: Harnessing Technology, Next Generation Learning 2008–2014: A Summary. Department for Innovation, Universities & Skills (2008)
4. Bogdan, R.C., Biklen, S.K.: Qualitative Research for Education: An Introduction to Theories and Methods. Pearson Education Inc., USA (2007)
5. Chou, C., Peng, H.: Promoting awareness of Internet safety in Taiwan in-service teacher education: A ten-year experience. *Internet and Higher Education*. 14, 44–53 (2011)
6. Crock, C.: ICT Competencies for Students: Mandated / Meaningful Conundrum. Presented at the Australian Computers in Education Conference (2002)
7. Çardak, Ç.S.: The Investigations of Students' Interactions and Learning Levels at Blended Learning (2012)
8. Educational Testing Service (ETS): i-Skills: Digital Transformation A Framework for ICT Literacy. , International ICT Literacy Panel (2007)
9. European Commission: Key Data on Learning and Innovation through ICT at School in Europe. Education, Audiovisual and Culture Executive Agency (EACEA P9 Eurydice) (2011)
10. FATİH Project: About the Project, [http://FATİHprojesi.meb.gov.tr/tr/icerikincele.php?id=6](http://FATIHprojesi.meb.gov.tr/tr/icerikincele.php?id=6) (last checked 1/31/2013)
11. FATİH Project: Project Components, <http://FATİHprojesi.meb.gov.tr/tr/icerikincele.php?id=14> (last checked 1/31/2013)
12. FATİH Project: Project Executive Board, <http://FATİHprojesi.meb.gov.tr/tr/icerikincele.php?id=8> (last checked 1/31/2013)

13. Ferrance, E.: Themes in Education: Action Research. Brown University, Northeast and Islands Regional Educational Laboratory At Brown University (2000)
14. Fraillon, J., Ainley, J.: The IEA International Study of Computer and Information Literacy (ICILS). International Association for Evaluation of Educational Achievement (2011)
15. Gaible, E., Burns, M.: Using Technology to Train Teachers: Appropriate Uses of ICT for Teacher Professional Development in Developing Countries. InfoDev / World Bank, Washington DC (2005)
16. Gülbahar, Y.: Usage of Electronic Portfolios for Assessment. Handbook of Research on New Media Literacy at the K-12 Level: Issues and Challenges. pp. 702–719. Information Science Reference, New York (2009)
17. Hennricks, C.: Improving Schools through Action Research: A Comprehensive Guide for Educators. Pearson, Boston (2006)
18. Hramiak, A.: Online Learning Community Development with Teachers as a Means of Enhancing Initial Teacher Training. *Technology, Pedagogy and Education*. 19, 47–62 (2010)
19. Jung, I.: Issues and Challenges of Providing Online Inservice Teacher Training: Korea's Experience. *International Review of Research in Open and Distance Learning*. 2 (2001)
20. Köklü, N.: Eğitim Eylem Araştırması (Tr) – Educational Action Research (2001)
21. Marti, M.C.: Teacher Training in ICT-Based Learning Settings: Design and Implementation of an On-Line Instructional Model for English Language Teachers (2006)
22. Mills, G.E.: Action Research: A Guide for the Teacher Researcher. Pearson Education Inc., New Jersey (2003)
23. Michuez, P.: Is it Computer Literacy, IT, ICT or Informatics? Education for the 21st Century — Impact of ICT and Digital Resources. 210, 369-373 (2006)
24. Moseley, L.: The role of assessment (2012)
25. NAACE: Draft Naace Curriculum Framework Information and Communication Technology (ICT) Key Stage 3 (2012)
26. NAACE: ICT Framework: A Structured Approach to ICT in Curriculum and Assessment (Revised Framework) (2007)
27. Nutt, J.: Professional educators and the evolving role of ICT in schools: Perspective report. CfBT Education Trust (2010)
28. Sharpe, L., Hu, C., Crawford, L., Gopinathan, S., Khine, M.S., Moo, S.N., Wong, A.: Enhancing Multipoint Desktop Video Conferencing (MDVC) With Lesson Video Clips: Recent Developments in Pre-service Teaching in Singapore. *Teaching and Teacher Education*. 19, 529–541 (2003)
29. Steiner, J.: Why have a standard-based curriculum and what are the implications for the teaching-learning-assessment process? *English Teacher's Journal* (2012)
30. The International Society for Technology in Education (ISTE), National Educational Technology Standards for Students, <http://www.iste.org/standards/nets-for-students> (last checked 1/31/2013)
31. Thomas, L.G., Knezek, D.G.: Information, Communications, and Educational Technology Standards for Students, Teachers, and School Leaders. *International Handbook of Information Technology in Primary and Secondary Education*. pp. 233–348. Springer US, USA (2008)
32. Tomei, L.A.: Taxonomy for the Technology Domain. Information Science Publishing, USA (2005)
33. UNESCO: ICT Competency Standards for Teachers. United Kingdom (2008)
34. UNESCO Bangkok: Strategy Framework for Promoting ICT Literacy in the Asia-Pacific Region. UNESCO Bangkok Communication and Information Unit, Bangkok (2008)
35. XINMIN Science Academy: ICT Competency Standards for Students (2008) <http://www.xinminsec.edu.sg/ict/joomla15/index.php?view=article&id=47> (last checked 1/31/2013)



36. Zhang, L.J.: Reforming a Teacher Education Program for PRC EFL Teachers in Singapore: Sociocultural Considerations and Curriculum Evolution. *International Journal of Educational Reform*. 13 (2004)



# The new Course of Study and a prospect of information studies education in Japan

Yoshiaki Nakano<sup>1</sup> and Katsunobu Izutsu<sup>2</sup>

<sup>1</sup> Osaka Electro-Communication University, JAPAN  
info@nakano.ac

<sup>2</sup> Hokkaido University of Education, JAPAN  
idutsu@gmail.com

**Abstract.** Japan launched the new Course of Study in April 2012, which has been carried out in elementary schools and junior high schools. It will also be implemented in senior high schools from April 2013. This article presents an overview of the information studies education in the new Course of Study for K-12. Besides, the authors point out what role experts of informatics and information studies education should play in the general education centered around information studies that is meant to help people of the nation to lead an active, powerful, and flexible life until the satisfying end.

**Keywords:** General subject “Information”, Course of Study, Scientific understanding of Information, Information Ethics

## 1 Three objectives of information studies education in primary and secondary education

The information studies education in Japanese primary and secondary education encourages pupils and students to develop their “Practical skills in using information actively”, “Scientific understanding of information”, and “Positive attitudes towards today’s information-laden society”.

The three objectives were defined in the first report of “the Researchers Conference for Promotion of K-12 Information Studies Education”, in October 1997. Based on them, the previous Course of Study launched its version of information studies education. [1]

The three objectives are defined as follows:

“Practical skills in using information actively”: The ability to use information means to collect, evaluate, express, process, and/or create the information needed, and to dispatch and communicate it in consideration of the receiver.

“Scientific understanding of information”: Understanding the characteristics of information means to use information actively and to understand basic theories and methods for handling information appropriately and for evaluating and improving one’s own use of information.

“Positive attitudes towards today’s information-laden society”: The positive attitude that should be accompanied by the intention to understand the role and

influence of information in our life, to consider the importance of information ethics and one's own responsibility for information, and to participate actively in creating a desirable society.

## **2 Information studies education in elementary schools**

Elementary schools do not provide any specific subject of information studies but encourages pupils to develop their skills in using information actively by means of information devices like personal computers in class activities centered around the subject "the Period for Integrated Studies."

The new Course of Study for Elementary Schools describes what needs to be taken into account when making teaching plans in its Chapter 1, the general provisions [2]:

Teaching subjects should be intended to provide pupils with learning activities that help them to get accustomed to information means such as personal computers and the Internet, to acquire the basics of keyboard operation and basic knowledge of information ethics, and to be able to use those means appropriately, and it should further be combined with appropriate active use of teaching and learning materials and tools such as audio-visual apparatus and so on.

What makes a great difference from the previous Course of Study is that the new course states the necessity for the acquisition and active use of "skills like the basics of keyboard operation" and "basic knowledge of information ethics".

Moreover, the new Course states on "the Period for Integrated Studies":

Information studies should involve learning activities in which pupils can, through attempts at problem solving or investigation, collect, sort, and send information and can consider the influence of information on our life and society.

## **3 Information studies education in junior high schools**

In junior high schools, the subject of Technology and Home Economics generally covers information studies. The new Course of Study classifies the field of Technology as follows [3]:

- A Technology of materials and their processing
- B Technology of energy conversion
- C Technology of animal and plant growth
- D Technology of information

The content of D "Technology of information" is defined as:

- (1) The Internet and information ethics
- (2) The design and creation of digital arts and products
- (3) Measurement and control by computer programs

Half of the 175 hours of the subject Technology and Home Economics are allotted to the content of Technology. The learning of information used to take up half the content, but now takes up only a quarter of it. Thus the time period for information studies that formerly reached 44 hours currently amounts to only 22 hours. This means a significant change in quality as well as quantity.

## **4 Information studies education in senior high schools**

In senior high schools, the subject of Information chiefly serves for information studies education. It is comprised of General Information (Information as general subject) and Major Information (Information as major subject).

### **4.1 The subject General Information**

#### **The content of new subjects of information**

Emphasizing the necessity for students' proper acquisition of practical skills in actively using information and for their growing awareness of information ethics and caution about security and norms of information, the new Course of Study provides the following two subjects so that students can deepen and broaden their knowledge of scientific and social views or ideas on information and information technology according to the actual conditions of their levels of skills, aptitudes, curiosity, interests, and choices of future careers. [4]

### **Society and Information**

#### *1 Aim*

This subject is intended to acquaint students with the characteristics of information and its influence upon our society, to develop their skills in properly using information devices to collect, process, and express information along with their effective communication skills, and to nurture their positive attitudes toward participation in today's information-laden society.

#### *2 Content*

- (1) Active use of information and its expression
  - Characteristics of information and media
  - Digitization of information
  - Expression of information and communication
- (2) The Internet and communication
  - Information means and their development
  - The Internet and its mechanism
  - Active use of the Internet and communication
- (3) Issues of today's information-laden society and information ethics
  - The influence of information on our society and its problems

- Ensuring information security
- Laws and individual responsibilities in today's information-laden society
- (4) Construction of desirable information societies
  - Social information systems
  - Information systems and humans
  - Problem solving in today's information society

## **Science of Information**

### *1 Aim*

This subject is intended to make students understand the role of information technology and its influences on today's information-laden society, to give them scientific ways of thinking necessary for actively using information and information technology to find and solve relevant problems, and to foster their ability to participate in and contribute to the development of today's information-laden society.

### *2 Content*

- (1) Personal computers and the Internet
  - Personal computers and information processing
  - Mechanisms of the Internet
  - Operation of information systems and services provided
- (2) Problem solving and effective use of personal computers
  - Basic approaches to problem solving
  - Problem solving and computer programming
  - Modeling and simulation
- (3) Information management and problem solving
  - The Internet and problem solving
  - Accumulation and management of information and database
  - Evaluation and improvement of problem solving
- (4) Progress of information technology and information ethics
  - Informatization of societies and humans
  - Security of information-laden society and information technology
  - Development of informatized societies and information technology

## **The Characteristics of the new subjects of information**

The new subjects of information are marked with keywords such as "Information and communication network", "Information-laden society", and "Information ethics". "Society and Information" consists of "Media" and "Communication", and "Science of Information" is built on "Problem solving".

## 4.2 The subject Major Information

### Current situation

Alongside of “General Information”, the previous Course of Study provided 11 subjects of Major Information for senior high schools.

However, only a very small number of schools teach subjects of Major Information. Japan does not have more than 20 senior high schools that offer the course Major Information.

### Aim of the new subjects

While the aim of “General Information” does not differ between the previous and the new Course of Study, that of “Major Information” is enriched with keywords like “Ethical views” and “Information industry”:

The subjects are intended to make students acquire primary and basic knowledge of and skills in different fields of information, make them understand what significance and role information has in the present society, and to develop their creative and practical ability to solve social problems in positive, rational, and ethical attitudes and to endeavor to develop information industry and societies.

### New subjects of Major Information and their characteristics

The new Course of Study reorganizes subjects of Major Information by introducing some new subjects as well as revising the overall content of all subjects, which is expected to help foster qualified young people who have acquired the capability of creativity, thought, problem solving and understanding occupational ethics.

The new Course of Study reorganizes the subjects of Major Information into 13 in conformity with those characteristics. (Table 1)

**Table 1.** New subjects of Major Information

- |  |
|--|
| <ul style="list-style-type: none"><li>• Basic subjects<ul style="list-style-type: none"><li>— Information Industry and Society, Expression of Information and Its Management, Information and Problem Solving, Information Technology</li></ul></li><li>• System design and management subjects<ul style="list-style-type: none"><li>— Algorithm and Computer Program, Network System, Database, Information System Practices</li></ul></li><li>• Creation and production of information content subjects<ul style="list-style-type: none"><li>— Information Media, Information Design, Edition of Expressive Media and Expression, Information Content Practices</li></ul></li><li>• Synthetic subject<ul style="list-style-type: none"><li>— Project Study</li></ul></li></ul> |
|--|

### 4.3 Other subjects of information in high schools of industry and commerce

The Course of Study notified in 1979 provided some major subjects of industry and commerce which were concerned with information studies. Subsequently, Japan established the course of information technology in high schools of industry and that of information processing in high schools of commerce, which have played a central role of information studies education in the nation.

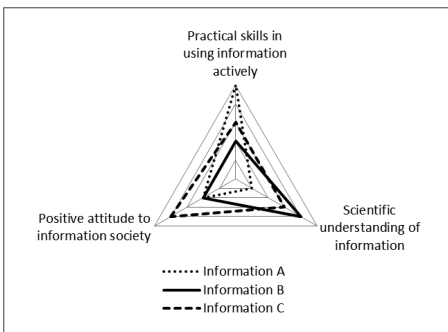
Likewise, the new Course of Study provides “Basic Information Technology”, “Electronic Information Technology”, “Programming Technology”, “Hardware Technology”, “Software Technology” and “Computer Systems Technology” for high schools of industry and “Information Processing”, “Business Information”, “Electronic Commerce”, “Programming”, and “Business Information Management” for high schools of commerce.

Other specialized high schools teach some other subjects of information: “Agricultural Information Processing” in agricultural schools, “Maritime Information Technology” in fishery schools, and “Effective Use of Nursing Information” in nursing schools. The subjects are learned by a lot of students in these specialized High Schools.

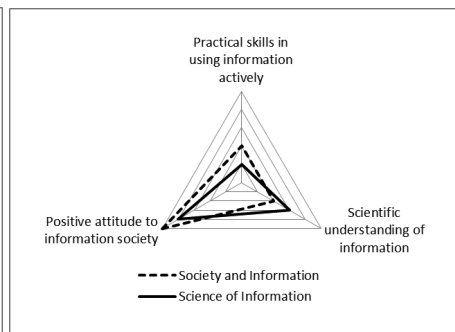
### 4.4 Current Problems

In the new Course of Study, where the three subjects “Information A, B, and C” are reorganized into the two “Society and Information” and “Science of Information”, these are still supposed to be designated by schools, not chosen by students themselves, and, what is more, the great majority of high schools plan to teach only “Society and Information”.

As can be seen from Fig. 1 and Fig. 2, “Society and Information” barely deals with scientific understanding of information and is almost void of such contents as information technology and problem solving that make up the fundamentals of Information Studies. [5]



**Fig. 1.** Previous subjects and gravity of 3 objectives (the authors' subjective judgment)



**Fig. 2.** New subjects and gravity of 3 objectives (the authors' subjective judgment)



What is even worse is that most teachers of Information Studies obtained a teacher's certificate of the subject by fifteen-day training some ten years ago. Since 2003, when the subject "Information" was introduced in schools, about half of the prefectures have not conducted an entrance examination for teachers of Information Studies, and no more than 20 % of prefectural or municipal boards of education have employed teachers of the subject continuously. Moreover, such teachers are required to have a certificate of a second subject in many of the prefectures and cities, which is not a required qualification for teachers of other subjects. [6]

## 5 Some suggestions for information studies education in Japan

Information studies education in the new Course of Study can be summarized with respect to the three objectives and school types/levels as in Table 2.

**Table 2.** The three objectives of information studies education and their practice at different school levels (the authors' subjective judgment)

	elementary school	junior high school	senior high school
practical skills in actively using information	***	**	**
positive attitude toward participation in today's information-laden society	**	**	***
scientific understanding of information	—	*	*

The number of asterisks in the table indicates that the objective is pursued actively (\*\*\*), moderately (\*\*), barely (\*), or not at all (—).

**Table 3.** The three objectives of information studies education and their idealized practice at different school levels

	elementary school	junior high school	senior high school
practical skills in actively using information	***	**	*
positive attitude toward participation in today's information-laden society	**	**	**
scientific understanding of information	*	**	***

Elementary and junior high schools put more weight on "practical skills in actively using information" than senior high schools. On the other hand, "positive attitude toward participation in today's information-laden society" is first dealt with in elementary schools but is more emphasized in junior and senior high schools. "Scientific understanding of information" is, by contrast, only touched on in some junior-high

subjects of Technology and Home Economics as well as in some senior-high subjects of Information.

The authors propose an ideal way of scheduling and practicing the subjects of information within the new Course of Study in Table 3. One cannot know the essential qualities of information without basic scientific understanding of its mechanism and operation. Moreover, the senior-high subject of Information needs more teachers and a wider range of subject choice as well as a more supportive infrastructure. It is urgent that the nation should foster skilled and knowledgeable teachers who understand well the essential nature of information studies and can command and display their skills and knowledge in classes. It is more desirable for every high school to teach both subjects “Society and Information” and “Science of Information” so that their students can choose between them according to their curiosity, interest and preferred future careers. For further studies, more senior high schools can ideally open subjects of Major Information.

Finally, scholars engaged in information science teaching including the authors need to know more about the current situation of and problems with, information studies education in primary and secondary schools. In today’s information-laden fast-paced society, people are encouraged to achieve the ability to lead an active, powerful and flexible life, coping with different problems and inconveniences all through their lives. Although Information studies are indispensable for achieving these objectives, their importance is currently extremely undervalued. We should exert ourselves to convince people in MEXT and boards of education, as well as learners and teachers at school, of the importance and attractiveness of “Scientific understanding of information”. The authors believe that it is our imperative duty to develop supporting tools needed in classes of information studies at school and to contribute to information studies education through research we are engaged in.

## Acknowledgment

This study is partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research 22500828.

## References

1. The Researchers Conference for Promotion of K-12 Information Studies Education: For implementation of systematic information studies education (The first report) (1997) [http://www.mext.go.jp/b\\_menu/shingi/chousa/shotou/002/toushin/971001.htm](http://www.mext.go.jp/b_menu/shingi/chousa/shotou/002/toushin/971001.htm) (last checked 1/31/2013)
2. Ministry of Education, Culture, Sports, Science & Technology (MEXT): Course of Study for elementary schools (2008) [http://www.mext.go.jp/a\\_menu/shotou/new-cs/youryou/syo/syo.pdf](http://www.mext.go.jp/a_menu/shotou/new-cs/youryou/syo/syo.pdf) (last checked 1/31/2013)
3. MEXT: Course of Study for junior high schools (2008)
4. MEXT: Course of Study for senior high schools (2009)
5. Nakano, Y., Tsutom, B.: WADA: The new Course of Study and future information studies education, *IP SJ Magazine*, Vol. 50, No. 10, pp. 996–1004 (2009)
6. Nakano, Y.: Information Studies Education for K-12, *IP SJ Magazine*, Vol. 48, No. 11, pp. 1181–1185 (2007)

# **Computational Thinking**



# The Role of Algorithm in General Secondary Education Revisited

Daniel Lessner

Department of Software and Computer Science Education,  
Faculty of Mathematics and Physics,  
Charles University in Prague, Czech Republic  
lessner@ksvi.mff.cuni.cz

**Abstract.** The traditional purpose of algorithm in education is to prepare students for programming. In our effort to introduce the practically missing computing science into Czech general secondary education, we have revisited this purpose. We propose an approach, which is in better accordance with the goals of general secondary education in Czechia. The importance of programming is diminishing, while recognition of algorithmic procedures and precise (yet concise) communication of algorithms is gaining importance. This includes expressing algorithms in natural language, which is more useful for most of the students than programming. We propose criteria to evaluate such descriptions. Finally, an idea about the limitations is required (inefficient algorithms, unsolvable problems, Turing's test). We describe these adjusted educational goals and an outline of the resulting course. Our experience with carrying out the proposed intentions is satisfactory, although we did not accomplish all the defined goals.

**Keywords:** computing science education, general secondary education, mandatory computer science foundations, concept of algorithm

## 1 Introduction

The average student of a Czech grammar school is not even aware of the existence of computing science (CS). We would like to change this. To provide some background for the desired discussion, we have developed an experimental introductory course to computing science, which will take up two hours weekly for ten months. The traditional role of algorithm in such courses, i.e. a preparation for programming, is revisited here, so that it would blend better into our general secondary education.

In this paper we first describe our reasons for adjusting the use of algorithms to grammar schools and the resulting goals. Then we follow the notion through the outline of our course. The next section introduces the criteria we used when working with algorithms described in natural language – an approach to enhance algorithmic thinking implemented in our course. Finally we share a few very brief observations from the realized experimental course.

To begin with we should like to clarify the terms we use.

## 1.1 Czech grammar schools

The Grammar school in Czech Republic is a branch of secondary education (the students are 15-19 years old). It provides general education as a preparation for further university studies of almost any kind. Around 20 % of the relevant age group attend grammar schools. The curriculum is defined in the Framework Education Programme for Secondary General Education (FEP) [1]. FEP defines six so-called key competences, complex structures of knowledge, skills, attitudes and values to be developed. These competences are: to be able to solve problems, to communicate, to learn, civic, entrepreneurial competence and personal and social competence. The educational content itself is first of all a means to develop these competences. Still, one needs languages to communicate, sciences to solve problems etc.

Computer related education is represented in FEP by a subject called *Information Science and Information and Communication Technologies*. The main focus is on utilizing digital technologies. Anything deeper is mentioned only rarely and briefly. FEP formally requires every grammar-school student to “apply an algorithmic approach to problem solving”. It is the last item of nine in that section, and it even includes “introduction to programming”. However, the usual way is to include these issues in an *optional* seminar for motivated students. We admittedly shift and change (and sometimes indeed lower) the usual goals and contents of such seminars, but this is due our effort to include *all* grammar-school students.

## 1.2 Computing science

In this paper, computing science means the discipline focused on efficient information processing [2]. The requirement of efficiency naturally, but not necessarily, leads to computer utilization. From the grammar school point of view, CS is the peer to natural sciences, such as physics or chemistry [6, 11]. They have a strong theory in the background, mathematical models, terminology, methods, extremely useful everyday technical applications and they provide inspiring problems and both cognitive and technical means to solve them.

Computing science in this paper has almost nothing to do with computer user-skills and very little to do with pure programming as a tool to develop software. The role of programming in our course has to be made clear here: it is not among the educational goals. This is not a unique idea for an introduction to CS [3]. Main goals of programming at grammar schools are clear communication, using a formal system and systematic thinking and workflow. We assume that these goals can be achieved by exposing students to the use of more formal systems (e.g. regular expressions, flowcharts etc.). A variety of possibilities is shown in the Bebras contest [5].

Omitting writing programs as a goal gives us extra time to focus on our goals more directly<sup>1</sup>. We work with sample programs, just as with other means, to describe an algorithm. We believe that exposing students to more paradigms is more beneficial for gaining insight into computing itself than one specific approach.

---

<sup>1</sup> After all, the ability to write programs is difficult to use without the ability to explain clearly what the program does.

### 1.3 Algorithm

The idea of algorithm seems to be clear, i.e. a reliable procedure feasible automatically, with no human interaction needed. However, specific descriptions differ and a precise formal definition is beyond the reach of the usual grammar-school student. To pronounce a process an algorithm the following basic conditions are required:

- *Explicit* (and finite) description of the procedure. Otherwise there is nothing to talk about.
- The procedure produces some output (and perhaps, but not necessarily, processes some input).
- The procedure is *finite*, regardless of the inputs.
- All the instructions are *elementary*, i.e. they are all known to everyone (everything) involved.
- The procedure is *deterministic*, i.e. the next step to do is always unambiguous. The consequence is that the output depends exclusively on the input.

We should mention *universality*, i.e. many different inputs of a kind can be processed by the same algorithm. It is also an often listed requirement for algorithm in Czech textbooks. It is however not an exact criterion and it may lead to some discussion (consider the algorithm to find  $\pi$ , or a trajectory to a specific comet, or *two* comets). It is a desired, but not a necessary feature.

Another desired, but not defining property is correctness. It is however not the property of a procedure itself. Correctness is bound to a procedure together with its task. Only for such a couple can we consider correctness. It does not affect the algorithmic nature of the procedure anyhow. The last property to mention is *efficiency*, i.e. resources consumption of the algorithm. Again, it is extremely important and desired, but not required for a procedure to be an algorithm.

The hereby described conception of algorithm allows many applications beyond the usual context of programming or symbol manipulation, yet it is not overly general and it provides a solid basis also for these traditional cases.

## 2 Why Do We Teach Algorithms?

Algorithm stands in the core of computer science (also in the sense of [4, 18]), some may even argue that it *is* the core. However, this fact alone is not a sufficient and understandable reason for students to embrace the term, and it should not be sufficient for the teacher either. The usual role of algorithms in computer-related education is that of a precursor for programming (e.g. [20, 8]), but not always, see [7]. Most of our grammar-school students will not become programmers. However, the notion of algorithm can still remain very useful for their everyday life. In this section we review the reasons for teaching algorithms and the consequential goals, both in the context of Czech grammar schools.

## 2.1 Reasons

Reasoning about educational content at our grammar schools must start with FEP in hand. Examining the detailed description of individual key competences (also in [21], what is a related material to FEP), we find overlaps of problem solving and communication competences with cognitive skills employed during algorithm development and utilization [22]. These include abstraction, disciplined systematic and formal thinking, and, on the other hand, creativity and appreciation for elegance. The very natural opportunity to develop key competences is the strongest reason for teaching algorithms at Czech grammar schools.

We would also like to introduce a few more thoughts about algorithms contributing to a solid general education. Formulating an algorithm, i.e. a procedure, which is reliable and does not need personal attention anymore, frees that person to do something else, perhaps something more useful, innovative, or enjoyable. The procedure can be carried out by anyone (anything) else. As any algorithm is essentially an information, it can be multiplied very cheaply, leaving the number of simultaneous processes only to physical limitations. This phenomenon is apparently of high importance in any profession supposed to provide reliable services. The capability to create and communicate reliable instructions and procedures may be the crucial competitive advantage.

Apart to the ability to utilize this phenomenon, another reason to teach algorithms is its recognition in other fields. A good example is the legal system. The administration of justice must be based solely on the law, independently of any personal opinions. To achieve this goal, the law has to be written in a certain way – meeting criteria almost identical to those we seek in algorithms. Another example is medical diagnostics.

Algorithms and their influence can be and should be recognized in other fields as well. The story of “machines taking over people’s jobs” has not yet come to an end. Financial markets can serve as a surprising example. A vast amount of transactions is controlled by computers. But they are not just executing instruments, they make the actual decision on what to buy or sell and when. Understanding algorithmics may allow students to better understand such changes and hopefully to foresee them soon enough to avoid the wrong career choice. They need to take an informed stand on which boundaries not to cross and which skills to develop in order to not allow the computers to become superior to them and to make further development safe, yet leave the actual work to computers.

Needless to mention, understanding algorithms is a necessary condition for understanding programming or other deeper aspects of computing.

## 2.2 Goals

The reasons described above lead us to formulate the following goals regarding algorithms<sup>2</sup>. Regarding basic notions and skills, students are to:

- understand the basic algorithmic properties well enough to recognize them (or their absence) on specific sample procedures, including appropriate reasoning and proofs (e.g. about finiteness);

---

<sup>2</sup> These goals do not constitute the whole course – we have chosen only what is related to algorithms.



- explain the consequences of each algorithmic property in practice (e.g. reliability, independence of the executor etc.);
- read and carry out an algorithm described in various forms (e.g. natural language, pseudocode with mathematical notation, flowchart, programming language in simple cases);
- find out what a given algorithm is probably good for;
- “debug” an algorithm: test it using appropriate inputs, follow the relation between input and output, isolate the flaw;
- modify, enhance or finish an existing algorithm;
- describe a known algorithm precisely in natural language, or choose another approach when appropriate;
- utilize basic concepts and control structures such as variables, decisions, loops, functions;
- utilize known basic techniques to find unknown algorithms (e.g. decomposition, systematic examination of all options etc.);
- know the basic criteria for algorithm comparison and compare the given algorithms;
- choose an appropriate basic instruction and estimate the algorithm’s complexity class.

Regarding limitations of algorithms, students are to:

- understand that algorithms with exponential complexity are often practically useless;
- be aware of the existence of workarounds for practically unsolvable problems (heuristics, approximate solutions, probabilistic algorithms etc.);
- know that well-defined principally unsolvable problems exist, i.e. we do not have an algorithm for every task;
- recognize typical problems unlikely to have an algorithmic solution (e.g. problems with no solution, questions about the future without sufficient data, dealing with subjectivity or inconsistent definitions, self-reference);
- understand that computability does not depend on a specific machine or technology, computers are equally strong;
- understand the concept of Turing’s test;
- recognize real world variations of Turing’s test (e.g. CAPTCHA, teacher checking for student’s true understanding);
- be aware of the advantages and disadvantages of both the human mind and a computer (including that a computer can only do what is algorithmic).

Regarding attitudes and opinions related to algorithms, students are to:

- appreciate abstraction utilization and the concept of black-box;
- prefer creative work and unpredictable environment;
- prefer to avoid stereotype and routine tasks and try to have a machine solve them instead;
- judge reasonably when to utilize an algorithmic approach a to a problem;
- judge reasonably when to prefer man over a machine for a task and vice versa.

### 3 The Notion of Algorithm in the Experimental Course

First we briefly describe the experimental course itself, so that we could discuss how algorithms are dealt with. The purpose of the course is to find out whether and how CS can be taught to grammar-school students and to provide some evidence for the related discussion. The course will introduce students to CS and provide them with a basic overview. They should be able to utilize the basic skills in real life situations. The course will also contribute to the development of the key competences mentioned above. All this is in accordance with other natural sciences at Czech grammar schools.

The course is structured into modules, which explicitly focus on different fundamental topics. Other topics are in the background, as can be seen in the example of algorithm. A module will last one month (four 90 minutes' lessons). The modules sometimes take longer if students lack necessary mathematical knowledge and skills<sup>3</sup>. More details on the course design can be found in [12]. We can proceed now to the description of each module and its contribution to algorithm study.

#### 3.1 Preliminary Work

Our previous experience showed that the notion of algorithm is too abstract and meaningless to begin with. So we decided to leave the notion for later, when we already have some more experience with algorithmic processes and their advantages. Here we describe modules which precede the algorithm module.

In the *information* module students shall understand what “information” means, how we measure it and how we encode it efficiently. We use a number guessing game as a model, similar to [7]. Information amount measuring is based on Hartley’s approach [9], i.e. decrease of possibilities<sup>4</sup>. The module also introduces the binary numeral system and the concept of decision (or encoding) trees as a visualisation of the process. Further details are published in [13]. From this paper’s point of view, trees are early algorithm examples. Last to be mentioned is the concern about efficiency: we want to be done with our tasks fast, i.e. we want our trees to be shallow and the frequent nodes to be close to the root. Students realize the existence of minimal, average and maximal steps needed. The ideas relevant to algorithms are not named explicitly, but they are certainly addressed and dealt with.

The *graph* module introduces graphs as a strong tool for relation modelling. We add some terminology to unify the many examples already known. Students learn that even such visual structures can be encoded into zeros and ones, e.g. as adjacency matrix. Most of the time is spent examining Eulerian graphs (can a watchman walk the park paths efficiently?) and finding the rule to determine whether a graph is Eulerian. Students reformulate the rule into an instruction set. We show it to them in the form of a simple Python program for comparison. They can also experiment with it and modify it to become acquainted with basic programming concepts.

---

<sup>3</sup> We need to apply mainly logic, combinatorics, probability, also logarithms are helpful. Sadly, these are rather not popular among students.

<sup>4</sup> Shannon’s classical entropy approach is out of reach for an average grammar school student.

Again, basic efficiency issues are discussed. This includes asymptotic estimates, but also more down to earth questions: Shall I first calculate all the degrees, and then check that they are even, or can I be done sooner? Do I even need to actually calculate the degrees for seeing whether they are odd or even?

The *problem* module deals with general problem solving strategies, mostly adapted from [17]. They form a base for future algorithm developing. The underlying principles are examined using classical problems, such as wolf, goat and cabbage. Apart from heuristics like “chop it into parts”, “take another point of view”, “forget the unnecessary conditions” and “why exactly are you even doing this?”, students realize the importance of defining the actual initial and the desired final state as well as the possible changes to each state. This leads directly to state space and its systematic traversing.

### 3.2 The Algorithm Module

The next module deals with algorithms explicitly. We provide a more detailed description here. The whole module begins with a task to describe sufficiently some seemingly easy procedure, such as to tie a knot or fold a paper into a cap. The sufficient description will not allow the fellow student to do anything differently from what the author intended. It turns out, that almost no one is capable of instructing others properly – not even on the fly, with direct feedback. This discovery is sufficiently striking for most of the students to try to do better and investigate the related theory.

The common and desired properties of already known procedures are examined. Based on this, the notion of algorithm itself is introduced. Then each property gets some attention: what is it, what does it cause, and how do we recognize them in given examples. This is also an opportunity to show a fairly abstract tool, the idea of a decreasing measure for proving finiteness. We use real world problems such as cooking or guitar tuning and classic CS algorithms such as bubble-sort. Correctness and efficiency are discussed, the latter in more detail in the next module.

The last area to cover touches the fundamental limitations of algorithmic procedures. We explain the idea of the halting problem and make it feel more real referring to virtualization. With a few more examples (mostly from logic), students will realize that the cause of our trouble is self-reference. Other types of unsolvable problems are examined: we cannot construct a triangle with two right angles in a plane, we can not check the equality of two real numbers, we can not predict quantum mechanical phenomenons, we cannot assure everyone of a happy life (unless we consider the trivial solution [10]).

We can see that there is virtually nothing about algorithm development itself. These issues are distributed among all the modules, where algorithms are developed. This approach is used also for other traditional topics (e.g. debugging).

### 3.3 Following Extensions

The concept of algorithm is not left behind during the other modules. The *efficiency* module lets students compare algorithms for the same task and provide some tools for that, such as the idea of basic step for considering large inputs. We examine and compare linear and binary search, sorting algorithms (e.g. already known bubble-sort with

merge-sort) and other examples. Based on this experience, some basic heuristics for optimization are shown, such as “reuse your results” or “prepare your data”. In this module also the existence of practically useless algorithms and practically unsolvable problems is shown. Students learn how to roughly distinguish the uselessly slow algorithms.

The next module deals with *evolutionary algorithms* as an illustration of some advanced computing science to complete the overview. The basic framework is explained and the teacher provides students with a simple Python code to search Hamiltonian paths. They experiment with it, tweak some parameters and discover issues like convergence, optimal population size, keeping the best individuals through generations, exploration vs. exploitation etc. All this serves as a challenge to the known and safe conceptions. This will help students to find a place for computing in the rest of the world and prepare them for the last module.

The last module is about links to *humanities*. The core topic is the relation between human mind and computer. The concept of the Turing test is introduced. Students will take a stance on the topic and define the fundamental difference between humans and machines. Their points of view are supported by the hands-on experiences with computing gained during previous modules. This module is not built like the others, i.e. solving a problem of some computational nature. Students do not produce algorithms and quantitative analysis, they analyze and discuss arguments and produce essays and presentations instead.

## 4 Criteria for Algorithms in Natural Language

In this section we describe the criteria used to evaluate natural language descriptions of algorithms. First we add a few thoughts to the whole idea, proposed and used e.g. in [14, 15, 19]. In their lives, most of the students will deal exclusively with instructions written in natural language – carrying them out, assessing them, and hopefully also creating them. We naturally expect that their skills can get better thanks to mathematics, programming etc. But we find the direct and explicit approach more promising.

One of the first advantages is motivation. Students usually see more sense in enhancing the related skills than in the case of e.g. programming. Furthermore, they see very well, how wrong their descriptions are. If the procedure does not work, students can blame neither the computer nor “the obscure programming language”. The advantage, which is not obvious at first, is their *growing* sympathy for formal systems of various kinds. Writing precise descriptions in natural language is difficult. This fact (supported by personal experience) makes them appreciate clear formal systems (even programming itself), which simply eliminate the possibility of many mistakes just by design. For the same reason they appreciate simple abstract models as practical under certain conditions.

We share a few more details on how we evaluate students’ results. First of all, we try not to evaluate them. As in the whole course, we try to design situations, in which the result quality naturally rewards (or punishes) the student. For example, a slow algorithm literally takes its time. With algorithms in natural language, a fellow student is usually a fair judge. A misunderstanding is exactly the kind of natural “punishment” we are looking for. For providing a more formal feedback, we found inspiration in language

education. Essay evaluation often breaks down to a set of criteria [16]. The evaluator assesses only the individual criteria fulfillment. The information on each criterion serves as valuable feedback to the student and allows him to advance somewhat systematically.

To employ this idea, we need a set of appropriate criteria. This work starts in the algorithm module. The criteria are continuously refined during the rest of the course. Students evaluate each other's work anyway, so we opened the process for them. This approach has clear advantages over just stating the already developed criteria. Most of all, criteria based on students' actual needs make more sense to them, they feel a sort of ownership, and they are more willing to meet the criteria. The teacher can of course make suggestions. His work, however, consists mostly of managing the process. This includes encouraging students to differentiate between necessary and satisfactory conditions and to formulate them precisely. Another aspect is generality of the criteria. We have no preferred computational model, so the criteria will work as universally as possible. Development of criteria for assessing algorithm description quality has a nice recursive touch: the criteria will be written meeting themselves.

Here is an example of a criteria set. As description flaws were similar, similar criteria were devised in both groups.

- Language: used correctly.
- Specification: clear description of allowed inputs and intended purpose.
- Semantics: clear nature of all text: instructions, asserting statements etc.
- Terminology: all used words are common knowledge or well defined.
- Instructions: do not allow alternative interpretations.
- Work flow: always clear where to continue.
- Decisions: all possible outcomes explicitly covered.
- Unexpected situations: non-existent, every possible situation is anticipated and covered<sup>5</sup>.
- Loops and recursion: explicitly denoted body, clear ending condition, clear routine to change variables (or whatever material the loop works with).
- End: Explicitly stated, together with clear output.

For estimating the quality of a description which satisfies these necessary conditions, we have developed *recommendations*. It is usually helpful to follow them, but it is not necessary. Just as the criteria above, these recommendations are hardly surprising for anyone familiar with programming. But they are quite a discovery for the beginners.

Here is a sample list of such recommendations:

- State clearly the purpose of the algorithm.
- Put one instruction on a line, do not write in paragraphs.
- Use indentation to indicate structure.
- Number or label the lines, refer to them.
- Do not avoid repetition of words, use one word for one meaning.
- Do avoid repetition of instruction sequences. Define loops or functions instead.
- Use the concept of a variable, when appropriate.

---

<sup>5</sup> Strictly taken, this is impossible without a proper computational model definition or a wide range of prerequisites.

- Comment on the algorithm, explain why it works and what the meaning behind individual instructions is.
- Each instruction must be detailed enough to prevent misunderstanding. Do not expect kind cooperation. Everything that is to be done must be said.

To illustrate the criteria we show a very wrong example. It is synthesized from the work of our students to show multiple flaws in one sample. The described procedure is supposed to add 1 to a binary number on the input. Knowing how it should work we can see that the student perhaps means well. He himself is unlikely to make a mistake while adding 1 to a binary number. However, the description itself is practically useless for anyone who does not know what to do from some other source. With the help of evaluation criteria, the student can systematically identify the flaws and correct his algorithm.

*Example of a procedure described in natural language:*

In binary numeral system we have only 1 and 0. If we run into a 1 while adding 1, we move over to the next digit. If there is a 1 again, we move over again, and we continue this, until we find a 0, which we overwrite to 1.

## 5 Evaluation

We tested the above described approach in two optional seminars (i.e. small groups of students who chose the subject among a wider offer). The concept underwent some adjustments during the school year. Also the groups were too small and proper control groups were not available. The obtained data is therefore almost useless and difficult to interpret seriously. However, we will share the main findings briefly.

The level of motivation of the students was higher than in other years (and other students) with more traditional approaches. Students have explicitly reported that they could see links to their lives, both present and future, even though not necessarily related to computers. They also found dealing with their natural language more challenging than other options.

We have not met all the defined goals. Some parts turned out to be too demanding (mostly finiteness proofs and the concept of state space). On the other hand, some turned out to be accepted surprisingly well (we had been especially apprehensive about the last two modules, but the results were good). Most of the topics worked as expected, namely asymptotic complexity is perhaps a borderline for many students.

## 6 Conclusion

We have shown a novel point of view to algorithms in general secondary education. It aims to match the main goals found in the Czech curriculum, mainly in developing key competences to solve problems and to communicate. On the other hand, the traditional focus on programming and computers themselves is suppressed (though not removed).

We argue that an algorithm described in natural language is a valuable tool in computing courses in general education. We have shown a criteria-based evaluation tool and explained its functions.

We observed a rise of motivation among students and improvement in their basic skills, such as precision in communication. We intend to investigate the effects of the new approach on the key competences themselves. However, that will be very challenging, for their description is far from algorithmic.

## References

1. Framework Education Programme for Secondary General Education (Grammar Schools). Výzkumný ústav pedagogický v Praze, Praha (2007)
2. Aho, A., Ullman, J.: Foundations of computer science. Computer Science Press, New York (2000)
3. Bell, T., Curzon, P., Cutts, Q., Dagiene, V., Haberman, B.: Introducing students to computer science with programmes that don't emphasise programming. In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11. ITiCSE '11, vol. 5, p. 391. ACM Press, New York, New York, USA (2011), <http://doi.acm.org/10.1145/1999747.1999904>
4. Bruner, J.S.: The process of education, vol. 115. Harvard University Press (1977)
5. Dagiene, V., Futschek, G.: Bebras International Contest on Informatics and Computer Literacy: A contest for all secondary school students to be more interested in Informatics and ICT concepts. Proc. 9th WCCE (2009)
6. Denning, P.J.: Computing is a natural science. Communications of the ACM 50(7), 13 (Jul 2007)
7. Fellows, M.R., Bell, T., Witten, I.: Computer Science Unplugged - offline activities and games for all ages: Original Activities Book. Computer Science Unplugged (1996)
8. Gal-ezer, J., Harel, D.: Curriculum and Course Syllabi for a High-School Program in Computer Science. Computer Science Education 9, 114–147 (1999)
9. Hartley, R.V.L.: Transmission of information. Bell System techn. Journal 7, 535–563 (1928)
10. Holan, T.: Všichni lidé šťastni (2010) <http://drupal.geometry.cz/k54> (last checked 1/31/2013)
11. Hromkovič, J., Steffen, B.: Why Teaching Informatics in Schools Is as Important as Teaching Mathematics and Natural Sciences. In: Proceedings of the 5th international conference on Informatics in Schools: Situation, Evolution and Perspectives. pp. 21–30. ISSEP' 11, Springer-Verlag, Berlin, Heidelberg (2011)
12. Lessner, D.: Computer science curriculum proposal for Czech grammar schools. In: Zborník príspevkov. pp. 99–104. ITATĀŽ11, PONT s. r. o., SeĀLa, Slovakia (2011)
13. Lessner, D.: Information Theory on Czech Grammar Schools: First Findings. In: Knobelsdorf, M., Romeike, R. (eds.) Pre-proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE), pp. 139–142, Hamburg (2012)
14. Miller, L.A.: Natural language programming: styles, strategies, and contrasts. IBM Syst. J. 20(2), 184–215 (1981), <http://dx.doi.org/10.1147/sj.202.0184>
15. Murphy, L., Fitzgerald, S., Lister, R., McCauley, R.: Ability to 'explain in plain english' linked to proficiency in computer-based programming. In: Proceedings of the ninth annual international conference on International computing education research. pp. 111–118. ICER '12, ACM, New York, NY, USA (2012)
16. O'Donovan, B.: Improving students' learning by developing their understanding of assessment criteria and processes. Assessment and Evaluation in Higher Education 28, 147 (2003)

17. Polya, G.: How to solve it: A new aspect of mathematical method. Princeton University Press, 2 edn. (1957)
18. Schwill, A.: Fundamental Ideas: Rethinking Computer Science Education. *Learning & Leading with Technology* 25(1), 28–31 (1997)
19. Simon, B., Chen, T.y., Lewandowski, G., McCartney, R., Sanders, K.: Commonsense computing: what students know before we teach (episode 1: sorting). In: Proceedings of the second international workshop on Computing education research. pp. 29–40. ACM (2006)
20. Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., Verno, A.: A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee. Computer Science Teachers Association, New York, second edn. (2003)
21. Šlejšková, E.: Klíčové kompetence na gymnáziu. Výzkumný ústav pedagogický v Praze, Praha (2008)
22. Wing, J.M.: Computational thinking. *Communications of the ACM* 49(3), 33 (2006)



# An Epistemic Hypermedia to Learn Python as a Resource for an Introductory Course for Algorithmics in France

Christophe Reffay<sup>1</sup>, Mahdi Miled<sup>1</sup>, Pascal Ortiz<sup>1</sup> and Loïc Février<sup>2</sup>

<sup>1</sup> STEF (Science Education Education training), Ecole Normale Supérieure of Cachan, France  
{christophe.reffay, mahdi.miled, pascal.ortiz}@ens-cachan.fr

<sup>2</sup> MOCAH team, LIP6, University Pierre et Marie Curie, France, loic.fevrier@france-ioi.org

**Abstract.** We launched an original large-scale experiment concerning informatics learning in French high schools. We are using the France-IOI platform to federate resources and share observation for research. The first step is the implementation of an adaptive hypermedia based on very fine grain epistemic modules for Python programming learning. We define the necessary traces to be built in order to study the trajectories of navigation the pupils will draw across this hypermedia. It may be browsed by pupils either as a course support, or an extra help to solve the list of exercises (mainly for algorithmics discovery). By leaving the locus of control to the learner, we want to observe the different trajectories they finally draw through our system. These trajectories may be abstracted and interpreted as strategies and then compared for their relative efficiency. Our hypothesis is that learners have different profiles and may use the appropriate strategy accordingly. This paper presents the research questions, the method and the expected results.

**Keywords:** Adaptive hypermedia, Navigation, Programming learning, Python, Trajectories.

## 1 Introduction

Despite the major role of technology and especially computers in the society, teaching informatics still remains barely present in primary and secondary French schools [1]. Nevertheless, an optional course called "ISN" (Informatics and Digital Sciences) is opening this year at the secondary school. This novelty may contribute to promote a renewal for teaching informatics in France. At the organization level, it presents a problem to have enough well-prepared teachers to take charge of such a technical topic. It seems to us that the France-IOI platform could be useful for the training of teachers as individuals, and a special sequence of exercises (currently 108) has been designed for their pupils too. During the last three years, a novel approach to build an adaptive hypermedia has been developed by the third author. It has been applied to teaching programming with Python [2]. We decided to combine France-IOI exercises and the adaptive hypermedia to support the practical part of the ISN course.

The rest of this paper is organised as follows. The next two sections present each of the main resources of this environment: France-IOI exercises and the adaptive hypermedia for Python. We then present how they are combined in the proposed experiment and what are the research questions, methods and expected results. Finally, we give a short conclusion and many perspectives of this work in progress.

## 2 The France-IOI Infrastructure and Sequence of Exercises

The main objective of the France-IOI non-profit association is to prepare French teenagers for International Olympiads in Informatics. For this particular public, the goal is to produce very fast thinkers and programmers mastering optimization techniques. The practice-oriented platform supports algorithmics learning and multi-language programming [3]. Its main feature consists in an automatic validation of learners' source-codes. Learners are invited to solve lots of exercises from programming basics to advanced algorithmic concepts.



Fig. 1. Exercise from the France-IOI platform: an example.

As France-IOI does for Olympiads challengers, the ISN course focuses on algorithmics more than programming. But the target level of expertise is dramatically lower for ISN pupils. For this reason, a sequence of many exercises (currently 108) including some concept presentation has been adapted for ISN pupils. This sequence of exercises is already available on the France-IOI platform [4]. One of those is illustrated in Fig. 1. Its task consists in writing an ordered sequence of instructions.

These exercises are organized in 12 topics, namely: display text and sequence, repeating instructions, variables and calculation, reading entries, test and conditions, advanced structures, advanced conditions and Boolean operators, “repeat” instruction, floats, array, strings and functions. New concepts are presented in some exercises in a

just-in-time and just-what-is-needed spirit. A global scenario tells a story where each exercise is a new challenge. These last two aspects lead designers to present these exercises in an explicitly suggested order. But pupils can access and solve exercises in any order. Each exercise is presented in a web page (as illustrated in Fig. 1) providing 5 parts (presented as modular tabs): Task, Resolution, Hints, (extra) Activity and (commented) Solution. For each learner, only the “Solution” remains unreachable until a valid code has been submitted to solve the corresponding problem. If a learner achieves only part of the exercises, leaving holes in the sequence, s/he will hardly figure out which concepts or important notion s/he missed. This is precisely where the epistemic graph proposed by P. Ortiz should help.

### 3 An Emerging Adaptive Hypermedia for Python Learning

This part focuses on the graph of *epistemes* and their pre-requisite dependencies. We first present a sub-graph extraction for illustration in Fig. 2. The heart of this section emphasizes four principles that guided the construction of the domain knowledge and the content of each *episteme*. We then show how these principles greatly simplified the construction of the particular subset, dedicated to support the resolution of France-IOI/ISN set of exercises. The last section situates this work in progress in the literature related to Adaptive Hypermedia applied to programming.

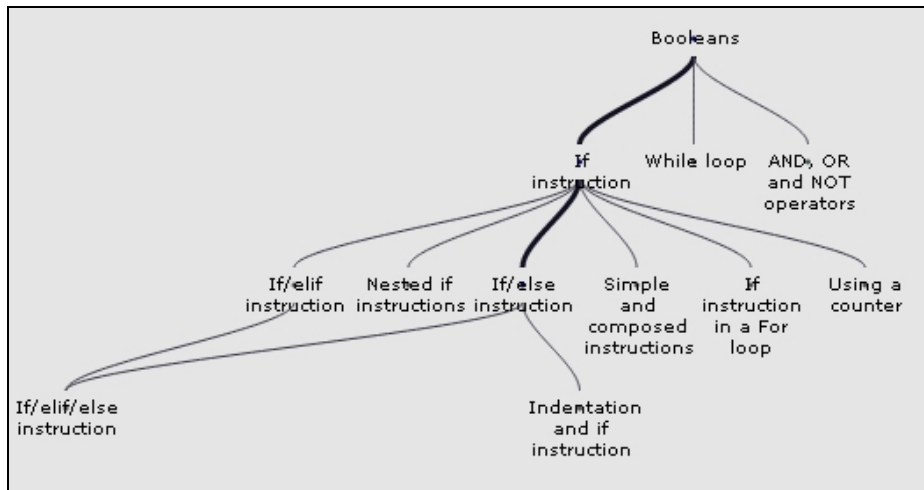


Fig. 2. Sub-graph headed from Boolean

Learning programming languages is usually assisted by documentation or textbooks which are rather linear. As presented in [5] for object-oriented concepts, initiation into programming involves many new concepts that are seen by novices as complex in the sense that it is hard to see where we have to start to be able to achieve a particular goal. For us, this is true even for basic concepts for a novice programmer.

Python appears to be a good candidate for a first programming language. We propose a hypermedia model based on a typical representation of the knowledge domain for Python by epistemic units also called "*epistemes*" [2]. These *epistemes* are linked by prerequisite relationships and form a Direct Acyclic Graph (DAG).

### 3.1 The Graph: Fine Grain Concepts and Prerequisite Dependencies

This part can be seen as the "Document Space" defined in [6]. The graph proposed in Fig. 2 is the result of a systematic process guided by four principles presented in the next section.

Each node (*episteme*) focuses only on a very limited concept or language construction and presents it in a *typical* program code. As an example, this is the reason why the various aspects of the "If" construction are split in: "If instruction" → "Indent an If" → "if/else instruction" → "if/elif/else instruction" (see Fig. 2).

A→B generally means: "concept A is a prerequisite for concept B". Sometimes this rule is relaxed and could be interpreted as "A is useful to build a more *typical* and comprehensive example presenting B".

For a large coverage of the language, in order to serve as an integrated continuum from introductory course to reference manual for Python, this DAG should count several hundreds of *epistemes*. For this reason we need various tools around the DAG in order to build an adaptive hypermedia. These tools may support: graph and episteme edition, graph browsing, recommendation, information retrieval (for a novice who may use inappropriate labels for concepts), etc. A first study aiming at selecting the optimal graph visualisation tool for a given task is presented in [7].

### 3.2 Four Principles Guiding the Graph and Episteme Edition

As mentioned above, the graph construction and episteme description are guided by the application of the following four strong principles (detailed in [2]):

1. **Salience**: Only the focused concept may be visible in an episteme;
2. **Separation**: Each concept should be presented independently to any other;
3. **Occam razor**: present only what is necessary, delete everything else;
4. **"In code we trust"**: the more a program code is frequent in the code ocean (python source code available on the web), the more **typical** and **relevant** it is for us.

How little is an episteme?

– Very little!

It should present a typical code using the concept and comment it. When appropriate, it will point out typical errors. For complex concepts, more interactive media may be used (demo, slideshow etc.) in order to reduce the textual explanation.

### 3.3 Producing a Sub-graph on Demand

As he was building the Python programming knowledge domain ( $G_1$ ), the third author extracted the sub-graph ( $G_2$ ) of 76 *epistemes*, sufficient for a learner to write in Python a solution for all 108 exercises (ISN) proposed by the France-IOI platform.

The main differences between  $G_1$  and  $G_2$  are due to the execution context that is simplified (in  $G_2$ ) by the fact that pupils write their program code directly into the web-based platform. In opposition,  $G_1$  provides *epistemes* that specifically support learners in their local Python programming environment installation phase and others describing the different ways to produce, interpret and execute Python code. Marginal adaptation has been made also because of a shift of Python version: 2.7 to 3.1.

Except these two main differences,  $G_1$  has been extensively reused to build  $G_2$ . Let us give some details on this systematic process:

For each exercise, provide the Python code of (at least) one acceptable solution. From each code, define the list of concepts/*epistemes* this code requires. If the concept is already presented by an episteme in the graph, just bind the exercise to it. If not, define and integrate the missing *episteme(s)* in the graph according to the 4 principles mentioned in the previous section. To build  $G_2$ , reusing  $G_1$ , only 3 new *epistemes* had to be defined and 30 were modified (repeatedly on the same facets).

### 3.4 Related Works in the Adaptive Hypermedia Research Field

According to the Adaptive Hypermedia (AH) review [8], methods, framework and techniques in this domain have been already studied extensively in lab conditions on theoretical concepts that shaped this research domain. Our system can be considered as a practical educational web-based AH where adaptation is mainly based on a map for navigation support like RATH [9]. However, we did not find large-scale experiments in secondary schools as proposed in this project in the literature.

Concerning programming, many ideas and adaptation techniques have been successfully implemented for example around QuizPACK (Quizzes for Parameterized Assessment of C Knowledge) [10] and QuizJET (Java Evaluation Tool) [11]. Other systems like BlueJ or Scratch [12-13] may be considered as IDE preferred by teachers for novices in order to consider interesting programming concepts as fast as possible without having to write heavy source code or deal with the syntax troubles. Our preference goes to better support the first steps (code correction) including the language syntax and typical constructions.

## 4 Experiment

We have presented in both of the previous parts the France-IOI platform and its proposed list of exercises for ISN on the one hand, and the graph of epistemes for Python on the other hand. We also emphasised the positive context in the introduction. Being currently in the process of finalizing software, we cannot measure the platform audience yet.

This part describes the objectives of the experiment, defines *trajectories* and presents the method through the 3 phases: data collection, analysis and expected results.

#### **4.1 Objectives and visual adaptation of the navigation map**

In this exploratory experiment, we want the navigation map to visually differentiate for an authenticated pupil: visited epistemes, validated exercises and recommended ones as a next step. We will focus only on these basic properties and adapt the navigation tool provided to a particular user according to his/her profile. For each exercise, a sub-graph representing the transitive closure of all needed epistemes will be provided in *graph* tab. The user will be able to browse this sub-graph where unvisited and non-validated epistemes are emphasized. Bigger navigation maps (i.e.: for a chapter or the entire course) may be tested in this experiment.

Our purpose is to leave the locus of control to the learners for their progression. We want to observe afterwards the different trajectories they finally draw through the system. These trajectories will be analyzed as patterns and interpreted as strategies.

#### **4.2 Collecting Data**

On the platform, each user is authenticated and may be related to an ISN group. We collect for all users the timestamps of page visits (exercise task, hints, episteme) and exercise resolution attempts (failures and success). These observed events are considered as primary traces (time-situated) [14] and will constitute the essential part of the corpus for this first experiment. If needed for disambiguation purpose, or better description of profiles, we may employ interviews or questionnaires for pupils.

#### **4.3 Defining Trajectories**

Considering primary traces, we could compose them (selection, filtering or composition techniques) to elaborate more significant traces which can also be called modelled traces [14]. In such a framework, we can define trajectories at different levels:

- The micro-level trajectory is the ordered sequence of events (visited objects, submission etc.) for a specific user between the (first) visit of the exercise description (i.e.: task definition) and the first submission of a correct solution.
- The macro-level trajectory is the ordered sequence of successfully solved exercises for a specific user in a given period and for a given set of exercises.

#### **4.4 What is Measured? Analyzed? How are Trajectories Compared?**

Taking classmates into account, individual trajectories will be compared in a class or for all classes to detect and enlighten recurrent patterns. For example, for a given exercise some students will visit epistemes (as hints) and others will not. We are interested to see if there is an impact on the resolution time or on the number of attempts.

Which are the criteria to evaluate trajectories? What is an efficient trajectory? Did the user visit the episteme recommended by the navigation tool? Did this visit positively impact his/her trajectory? Are there strong and weak learners' specific trajectories?

“Resolution duration” for a given trajectory is hard to define. We may have to distinguish a “global resolution duration” (i.e.: simple timestamps' difference between the first and the last event of the micro-trajectory) and a “connected resolution duration” considering only the (short) connected time periods within this time interval.

We plan to provide to the teachers and pupils of ISN groups an automatic representation with a global graph where all epistemes would be represented as pies reflecting the proportion of pupils (in the class) who validated it.

#### **4.5 What is Expected?**

The first expected element is about the use of different strategies to solve an exercise. Automatic analysis of primary and modelled traces will give hints about recurrent patterns in trajectories at micro and macro levels. We could expect stronger students solving exercises without using hints or visiting epistemes contents. We will identify the most visited epistemes and try to evaluate their impact on the resolution process. Resolution duration statistics should inform on the practical difficulty pupils are facing while solving exercises in an introductory programming course.

### **5 Conclusion**

Observing practical processes of algorithmics and programming learning at a national level, we are testing a first implementation of an original combination between a set of exercises and an adaptive graph of epistemes based on strong principles. Being operational in time for the first cohorts of the ISN course, we hope this may attract many teachers and pupils, offering a large number of participants for this experiment. The many traces we hope to collect should (1) inform designers about acceptability and usability of the scenario and tools (including the DAG of epistemes) and (2) shed light on our research questions. Potential reuse of these data with data mining and smart visualisation techniques may contribute to educational data mining or learning analytics fields. But other fields of investigation may concern these issues:

- How to generalize for other programming languages? Other subjects? Other levels? Other countries?
- How to support navigation and editing processes for such an epistemic interactive hypermedia? How can we share, reuse and collectively update it?
- How to introduce clones or forks of such an epistemic interactive hypermedia or adapted parts in order to better fit particular needs or constraints?
- And more generally: How to support teachers and decision-makers to refine programming curricula from such usage experiences?

As a next step, we can investigate if opening the navigation history of pupils to their peers as shown in [15] could positively influence other pupils' trajectories.

## References

1. Baron, G.-L., Bruillard, É.: L'informatique et son enseignement dans l'enseignement scolaire général français : enjeux de pouvoirs et de savoirs. In Lebeaume, J., Hasni, A., Harlé, I. (eds.), *Recherches et expertises pour l'enseignement scientifique*. Bruxelles : De Boeck, pp. 79–90 (2011) [http://www.stef.ens-cachan.fr/annur/bruillard/2010\\_B&B\\_DeBoeck.pdf](http://www.stef.ens-cachan.fr/annur/bruillard/2010_B&B_DeBoeck.pdf) (last checked 1/31/2013)
2. Ortiz, P.: *Hypermédia adaptatif à épistèmes pour l'apprentissage des langages de programmation*. Poster aux Rencontres Jeunes Chercheurs en Environnements Informatiques pour l'Apprentissage Humain, Amiens, 22-23/05 (2012)
3. Hiron, M., Février, L.: A Self-paced Learning Platform to Teach Programming and Algorithms. *Olympiads in Informatics*, Vol. 6, pp. 69–85 (2012) [http://www.mii.lt/olympiads\\_in\\_informatics/htm/INFOL105.htm](http://www.mii.lt/olympiads_in_informatics/htm/INFOL105.htm) (last checked 1/31/2013)
4. <http://france-ioi.org>, The platform of the France-IOI non-profit Association
5. Pedroni, M., Meyer, B.: Object-Oriented Modeling of Object-Oriented Concepts: A Case Study in Structuring an Educational Domain. In Hromkovic, J, Kralovic, R. & Vahrenhold, J. (eds). *ISSEP 2010*, January 2010, Zurich, CH. LNCS 5941, pp. 155–169 (2010)
6. Henze, N., Nejdil, W.: Logically Characterizing Adaptive Educational Hypermedia Systems, Workshop Session at WWW 2003, Adaptive Hypermedia (2003)
7. Miled, M. : Vers une mise en relation des activités d'édition et de navigation dans les ressources d'apprentissage : cas de l'apprentissage d'un langage de programmation. *Rencontres Jeunes Chercheurs en EIAH*, Amiens, May 22–23 (2012) <http://hal.archives-ouvertes.fr/hal-00705889>
8. Brusilovsky, P.: Adaptive Hypermedia. *User Modeling and User Adapted Interaction* 11 (1/2), 87-110. (2001) <http://www.sis.pitt.edu/~peterb/papers/brusilovsky-umuai-2001.pdf> (last checked 1/31/2013)
9. Hockemeyer, C., Held, T., Albert, Rath, D.: A relational adaptive tutoring hypertext WWW-environment based on knowledge space theory. *Proceedings of CALISCE'98*, 4th International conference on Computer Aided Learning and Instruction in Science and Engineering, Goeteborg, Sweden, pp. 417–423 (1998)
10. Brusilovsky, P. Sosnovsky, S.: Individualized Exercises for Self-Assessment of Programming Knowledge: An Evaluation of QuizPACK, *ACM Journal on Educational Resources in Computing*, 5 (3) (2005)
11. Hsiao, I., Brusilovsky, P., Sosnovsky, S.: Web-based Parameterized Questions for Object-Oriented Programming. In: J. Nall and R. Robson (eds.) *Proceedings of World Conference on E-Learning, E-Learn 2008*, Las Vegas, USA, Nov. 17–21 (2008)
12. Kölling, M., Rosenberg, J.: An object-oriented program development environment for the first programming course. *SIGSE Bulletin*, 28 (1), pp. 83–87 (1996)
13. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M.: *Scratch: A Sneak Preview*. Second International Conference on Creating, Connecting, and Collaborating through Computing. Kyoto, Japan, pp. 104–109 (2004)
14. Djouad, T., Mille, A., Reffay, C., Benmohammed, M.: A new approach based on modelled traces to compute collaborative and individual indicators' human interaction. In *Proceedings of the 10th IEEE International Conference on Advanced Learning Technologies*, Sousse, Tunisie (2010) <http://liris.cnrs.fr/Documents/Liris-4768.pdf> (last checked 1/31/2013)
15. Hsiao, I-H., Brusilovsky, P.: Motivational Social Visualizations for Personalized E-learning. In: *Proceedings of 7th European Conference on Technology Enhanced Education (ECTEL)*, ECTEL 2012, Saarbrücken, Germany, September 18–21, 2012, Springer-Verlag, Volume 7563/2012, pp. 153–165 (2012)



# Computational Thinking in Dutch Secondary Education

Nataša Grgurina

Teacher Education, Faculty of Behavioural and Social Sciences, University of Groningen  
P.O. Box 800, 9700 AV Groningen, The Netherlands  
n.grgurina@rug.nl

**Abstract.** We shall examine the Pedagogical Content Knowledge (PCK) of Computer Science (CS) teachers concerning students' Computational Thinking (CT) problem solving skills within the context of a CS course in Dutch secondary education and thus obtain an operational definition of CT and ascertain appropriate teaching methodology. Next we shall develop an instrument to assess students' CT and design a curriculum intervention geared toward teaching and improving students' CT problem solving skills and competences. As a result, this research will yield an operational definition of CT, knowledge about CT PCK, a CT assessment instrument and teaching materials and accompanying teacher instructions. It shall contribute to CS teacher education, development of CT education and to education in other (STEM) subjects where CT plays a supporting role, both nationally and internationally.

**Keywords:** computational thinking, situated learning, engaged computing, computer science

## 1 Present Situation

In 2006, J.M. Wing asserted that “to reading, writing, and arithmetic, we should add computational thinking to every child’s analytical ability” [6]. Educators recognize this need and inquire into the precise description of this concept and the ways to teach it. In 2010 in the United States, the National Research Council held a workshop on the nature and scope of Computational Thinking (CT). While there was a broad consensus on the importance of (teaching) CT, the workshop did not achieve a conclusive definition of this concept [4]. The Computational Thinking Task Force of CSTA did however suggest an operational definition of CT tailored to the needs of K-12 education. They state that:

*CT is a problem-solving process that includes (but is not limited to) the following characteristics:*

- *Formulating problems in a way that enables us to use a computer and other tools to help solve them*
- *Logically organizing and analyzing data*

- *Representing data through abstractions, such as models and simulations*
- *Automating solutions through algorithmic thinking (a series of ordered steps)*
- *Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources*
- *Generalizing and transferring this problem-solving process to a wide variety of problems*

*These skills are supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:*

- *Confidence in dealing with complexity*
- *Persistence in working with difficult problems*
- *Tolerance for ambiguity*
- *The ability to deal with open-ended problems*
- *The ability to communicate and work with others to achieve a common goal or solution [2]*

In the USA and the UK, teaching these CT problem solving skills does not get enough attention from policy makers and is hardly represented in school curricula [5]. In the Netherlands, the situation is similar [1]. The Computer Science (CS) course in the upper grades of secondary education seems to be a natural setting to introduce CT teaching. In this course students regularly collaborate on large practical assignments based on realistic problems in order to produce working solutions. For example, a typical assignment would require students to design a model of traffic lights for a busy crossing or elevators for an apartment building. Both these problems:

- Are open and can have various correct solutions;
- Come with a minimal specification, prompting the students to investigate the behavior of the system to be modeled and thus encouraging them to use higher order thinking skills;
- Originate from the *real world*: solving (modeling) them necessitates the use of information-processing agents – like those used within a CS course.

## 2 Research

Obviously, a number of aspects of CT can be recognized in regular CS teaching practice, albeit lacking coherence and not being explicitly specified as learning objectives in the CS curriculum. Therefore, we propose to conduct design research concerning the teaching and learning of Computational Thinking (CT) within the context of a CS course in upper secondary education in the Netherlands. This research will take four to six years and will result in a PhD dissertation.

## **2.1 Research Questions**

We shall study the following issues:

1. What is an operational definition of Computational Thinking, tailored to the specific situation and needs of secondary education in the Netherlands?
2. How can students' CT problem solving skills be assessed?
3. What is a suitable pedagogical approach to teaching students and stimulating their learning of CT problem solving skills?

## **2.2 Method**

In this research, a curriculum intervention (pedagogical approach, teaching materials and teacher education materials) and an appropriate CT assessment tool will be developed as a result of an iterative (cyclic) process.

The first phase of the research is dedicated to obtaining an operational definition of CT. In the second phase, an instrument for the assessment of students' CT will be developed. The results of these two phases will yield the data for the pedagogical approach that will be developed in the third phase of the research: a curriculum intervention for students as well as accompanying teachers' instructions will be developed and tested in a pilot. In the fourth phase, the effects of the curriculum intervention will be assessed in an experiment on a larger scale and the final version of curriculum intervention (i.e. teaching materials) will be developed.

### **Phase 1**

Essential aspects of CT will be described, based on the CSTA definition of CT, existing teaching materials and additional literature. This draft definition will be presented to a number of experienced CS teachers. Their pedagogical content knowledge (PCK) pertaining to aspects of CT described in the draft definition will be established using content representation framework [3]. This will yield a final operational definition of CT tailored to the needs of CS course in Dutch secondary education.

### **Phase 2**

The CT description obtained in phase 1 will be used to develop an instrument for the assessment of students' CT problem solving skills. After consulting with experts, both nationally and internationally, and necessary modifications the author will test this instrument for usability and reliability in her own classroom while teaching CS using suitable sections of regular teaching materials. The findings will lead to final adjustments of this instrument. At the same time, the students' learning will be observed through video and sound recordings of individual and collaborative work, semi-structured interviews and other qualitative methods. Special attention will be paid to the difficulties students experience, misconceptions, use of CT skills and visibility of problem solving strategies.

### **Phase 3**

The findings of the phases 1 and 2 will contribute to the development of teaching materials for students and instructions for teachers (in the form of a course for teachers). The design of students' materials will be based on the idea of working with concrete problems in real world situations (situated learning). The curriculum intervention includes programming and modeling using freely available software.

After consultations with national and international experts (and possible modifications), these teaching materials will be tested by a small number of teachers who will report their experiences and findings through questionnaires and interviews. This will lead to further adjustments of the teaching materials.

### **Phase 4**

The experiment will take place in some dozen schools. Using the CT assessment instrument mentioned earlier, the effect of curriculum intervention (i.e. teaching with newly developed teaching materials and an associated pedagogical approach) on students' CT problem solving skills will be assessed and compared to CT problem solving skills of the students in control groups who were not taught CT problem solving skills explicitly. This will also lead to further adjustments of teaching materials.

## **3 Results**

There has not been much research on (the effects of) teachers' instructions on CT problem solving skills of their students. This research will provide an assessment tool to make students' learning of CT visible, together with a validated assessment instrument to measure students' computational thinking. Furthermore, teaching materials for students and accompanying teachers' instructions will be developed.

Since CT can be viewed as a form of situated learning, the results of this research will be interesting for scientific research into situated learning in related STEM subjects.

New insights into teaching and learning CT will help teachers to prepare their students more adequately for life in our modern society and for effectively applying CT professionally, regardless of whether ICT plays a central role in their occupation or not. Teacher training departments will be equipped better to prepare future (CS) teachers for their jobs.

Besides the contribution of this research to the growth of the body of CS pedagogical content knowledge in general, it will mean to the author a further development of the pedagogical approach she has been practicing in her classes for years and which has convinced her more and more that that CS is not an isolated art but one that facilitates learning and understanding in other disciplines.

## References

1. Barendsen, E., Zwaneveld, B.: Informatica in het voortgezet onderwijs - voorstel voor vernieuwing (Notitie voor KNAW) (2010)
2. CSTA, <http://csta.acm.org/Curriculum/sub/CompThinking.html> (last checked 1/31/2013)
3. Loughran, J., Berry, A., Mulhall, P.: Understanding and developing science teachers' pedagogical content knowledge (2006)
4. National Research Council (US). Committee for the Workshops on Computational Thinking. Report of a workshop on the scope and nature of computational thinking, Natl Academy Pr. (2010)
5. Schnabel, R. B.: Educating computing's next generation. *Communications of the ACM*, 54(4), p. 5 (2011)
6. Wing, J. M. Computational thinking. *Communications of the ACM*, 49(3), pp. 33–35 (2006)



## **Different Views**





# A Model for Teaching Informatics to German Secondary School Students in English-language Bilingual Education

Martin Weise (né Reinertz)

Reinoldus- und Schiller-Gymnasium, Dortmund (Germany)

**Abstract.** Informatics as a school subject has been virtually absent from bilingual education programs in German secondary schools. Most bilingual programs in German secondary education started out by focusing on subjects from the field of social sciences. Teachers and bilingual curriculum experts alike have been regarding those as the most suitable subjects for bilingual instruction – largely due to the intercultural perspective that a bilingual approach provides. And though one cannot deny the gain that ensues from an intercultural perspective on subjects such as history or geography, this benefit is certainly not limited to social science subjects. In consequence, bilingual curriculum designers have already begun to include other subjects such as physics or chemistry in bilingual school programs. It only seems a small step to extend this to informatics.

This paper will start out by addressing potential benefits of adding informatics to the range of subjects taught as part of English-language bilingual programs in German secondary education. In a second step it will sketch out a methodological (= didactical) model for teaching informatics to German learners through English. It will then provide two items of hands-on and tested teaching material in accordance with this model. The discussion will conclude with a brief outlook on the chances and prerequisites of firmly establishing informatics as part of bilingual school curricula in Germany.

## 1 Setting course for informatics in English-language bilingual education

### 1.1 The situation as is

At the time of this paper, informatics as a school subject is virtually absent from bilingual<sup>1</sup> curricula in German secondary education. According to ([10], p. 96) there are merely one “Gymnasium” and one comprehensive school that offer bilingual informatics classes in North Rhine-Westphalia, the most populous German state. The latter of these schools even limits the classes to one hour of instruction per week. There are various reasons for this apparent reluctance to take up informatics into the range of subjects taught through English:

---

<sup>1</sup> As the title suggests, the scope of this paper is limited to bilingual programs in Germany that use English as the language of instruction.

- a dramatic shortage of informatics teachers with professional qualifications in both informatics and English
- a lack of teaching materials
- informatics classes are not compulsory in most German states

However, the low prevalence of informatics in bilingual school curricula may also create the impression that *informatics as a school subject* does not lend itself very well to a bilingual approach per se. On the contrary, as one of its key points, this paper will strongly contradict this view. It will propagate informatics as one of the subjects best suited for bilingual education. The discussion will start with a brief glance at the characteristics of secondary school informatics education. It will then move on to propose a conceptual framework for bilingual informatics education in secondary schools. And finally, the paper will provide two examples of hands-on teaching materials that enable a first glimpse at what bilingual informatics classes could look like. Plus, it might well be a motivation for informatics teachers to start using bilingual elements in their teaching.

## 1.2 The “why” of bilingual informatics education

Most bilingual programs in German secondary education started out with a strong focus on social science subjects. Teachers and bilingual curriculum experts traditionally regarded those as most suitable for bilingual instruction (cf. [6]). This was largely due to the intercultural perspective that a bilingual approach provides.

It is no surprise that, with globalization at its fullest, this limited perspective has recently come under somewhat heavy attack. The interconnected world has continually been placing greater demands on people’s control of a lingua franca such as English. Education experts and school administrators alike have begun to see the value of other subjects taught through a foreign language. Many publications from the area of bilingual education focusing on the natural sciences (such as [13]) have explored four main lines of argument:

1. *The main language of academic discourse* in the subjects under consideration is English. High-quality translations to other languages take time – which means that secondary school students only have access to recent findings/publications if they have advanced English skills which are linked with content knowledge.
2. With regard to *subject-specific terminology*, there are usually only small differences between German and English. This is because both versions are often rooted in either Latin or ancient Greek.
3. New findings/developments in the natural sciences also have a *cultural dimension* (as much or maybe today even more so than in the social sciences).
4. *English-language publications* (such as US and UK textbooks) are often much more geared towards accessibility than their German-language equivalents. Hence they are a valuable resource for learners in Germany. Yet their classroom use places high demands on the students’ English-language skills.

These points are certainly also valid for informatics education, but this is not the whole story. There is an even stronger argument that does not merely ‘qualify’ informatics as a school subject fit for bilingual approaches. In fact, it makes it seem strange that there

is so little tangible interest in bilingual informatics classes: namely that it is hard to conceive of informatics education in a complete absence of English. Why is that?

Major parts of what students have to grapple with right from the start when taking informatics classes are only accessible through English. Quite frequently they *must* understand *technical specifications* if they want to be successful learners. For instance, learners often need to understand and make use of original API documentations. They are supposed to gain an understanding of established network protocols such as RFC 868, which has even made it into the *Standards for informatics Examinations* (cf. [8], pp. 54 ff) in its original (i.e. English) version. And even today's centralized A-level requirements make explicit reference to English-language versions of data structures and abstract data types. Hence students need advanced English language skills integrated with content (= informatic) knowledge, especially if classrooms are to be student-centered. On a different note even a brilliant teacher could not possibly provide a translation of every such item. What this means is that English and its content-based application are *already* a crucial element in informatics classrooms.

If one takes these aspects just a step further, a new dimension of informatics in bilingual education opens up: programming *languages* – including those used in German classrooms – are also *languages*. And they are usually based on English keywords. This applies to control structures (if, while, for, ...) and other linguistically-anchored identifiers (public, class, ...). Plus, there is usually a strong relation between the functions these tokens exhibit in the programming language and the meaning of these words in the English language. Consequently, one of the most interesting questions to investigate is whether there is a correlation between a high degree of competence in English and a reasonable understanding of basic constructions in programming languages. Unfortunately there is a dramatic lack of research in this area, but what little we know about natural language acquisition makes it seem improbable that there is no such connection at all. If the answer is positive, this may be the key to reducing common misconceptions in informatics rooted in everyday language (e.g. when students talk about “if-loops” (if-Schleifen) instead of “if-statements” (if-Abfragen) – something that occurs quite frequently in German-language informatics classes and requires much effort and time to eradicate (cf. [7], p. 28).

It is a logical conclusion that high-quality informatics instruction is necessarily – at least in part – bilingual. This is even true if it takes place outside any official bilingual school curriculum. It is thus hard to imagine any subject better suited to a bilingual approach. After all, *language* is one of the most fundamental concepts in informatics instruction (cf. [5]) – as it is in language teaching. This reflects in much of the core content informatics lessons revolve around (modeling *languages*, programming *languages*, formal *languages*).

### **1.3 Two-fold relevance? An unsubstantiated hypothesis based on mathematics and the natural sciences**

While a vast body of research in the field of bilingual education suggests that a student's general cognitive abilities, cultural competencies, and foreign language skills improve through bilingual education, this paper has not yet addressed the potential relevance for informatics as a school subject – and thus a student's informatic competencies. The bad

news first: There is virtually no present research on bilingual informatics education and hence no reliable data to support the claim that teaching informatics in any language other than a student's L1<sup>2</sup> is also relevant to informatics education itself.

However, there is such evidence for various other subjects from the field of natural sciences (cf. [13], p. 116 f) or mathematics (cf. [14], p. 31 f.). The sources point out that bilingual education, on the one hand, prepares students for work in these fields after they graduate. This is due to the fact that the aforementioned areas actually require both applicants and professionals to have subject-specific foreign language competencies, mostly in English (cf. [6], p. 3). On the other hand (and this is more on the didactical side of the benefit) bilingual education provides a second point of access to abstract subject-specific knowledge through the foreign language:

“[...] the availability of both languages in bilingual mathematics education has a positive effect on the acquisition of mathematical competencies; bilingual education in mathematics can thus benefit from relying on two languages.”<sup>3</sup>(cf. [14], p. 33).

If this is true of mathematics classrooms, why would it not be true of informatics education in schools? In fact, informatics and mathematics are highly interconnected and therefore often similar with regard to the level of abstraction they demand of students. What is more, informatics, like mathematics, uses symbolic codes of representation that students must first access through natural language. If it is beneficial in mathematics that the subject is taught in a second language, why not draw the same conclusion as far as informatics is concerned – in particular against the background of the observations in section 1.2? The author does realize the hypothetical nature of this claim, which results from the lack of research in the field. Unfortunately though, a thorough exploration would go beyond the scope of this paper, whose primary aim is a first attempt at providing a didactical teaching model for bilingual education in informatics.

## 2 The Teaching Model

The following is an outline of a methodological framework for teaching informatics in the bilingual classroom. It is based on a model developed by J. Leisen, which proposes a “change of representation forms” as a general instruction principle for all subjects in bilingual education. Although Leisen considers his approach equally suitable for all kinds of subjects, his own examples are mostly from the field of natural sciences:

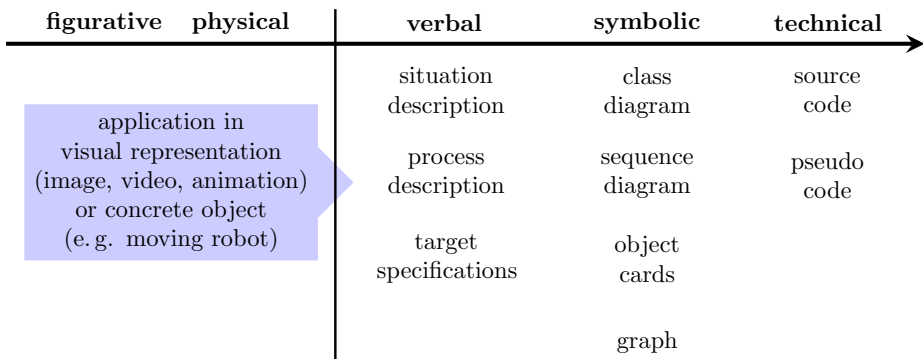
“It is crucial for school subjects to present their facts in various ways. These forms of representation differ with respect to their levels of abstraction. A

---

<sup>2</sup> In language acquisition theory, L1 commonly refers to a person's native language whereas L2 stands for a language (usually a foreign one) learned later on in life.

<sup>3</sup> Translated from the German: “Hiermit kommt zum Ausdruck, dass das Heranziehen beider Sprachen sich im Rahmen des bilingualen Mathematikunterrichts positiv auf den Erwerb mathematischer Kenntnisse auswirkt; der bilinguale Mathematikunterricht kann sich vorteilhaft auf zwei Sprachen stützen.” (Ibid)

constant change from one form to another is the methodological key to ‘understanding’ in a subject and it can serve as an incentive for subject-related discussion. It is methodologically wise to make this change of representation forms the focus of bilingual teaching methodology.”<sup>4</sup> ([9], p. 10)



**Fig. 1.** Various forms of representation in informatics and their increasing levels of abstraction (based on [9], p. 2)

Bilingual lessons designed on the foundations of this approach will exhibit a constant (and usually progressive) shift in abstraction levels with regard to how facts are dealt with in class (cf. [9], p. 11): *The starting point* is usually a *physical* representation of a piece of content (e.g. through a concrete object, an experiment, or actions). → *In a second step*, the teacher or his students ‘translate’ this item into a *figurative* form of representation (such as an image, a pictogram, a schematic drawing, etc.) → *with the aim* of providing a precise (written or oral) *verbal* description . → *This description* then serves as a basis for a *symbolic* form of representation (object diagrams, Nassi-Shneiderman diagrams, graphs, etc.) → *before* it is eventually transposed into a *mathematical/formula-based* form of representation.

This model contains *all* major aspects of a bilingual teaching methodology: content learning, language learning, and foreign language learning. The foreign language serves as a *language of learning*, *language for learning*, and *language through learning* (cf. [3], p. 36) which is in line with the Content and Language Integrated Learning (CLIL) approach to bilingual education (cf. Ibid).

Informatics as a school subject is ideally suited for the approach just described. This is largely owed to the core element of informatics education: modeling. Curriculum

<sup>4</sup> Translated from the German: “Es ist grundlegend für die Sachfächer, dass sie die Sachverhalte [...] in verschiedenen Darstellungsformen darstellen [...]. Diese liegen auf Ebenen unterschiedlicher Abstraktion. [...] Der Wechsel [...] erweist sich als der didaktische Schlüssel zum fachlichen Verstehen und ist ein Anlass zur fachlichen Kommunikation. Es ist didaktisch klug, [...] diesen Wechsel der Darstellungsformen in das Zentrum der Didaktik des bilingualen Sachfachunterrichts zu stellen” (Ibid)

experts agree that modeling is so essential (because it is key in gaining informatics literacy) that it must not be omitted from informatics education in schools. In fact, it should be omnipresent as its very foundation (cf. [15]).

No matter what the content of a specific lesson, the underlying progression is therefore always similar:

- (1) Starting out from a real-world problem/phenomenon
- (2) students follow various modeling steps
- (3) to create an informatics system that solves the problem under consideration.

Hence, modeling is the very link between real-world and informatics systems. This is particularly evident in lesson series on *object-oriented programming* or *finite-state machines*, which commonly follow the three-phase process laid out above.

But from a scientific perspective *modeling* is nothing other than a change of representation forms. One might thus argue that a “change of representation forms” is a fundamental informatic principle per se – whereas it may be a mere approach to teaching in other subjects. Against this background, Fig. 1 suggests itself as the *informatics version* of the process diagram published in [9]. The diagram shows various forms of representation in connection with their levels of abstraction.

Two aspects of this diagram require further attention:

1. From the “verbal” abstraction level onward there are only explicitly computer-scientific forms of representation. These are well-known concepts in informatics anyway.
2. It is not necessary for learners to always start at the lowest level of abstraction. Once learners have gotten used to how real-world situations can be adequately described via language, teachers can slightly neglect lower levels and supply a *verbal* representation as the starting point – possibly with some kind of visual support. However, one should keep in mind that the translation from a figurative form of representation into a verbal one is modeling as well.

### 3 Ideas for teaching

#### 3.1 Object diagrams

The following is an English translation of a (previously German) *situation description* from [1]. It might serve as the starting point for a lesson on object-oriented modeling.

“Bookworm” – An Online Bookstore

“Bookworm” currently has a special offer on three books:

- “Lord of the Rings”, author: J. R. R. Tolkien, ISBN: 978-0261102385, price: \$32.50
- “Angels & Demons”, author: Dan Brown, ISBN: 978-0552150736, price: \$6.45
- “Objects First with Java”, author: David J. Barnes, ISBN: 978-0132835541, price: \$40

There are 10 remaining copies of “Lord of the Rings”, 4 remaining copies of “Angels & Demons”, and 7 copies left of “Objects First with Java”.

Martin and Stephanie each have an account with “Bookworm”. Martin’s account number is 123-45A-X23 and his account balance is \$300. Stephanie’s account balance is \$125. Her account number is 123-45A-X25.

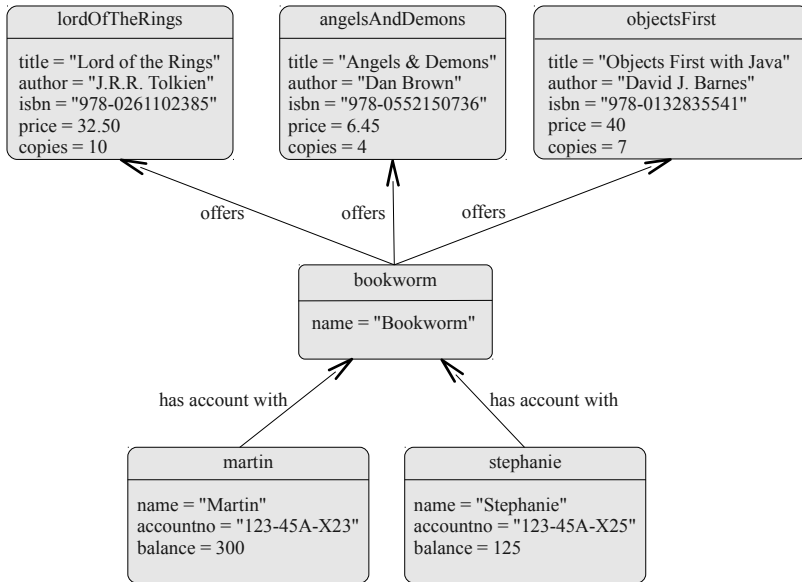
The basic idea is to provide students with the textual description and instruct them to use *Abbott’s approach*<sup>5</sup> (a reduced variant is known as the *subject-verb method*) in order to elicit objects (including potential attributes and their values) from the written input. More advanced students can already go on to compiling their *object cards* into an *object diagram*. Of course, a teacher might want to acquaint his students with both *Abbott’s approach* and the above *situation description* at the same time. But for now let us just assume that the learners under consideration are well-acquainted with the technique. With prior experience, students might arrive at a model similar to the one depicted in Fig. 2.

The goal behind using *Abbott’s approach* in teaching is for students to transform verbal forms of representation into symbolic ones as they match word classes to items in an object-oriented model (cf. 2). This happens as they perform an analysis of word classes on an informal written description. Against the background of bilingual education, this process seems particularly promising:

- Textual descriptions use natural language. Consequently, students should understand them via the basic communication skills and knowledge acquired in their “English as a foreign language” classes. There is no need for additional, subject-specific language instruction.
- *Abbott’s approach* helps to integrate content and language learning. As students transform textual descriptions into *object diagrams*, they need to acquire and use computer-scientific language. Terms such as “object”, “attribute” and “attribute value” are so similar to their German equivalents that there is a high likelihood of positive transfer between the students’ L1 and L2.

---

<sup>5</sup> The *subject-verb-method* involves developing a conceptual (object-oriented) model from a textual description in natural language. It focuses on “the use of nouns and noun phrases as references in natural language [...]. Proper nouns and references suggest objects.” (cf. [2]). At a later point, it attends to verbs (usually in a more elaborate *process description*), which yield potential methods (cf. *ibid*).



**Fig. 2.** A possible *object diagram* derived from the situation description (as suggested in [1]).

- *Abbott’s approach* is helpful to extract objects from written descriptions (especially since there are few other techniques to facilitate this process), but it is certainly far from being a formal method (cf. [4], ch. 4). This is mainly because real-life situations (or rather descriptions thereof) are usually much more complex than is permissible for the approach. There is often a plethora of different, but valid models for one and the same textual description. Instead of posing a problem, this is actually an advantage for bilingual classrooms: models thus derived are an opportunity for classroom discussion involving general as well as subject-specific language.

### 3.2 Lego Mindstorms NXT robots with Java

The task below is from the beginning of my lesson series on Lego NXT robot control via Java. It is part of a worksheet I have successfully used in 9th grade informatics classes. As a prerequisite, students must be acquainted with the basics of operating systems and standard applications (such as directory structures and opening/saving files). For step 1 of the task I usually set up an NXT brick with a small program which makes it travel 30cm in a straight line and then perform a right-angle turn.



### *Travel and turn*

Robots cannot (yet) do everything humans can do. But they are pretty good (i.e. fast, accurate, and they do not get tired!) at carrying out repetitive actions. Let us take a look at some of the basic actions robots can perform.

1. Observe your teacher's NXT robot. Take exact notes about the actions it carries out.
2. Download `example.zip` and unpack it. Open the project using *BlueJ*.
3. Double-click the `MoveShape` class and look at the source code. Try to figure out which of the commands have made the robot do what it did. – Then upload the program and try it on your own robot.
4. Alter the program so that your NXT robot will travel 50cm instead of just 30.
5. Make your robot travel a square with a *diagonal* of 30cm. Then make your robot travel a triangle with a *side length* of 30cm.

The relevant part of the source code from `example.zip` looks like this:

```
1 public void go ()
2 {
3     //These commands tell your robot what to do:
4     gear.travel(300);
5     gear.rotate(90);
6 }
```

As far as the bilingual teaching model laid out in this paper is concerned, the most interesting aspect is that students must change between forms of representation numerous times to complete the steps above. They start out by observing a real *model* and have to describe its actions in standard *written language*. Then they move on to drawing connections between their observations/descriptions and the *source code* handed out to them. One might, of course, also provide another intermediate layer in the form of a Nassi-Shneiderman diagram for the robot's actions before moving on to the source code. However, my observations from the classroom suggest that this is not necessary. As a last step, students must *write their own source code* by making slight alterations to the code provided. In subsequent lessons I then continue with the same approach and quickly extend the set of Java commands my students analyze and use to include if-statements and loop constructs.

Furthermore, it is not difficult to add an ethical (and possibly also cultural) dimension: For instance, I usually have my students reflect on potential dangers the initial program might pose if their NXT robot was an industrial robot that weighs several tons. Without much difficulty, my students arrive at the conclusion that there are no safety mechanisms so far and that any unfortunate person happening to cross the robot's path would suffer severe injuries. Hence my students develop an intrinsic understanding of why sensors are crucial elements of robots and how they could affect a robot's behavior.

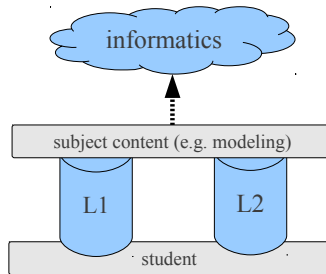
The same can be done if one puts emphasis on the effects simple robots (might) have had on the labor market.

Scholars in the area of bilingual education have been regarding such cross-references as particularly beneficial. This is not only because they provide ample opportunity for discussion, but also because they provide an opportunity for students to gain a broader perspective on subject-related aspects with regard to their cultural and ethical implications. A well-designed task should consequently contain potential for both, an ethical/cultural perspective *and* insights into new subject-specific content.

#### 4 Conclusion and the Role of L2 Competence

This paper has laid out in English the major points I already made in German language in [11] and [12] about informatics classes in bilingual education. In addition, it provided support for its main argument from teaching methodologies in the natural science subjects and introduced new teaching materials in the context of robotics. Although the discussion in this paper is yet far from being a full-fledged theory, it does indicate that it is a promising idea to include informatics among the range of subjects taught in English.

As laid out in section 1.3, there is reason to believe that including the students' L2 (i.e. English) in teaching informatics to German learners will help to reduce subject-related misconceptions and ease understanding of informatic concepts. This can be aptly illustrated by Fig. 3, which was inspired by a diagram proposed by Schubnel depicting the contribution of L2 competence to bilingual education in mathematics ([14], p. 33). The idea behind this is that the students' L2 (English) can serve as a crutch (in addition to their L1!) in understanding subject content. In other words, it serves the purpose of providing a second column on which informatic competence can build up and rest, since learning in informatics necessarily takes place through natural language. Furthermore, the L2 at hand is the language of almost any workplace in the field of informatics.



**Fig. 3.** Supposed role of L2 in bilingual informatics education

However, there are at least two major problems: Most German informatics teachers quite naturally lack C2<sup>6</sup> (or at least C1) qualification in English, whereas most teachers of English as a foreign language lack informatics literacy. Moreover, with most of the assumptions in this paper about the beneficial effects of high L2 competence being based on mathematics and the natural sciences, scholarly research into the true contribution of L2 competence to understanding in *informatics* is a crucial next step.

## References

1. Arbeiter, T.: Objektorientierte Modellierung und ihre Umsetzung inklusive Prüfung der erreichten Kompetenzen. Hausarbeit gemäß OVP, Studienseminar für Lehrämter an Schulen – Seminar für das Lehramt für Gymnasien und Gesamtschulen, Arnshausen (2008)
2. Boyd, N.: Using Natural Language in Software Development. <http://www.educery.com/papers/rhetoric/road/> (last checked 1/31/2013)
3. Coyle, D., Hood, P., Marsh, D.: CLIL. Content and Language Integrated Learning. Cambridge University Press, 3rd, 2011 edn. (2010)
4. Graham, I., Wills, A.: UML Tutorial. <http://uml-tutorials.ti-re.me.com/> (last checked 1/31/2013) (2001)
5. Gramm, A.: Entwurf eines bilingualen Sachfachunterrichts Informatik (2002), [http://andreasgramm.de/papers/Gramm\\_Bilingualer\\_Sachfachunterricht\\_Informatik.pdf](http://andreasgramm.de/papers/Gramm_Bilingualer_Sachfachunterricht_Informatik.pdf) (last checked 1/31/2013)
6. Hallet, W.: Bilingualer Unterricht: Fremdsprachig denken, lernen und handeln. Der fremdsprachliche Unterricht Englisch 78, 2–8 (2005)
7. Humbert, L.: Didaktik der Informatik – mit praxiserprobtem Unterrichtsmaterial. Leitfäden der Informatik, B.G. Teubner Verlag, Wiesbaden, 2nd edn. (August 2006), <http://humbert.in.hagen.de/ddi/> (last checked 1/31/2013)
8. Kultusministerkonferenz: Einheitliche Prüfungsanforderungen Informatik (1989, idF von 2004)
9. Leisen, J.: Wechsel der Darstellungsformen. Ein Unterrichtsprinzip für alle Fächer. Der fremdsprachliche Unterricht Englisch 78, 9–11 (2005)
10. Ministerium für Schule und Weiterbildung NRW: Das Schulwesen in Nordrhein-Westfalen aus quantitativer Sicht 2010/11. Statistische Übersicht 373 (2011)
11. Reinertz, M., Humbert, L.: Bilingualer Informatikunterricht. In: Thomas, M., Weigend, M. (eds.) Ideen und Modelle. 5. Münsteraner Workshop zur Schulinformatics (2012)
12. Reinertz, M., Humbert, L.: Informatik bilingual. In: Hallet, W., Königs, F.G. (eds.) Handbuch Bilingualer Unterricht/Content and Language Integrated Learning. Klett/Kallmeyer (2012)
13. Richter, R., Zimmermann, M.: Biology. Und es geht doch: Naturwissenschaftlicher Unterricht auf Englisch. In: Wildhage, M., Otten, E. (eds.) Praxis des bilingualen Unterrichts. Cornelsen Scriptor, 3rd edn. (2009)
14. Schubnel, Y.: Bilingualer Mathematikunterricht. Ein Beitrag zu einem zusammenwachsenden Europa. Ph.D. thesis, Pädagogische Hochschule Freiburg (2009)
15. Thomas, M.: Informatische Modellbildung – Modellieren von Modellen als ein zentrales Element der Informatik für den allgemeinbildenden Schulunterricht. Dissertation, Universität Potsdam, Didaktik der Informatik (2002), [http://ddi.cs.uni-potsdam.de/Personen/marco/Informatische\\_Modellbildung\\_Thomas\\_2002.pdf](http://ddi.cs.uni-potsdam.de/Personen/marco/Informatische_Modellbildung_Thomas_2002.pdf) (last checked 1/31/2013)

---

<sup>6</sup> The term C2 refers to the highest possible competence level (“mastery or proficiency”) in a language as defined in the Common European Framework of Reference for Languages. C1 level is slightly below C2.



# What you see is what you have in mind: constructing mental models for formatted text processing

Carlo Bellettini, Violetta Lonati, Dario Malchiodi, Mattia Monga,  
Anna Morpurgo and Mauro Torelli

Dipartimento di Informatica  
Università degli Studi di Milano  
Milan, Italy

{bellettini, lonati, malchiodi, monga, morpurgo,  
torelli}@di.unimi.it

**Abstract.** In this paper we report on our experiments in teaching computer science concepts with a mix of tangible and abstract object manipulations. The goal we set ourselves was to let pupils discover the challenges one has to meet to automatically manipulate formatted text. We worked with a group of 25 secondary-school pupils (9-10th grade), and they were actually able to “invent” the concept of mark-up language. From this experiment we distilled a set of activities which will be replicated in other classes (6th grade) under the guidance of math teachers.

## 1 Motivation

As repeatedly pointed out in the computer science education literature [13, 5–7], the computing discipline is often confused with literacy in basic computer applications. This is particularly evident in the context of general (non-vocational) education; for instance, in Italy, computer science teaching from 6th to 10th grade is commonly limited to the use of office automation tools (like word-processors) or communication applications (web and email).

Nevertheless, computational thinking has the potential to be very formative for pupils, and several core aspects of computing are sufficiently basic to be taught as a fundamental subject [1, 10, 11, 15]. However, two main hurdles have to be considered in introducing fundamental computing concepts in K-12 education:

1. the *abstract nature* of computing, which makes it difficult to *illustrate* the concepts by referring to physical objects;
2. the *expectations* of students, who need clear links between learning about computing and using a computer as a productivity tool, activity to which they are exposed in their daily lives.

The basic use of computers can be exploited as a chance to explore computational concepts. For instance, even the basic use of a word-processor cannot be effective unless the user has acquired some sort of interpretation model of how the application tool is structured and how it works [4]. In fact, such models are usually deduced by a *bricolage* (trial-and-error) approach which often leads to misconceptions and ambiguities that prevent users from understanding the application correctly and using it

effectively [3]. Teaching such models explicitly would also provide an introduction to some fundamental computer science concepts, for instance the digital representation of information or the need of unambiguous formal description of procedures that can be executed automatically.

We tried to implement such ideas starting from one of the most common computer activities in class (the use of a word processor), and designing a sequence of learning activities with the aim of making pupils aware of the challenges posed by the automatic processing of formatted text, mainly focusing on information representation issues.

A relevant part of this didactic sequence consists of a type of kinesthetic/tactile learning activities [2] which we have called *algotricity*. These activities are then followed by more abstract ones implying the use of symbolic meta-languages. All the activities are designed to push pupils to experiment with the problem of text representation and explore suitable techniques to represent both text and formatting. In order to close the loop from known applications to abstractions, and back to computers, the sequence ends with the practice of such techniques again on a computer.

The approach is inspired by *constructivism* [4] and the *allosteric learning model* [9], which point out that the direct transmission of knowledge is rarely the best way for learning. On the contrary:

“Learning is a highly active process which works in a conflicting way and in an integrative mode between what the learner has in his mind and what he can find and understand through his conceptions on his environment. When a learner elaborates a new model, all his mental model[s] must be reelaborated in an interaction between the conception and the external information. [...] Most of the time they need a didactic environment to cause interferences with their sitting conceptions. And this is the main function[s] of the teacher: he has to propose or to suggest an heuristic environment that may interfere with the learner conceptions.” [9]

That is exactly what we meant in designing our activities, where pupils are put in new situations with a goal to reach and few or no hints about a-priori criteria or techniques to refer to.

Notice that the general goal of our didactic proposal is not to teach how to use the application but to let the pupils discover some of the principles its designer had in mind. In fact, this can be both useful to improve one’s proficiency in using the application and – above all – to understand the challenges of digital information processing.

We started by experimenting with this approach within a group of 25 pupils in 9th and 10th grades of an Italian secondary school. In Section 2 we detail the activities we proposed to pupils, in Section 3 we report some remarks and considerations on how such an experiment was designed and implemented, and in Section 4 we draw our preliminary conclusions and try to prepare the ground for future work.

## 2 Description of the activities

The overall activity (8 hours in 4 non-consecutive days) was organized in four phases:

1. familiarization with text formatting on a word processor;
2. a *dramatization* of the process through the use of tangible objects;
3. a game designed to force pupils to restructure their mental models and discover the power of symbolic meta-languages;
4. a final use of special software tools for formatting texts, which could also show the data structure used to record the meta-information.

We started and ended off in a computer lab, in order to partially match pupils' expectations about informatics [12]. The risk we wanted to avoid was that of being perceived as unrealistic or fruitlessly "academic". Our goal was to let pupils be confronted with the challenges of dealing with formatted digital text, and we designed the activities around two main ideas: (a) computers and software tools should be of secondary importance, but the conceptual link with them should be clear; (b) the approach should be mostly *allosteric* [9]: the direct transmission of knowledge should be kept to a minimum, and pupils should be forced to reconsider their mental models of text formatting by discovering useful techniques by themselves. Besides, the abstract nature of computing should be conveyed by concrete examples and physical activities.

Pupils were asked to work in small groups to encourage this confrontation with the challenge. Moreover, almost every task was proposed together with an accompanying meta-cognitive reflection. In particular we often asked pupils to imagine how their descriptions would be understood by someone who was unaware of the context.

### 2.1 Basic text formatting

We started in a computer lab and the first goal was to introduce basic formatting elements such as italics, bold, etc. The pupils already had some knowledge about "formatting", but they had rarely reflected on its semantics. Thus, we discussed with the pupils *the role of formatting as a conveyor of information*.

The pupils (working in pairs) were requested to produce a formatted text of their choice. Then they had to answer some questions about their work: Which type of formatting did you use? Why did you choose it? Did you use more than one formatting for the same text passage? They first mainly noted the aesthetic value of formatting, but by reflecting on how they had used formatting in a text, they also noted that it plays an important role in transferring information and that indeed the *meaning* of a text is communicated by the words *together with* their formatting.

### 2.2 Recording of meta-information

The second phase was carried out in the gymnasium of the school. Retrospectively this was a bad choice, since the acoustic of the big hall was quite disastrous; moreover, the groups were too far apart to allow a fruitful discussion.

The proposed task was the reproduction of formatting on a hand-written text through the use of objects. Each group was given a printed formatted text and assigned an area of the gym with a copy of the text written on a big poster on the floor. This copy contained no formatting, even the alignment was slightly different with respect to the printed version. In order to mimic the formatted text, formatting had to be *codified* by using objects available in the gym. The pupils were free to use their own conventions, but they were requested to make the formatting understandable to others, thus every group of pupils (6 persons) was asked to write down the rules they used in the *codification* phase. Finally, each team moved to the poster of the next team, received the rules written by that team, and had to follow them and reproduce the original formatted text.

Most rules turned out to be ambiguous, especially when more than one kind of formatting had to be applied to the text. The objects were used mainly to mimic the formatting in the prototypical text: for instance the presence of a color in the prototypical text was represented by an object of the same color. However, in some cases (see for example the last word in Fig. 1 and the previous word, covered with plastic cups, for comparison), pupils discovered a more abstract symbolic use as a means to cope with a shortage of objects: in the example, two spoons were used to mark the beginning and the end of the word respectively, since there were not enough spoons to cover the whole word (see the previous word, covered with red beakers, for comparison).

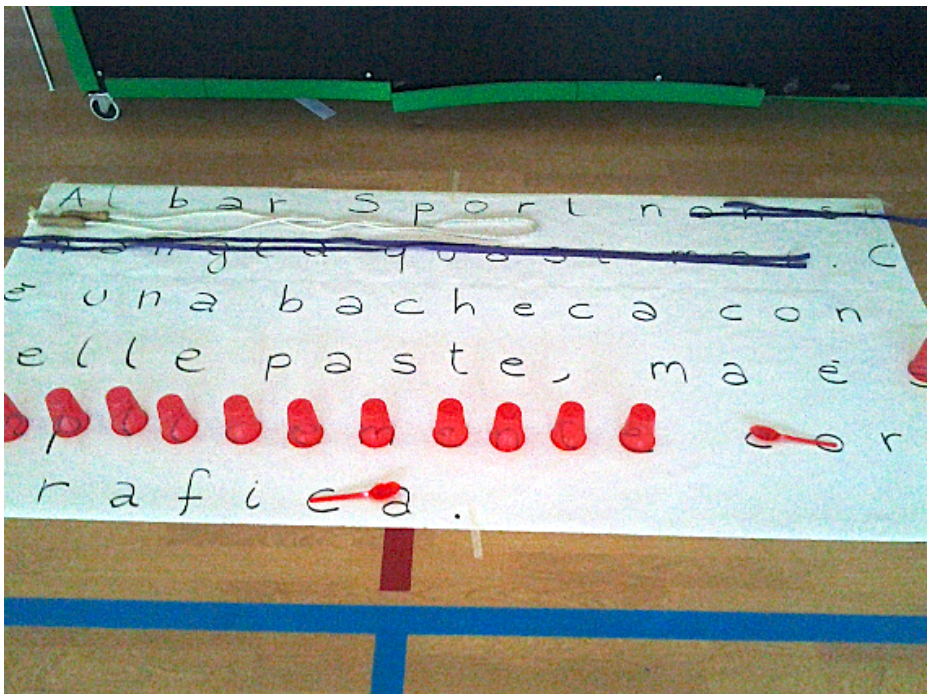



Fig. 1. Reproducing meta-information with objects



## 2.3 Discovering meta-language tricks



Guardando il **colore** della sua crema i vecchi riuscivano a trarre le previsioni del tempo.

Fig. 2. The text for the game

The shortage of objects had let a group discover a smarter “symbolic” use, thus we proposed a game. The game was carried out in a classroom in teams of four pupils. Teams were again requested to reproduce a formatted text (see Fig. 2) with objects and write down codification rules precisely enough to be followed by another team. However, each object had a *cost* and the goal of the game was to produce the formatting with the minimal cost. In fact, the more an object could be used to mimic the appearance of a piece of formatting, the higher was its cost: *e.g.*, since spaghetti pasta could be easily associated to the *underline* style, its cost was very high. We aimed at promoting the *symbolic* use of objects. The winner would be the team able to hand in an unambiguous codification with the lowest cost.

As Fig. 3 shows, the introduction of cost led the pupils to discover what is actually frequently done in *mark-up languages*: the use of *tags* at the beginning and at the end of (possibly overlapping) regions. Also, the pupils chose to use the cheapest kinds of pasta to represent the more frequent formatting elements. Pupils also devised some unexpected solutions reinforcing such an approach: they found a way to give more than one function to the cheapest kind of pasta, by changing its layout: in Fig. 3 the smallest piece of pasta is used both for bold (see the word “colore”) and for yellow highlight (see the portion “le previsioni del tempo”) and the different formatting elements are distinguished by the position of the piece, just underneath the first and last letter, or just above the first and last letter, respectively.

A second round of the game was proposed with cards instead of objects. The pupils only had a multi-set of alphabetic characters on small pieces of paper. Some letters and other keyboard characters (for example, the letters not included in the Italian alphabet: *j, k, w, x, y*) were not used in the text and could be easily exploited for weaving meta-information. However, this trick was not suggested by the facilitators but discovered by the pupils. Another solution proposed by the pupils was to use the characters with a meta-meaning by placing them with a different orientation: a  $\ominus$ , for example, for *bold*.

## 2.4 Rediscovering formatting tools

The final phase was carried out again in a computer lab. Pupils were introduced to a special software tool able to show a formatted text in three different views: formatted, encoded using a wiki syntax, encoded with a simplified HTML-like syntax. They were again requested to reproduce formatted texts by working on the other views: the tool enabled them, however, to see immediately the effect of their (syntactical) manipulation in the formatted view.

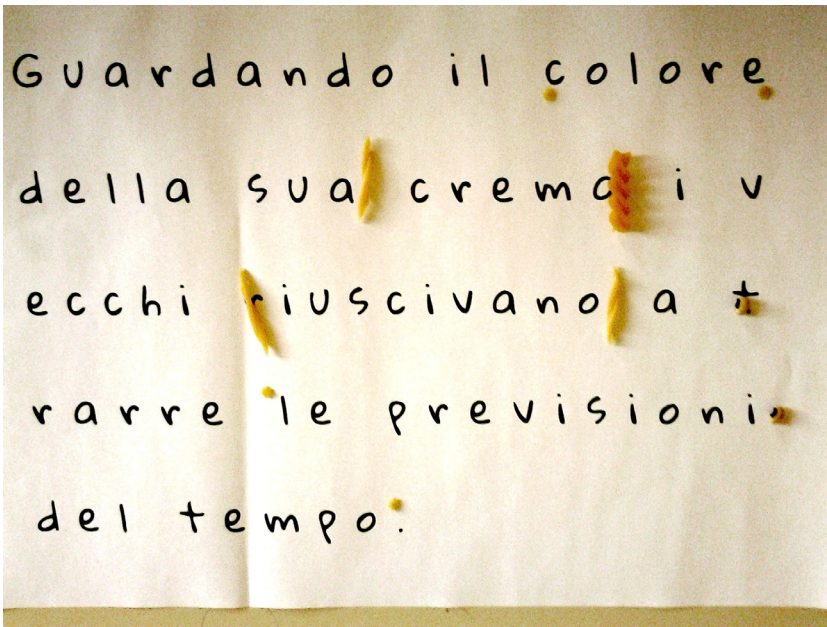


Fig. 3. Reproducing meta-information with *costly* objects

Pupils showed no difficulties in accepting these new formal syntaxes, especially the wiki one: they immediately noticed the analogy with their own rules devised in the earlier activities; they also had already pointed out that their mates' rules, though different, all shared this same approach.

## 2.5 Evaluation of the experiment

Even pupils in 9th and 10th grade with a good familiarity with technological artifacts are generally unaware of two core principles in informatics: the necessity of absolutely unambiguous descriptions and the concepts of efficiency and optimization.

We think the experiment was successful. For example, all the pupils demonstrated to have grasped the idea of a meta-language expressed in the same alphabet as the language itself. This is considered quite an abstract concept, but it was accepted as rather natural (even obvious) by the pupils.

The link with word-processor and web technologies known to pupils was recognized. At the final recap we were also able to show that the same concepts are behind the scenes in several slightly different contexts, for example when editing Wikipedia entries.

The pupils' feedback was mostly positive: they did have fun and believed to have learned something. However, some of them said that the tasks were sometimes too easy, and the part in the gym (see Section 2.2) was considered boring by several participants.

### 3 How to design and implement an algomotorial sequence of activities

Planning a sequence of activities based on the approach we propose requires a careful preliminary design phase and the preparation of ad-hoc materials.

After setting our main didactic goals – namely to teach pupils the concept of mark-up language and that information coding and processing (as all automatic processes) need non-ambiguous rules – we had to identify a sequence of intermediate logical steps to allow pupils to gradually build up their knowledge. We identified the following logical steps:

1. given a formatted text, distinguish between text and formatting elements;
2. given a printed or hand-written plain text and a set of objects, find a way how to decorate the text in order to represent formatting elements, and formalize the rules that describe the functions of objects and the way to place them;
3. after introducing a reduction of available objects, strive to devise a more concise way to represent formatting elements, as is common in mark-up languages;
4. reproduce the whole formatted text by using the same “objects” (*i.e.* symbols, that is the keyboard characters) to represent both text and formatting, thus inventing one’s personal mark-up language tag set;
5. given a fixed set of tags and a table describing their functions, apply them in an automatic wiki-like or HTML-like environment, with immediate rendering feedback.

Both steps 2 and 3 were repeated by using more and more abstract sets of objects: initially tangible objects featuring different colors, sizes, shapes (*e.g.* ropes, spoons, beakers, ...), then small objects with similar neutral features (*e.g.* kinds of pasta, see Fig. 3), and finally symbolic objects (character cards).

The careful design of the steps is critical, and we try to follow Vygotsky’s concept of *zone of proximal development* [14]. In fact, if the steps are too gradual, some pupils do not find the activity interesting and get bored. On the other hand, if the steps are not gradual enough, some pupils get lost. Indeed, the same sequence can turn out to be adequate for some and not for others, as could be observed from the pupils’ reports. For instance, some pupils chose step 3 as the least interesting, while others chose it as their favorite, one of them writing “It was demanding, but rewarding!”

Moreover it is very important to plan each step so that the activity makes sense *per se* in order to keep the pupils engaged [8]. With respect to this we found that the “didactic environment” [9] is critically important: we designed the dramatization phase with the goal of mark-up set-of-objects in mind, so we planned the activity in a large space and put the objects far from the working groups, expecting the distance would discourage the pupils from using too many objects. This did not happen at all. On the contrary, the setting of a game environment, where the goal was to economize on objects, naturally led them to discover the idea of marking only the beginning and the end of the text to be formatted. Similarly, we obtained poor results when we asked pupils to fill in a form writing a precise formulation of the rules according to which they placed the objects over the text: probably they did not feel interested enough in finding a good formal description, since an explicit context motivating such a request was missing.

Another critical point is to find the suitable trade-off between free exploring and external constraints. Pupils need authentic confrontations and should have the real possibility to make mistakes, but an open setting may have too many degrees of freedom and make pupils feel lost; on the other hand too many constraints may reduce their possibility to explore and investigate new solutions, hence inducing pupils to believe there is only one *right* answer. For instance, in the game activity (step 3) we asked pupils to write the rules they used to place pieces of pasta on the text, without handing out a form nor specifying the way how rules should be formalized; this freedom suggested to them not to use a verbal description, instead they opted for a more straightforward and effective description based on a tabular correspondence between objects and their function. It is worth mentioning another unexpected episode: the shortage of objects allowed a group to discover the possibility of coding formatting information by using only two objects, one at the beginning and one at the end of the text to be formatted, instead of using objects all along the text (often one for each character).

Finally, it is important that the teacher (better called the facilitator, or mediator [9]) observes what happens during the activities while keeping an open-minded prompt attitude. He/she should always clearly keep in mind the goal of the single activity and the overall sequence, helping pupils towards this goal, but should not force them along a set path. Unexpected events during the activities may be exploited to point out relevant issues which were not scheduled in the original design. For instance, the difficulty of some groups in providing a formal description of their rules was exploited to make them aware of the complexity of the task and of the amount of information we generally take for granted.

## 4 Conclusions

The proposed activity turned out to be a good way for conveying abstract computing concepts to pupils of secondary schools.

We are now working on refining the activity: as a mid-term goal we aim at working on producing didactic material that should be self-contained enough to be used by independent teachers in their classes. As a first step we re-proposed the activity in another school, with younger pupils (6th grade) under the guidance of a math teacher who had not participated in the conception. We are now studying reports and videos of the experiment: the first impression is that it worked also in this different context. We intend to repeat the experiment shortly with the support of a team of teachers in different schools.

We found that the *algomotricity* approach can be effective in presenting abstract symbolic manipulations in very concrete ways. By choosing activities clearly connected with the acquaintance of pupils with computers and tools we believe this approach can be very successful in fostering a thorough understanding of informatics concepts.

## Acknowledgments

We would like to thank Giorgio Fattorelli and the school Marie Curie IIS for giving us the opportunity to experiment with our ideas in their school.

## References

1. Barr, J., Cooper, S., Goldweber, M., Walker, H.M.: What everyone needs to know about computation. In: Lewandowski, G., Wolfman, S.A., Cortina, T.J., Walker, E.L. (eds.) Proceedings of the 41st ACM technical symposium on Computer science education, SIGCSE 2010, Milwaukee, USA. pp. 127–128. ACM (2010)
2. Begel, A., Garcia, D.D., Wolfman, S.A.: Kinesthetic learning in the classroom. In: Proc. of the 35th SIGCSE TSCSE. pp. 183–184. ACM, New York, USA (2004), <http://doi.acm.org/10.1145/971300.971367>
3. Ben-Ari, M.: Bricolage forever! In: Proceedings of the 11th Annual Workshop of the Psychology of Programming Interest Group, University of Leeds, UK (1999)
4. Ben-Ari, M.: Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching* 20(1), 45–73 (2001)
5. Brinda, T., Puhlmann, H., Schulte, C.: Bridging ICT and CS: educational standards for computer science in lower secondary education. In: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education. pp. 288–292. ITiCSE '09, ACM, New York, NY, USA (2009), <http://doi.acm.org/10.1145/1562877.1562965>
6. Calzarossa, M., Ciancarini, P., Mich, L., Scarabottolo, N.: Informatics education in Italian high schools. In: Kalaš, I., Mittermeir, R. (eds.) *Informatics in Schools, LNCS*, vol. 7013, pp. 31–42. Springer (2011), [http://dx.doi.org/10.1007/978-3-642-24722-4\\_4](http://dx.doi.org/10.1007/978-3-642-24722-4_4)
7. Dagienė, V.: Informatics education for new millennium learners. In: Kalaš, I., Mittermeir, R. (eds.) *Informatics in Schools, LNCS*, vol. 7013, pp. 9–20. Springer (2011), [http://dx.doi.org/10.1007/978-3-642-24722-4\\_2](http://dx.doi.org/10.1007/978-3-642-24722-4_2)
8. De Vecchi, G., Carmona-Magnaldi, N.: *Faire construire des savoirs* (2ème ed.). Hachette Education (2003)
9. Giordan, A.: From constructivism to allosteric learning model. [http://www.ldes.unige.ch/ang/publi/articles/unesco\\_AG\\_96/unesco96.htm](http://www.ldes.unige.ch/ang/publi/articles/unesco_AG_96/unesco96.htm) (1996), UNESCO Conference on Science Education 2000+
10. Hromkovic, J.: Contributing to general education by teaching informatics. In: *Informatics Education – The Bridge between Using and Understanding Computers, Proc. of ISSEP 2006, Vilnius, Lithuania. LNCS*, vol. 4226, pp. 25–37. Springer (2006), [http://dx.doi.org/10.1007/11915355\\_3](http://dx.doi.org/10.1007/11915355_3)
11. Hromkovic, J., Steffen, B.: Why teaching informatics in schools is as important as teaching mathematics and natural sciences. In: Kalas, I., Mittermeir, R.T. (eds.) *Informatics in Schools. Contributing to 21st Century Education - 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives, ISSEP 2011, Bratislava, Slovakia, October 26-29, 2011. Proceedings. LNCS*, vol. 7013, pp. 21–30. Springer (2011)
12. Taub, R., Ben-Ari, M., Armoni, M.: The effect of CS unplugged on middle-school students' views of cs. In: Brézillon, P., Russell, I., Labat, J.M. (eds.) *Proc. of the 14th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2009, Paris, France*. pp. 99–103. ACM (2009), <http://doi.acm.org/10.1145/1562877.1562912>
13. The Royal Society: Shut down or restart? The way forward for computing in UK schools. <http://royalsociety.org/education/policy/computing-in-schools/report/> (last checked 1/31/2013) (2012)
14. Vygotsky, L.: *Mind in Society: Development of Higher Psychological Processes*. Cambridge: Harvard University Press (1978)
15. Wing, J.M.: Computational thinking. *Commun. ACM* 49(3), 33–35 (Mar 2006), <http://doi.acm.org/10.1145/1118178.1118215>



## **Workshops**





# Integrating School Practice in Austrian Teacher Education

Lukas Planteu<sup>1</sup>, Bernhard Standl<sup>2,3</sup>, Wilfried Grossmann<sup>3</sup> and Erich Neuwirth<sup>3</sup>

<sup>1</sup> Bundesrealgymnasium Vienna 7, Austria  
pla@brg7.at

<sup>2</sup> Bundesrealgymnasium Vienna 10, Austria

<sup>3</sup> University of Vienna, Faculty of Computer Science, Austria  
{wilfried.grossmann, erich.neuwirth,  
bernhard.standl}@univie.ac.at

**Abstract.** We present a concept of better integration of practical teaching in student teacher education in Computer Science. As an introduction to the workshop different possible scenarios are discussed on the basis of examples. Afterwards workshop participants will have the opportunity to discuss the application of the concepts in other settings.

**Keywords:** Computer science teaching, teaching practice

## 1 Introduction

At the moment teaching experience at schools is rather limited in the curriculum of teacher education in Austria for all subjects. In order to overcome this problem we started a co-operation between the faculty of Computer Science at the University of Vienna and two Viennese schools and organized joint courses for teacher students in computer science. Besides the goal of offering student teachers more opportunities for practical teaching in university education such courses also allow testing different teaching concepts. In particular we are interested in promoting learner-centred teaching, project-based teaching and alternative ways of classroom management in teaching computer science.<sup>1</sup>

## 2 Design

The general design of the courses developed in cooperation between university and school teachers can be described by the following six phases:

1. Initial Phase: At the beginning of the course we organize a joint meeting of students and the cooperating teacher and university lecturer. This meeting offers students the opportunity to learn about knowledge of the pupils from the teacher's perspective. Furthermore the topics to be covered according to the curriculum are discussed.

---

<sup>1</sup> Work of B. Standl was supported by the European Union's territorial cooperation program between Austria and the Czech Republic of the under the EFRE grant M00171, project iCom (Constructive International Communication in the Context of ICT)

2. Introductory phase: In this phase the students visit the school and are introduced to the class.
3. Preparatory phase: Students prepare concepts for their teaching. Besides the content dimension the students are also invited to reflect about the different competence dimensions and methods for assessment and also to show how Computer Science and ICT can be applied in an interdisciplinary setting or in everyday life.
4. Discussion phase: The teaching concepts are introduced at the university. In general we do not change the students' concepts too much, because we want to give student teachers the opportunity to make their own teaching experience.
5. Teaching phase: Students apply their concepts in teaching pupils at school.
6. Reflection and evaluation phase: After the course we arrange separate discussion rounds with the pupils as well as with the students and ask them to fill in a questionnaire.

As a communication platform we use mainly Moodle, because it is at the moment the de facto standard platform at Austrian schools.

### **3 Workshop Topics**

In the workshop we will demonstrate by examples how the concept worked in detail. In particular we discuss the following three scenarios:

- (i) Using graphic software for interdisciplinary applications of Computer Science in grade 10 classes;
- (ii) Teaching basic ICT and CS skills in grade 4 and grade 5 classes according to the Austrian educational standards;
- (iii) Teaching Computer Science in a project week with a grade 11 class. This project week was planned as a preparation for school leaving examination (Matura) in Computer Science next year.

Besides the presentation of the teaching concepts in the different settings we will also discuss results of the evaluation of student teachers and pupils. In general we can say that the courses were well accepted by both groups. For students teachers the main profit was additional teaching experience and the informal contact with the teacher. With respect to teaching skills the most important experiences were learning adaptive behaviour according to different settings, time management in the classes, and understanding pupils' conceptions of different topics in Computer Science. From the learners' perspective the response was also positive because the new setting increased their attention. Especially concerning the project week everybody agreed that continuous project-oriented work is much more effective than the traditional splitting of lessons over several weeks.

The material presented will be used as input for discussion with participants about pros and cons of the approach and about opportunities to transfer the ideas into other CS teacher curricula.

# BubbleSort, SelectSort and InsertSort in Excel & Delphi – Learning the Concepts in a Constructionist Way

Jan Benacka

Department of Informatics, Faculty of Natural Sciences,  
Constantine the Philosopher University, Nitra, Slovakia  
jbenacka@ukf.sk

**Abstract.** A method is presented of acquiring the principles of three sorting algorithms through developing interactive applications in Excel.

**Keywords:** spreadsheets, sorting, constructionism

## 1 Introduction

In the workshop a method is presented of acquiring the principles of sorting algorithms SelectSort, InsertSort and BubbleSort through developing Excel applications. The author uses the method when teaching programming to undergraduates of Teaching informatics. The advantage of Excel is that the steps of the solution are on the screen, and the environment reacts immediately to the written formulas. Relative and absolute addressing is used only. The just acquired step of the solution is rewritten in Delphi or Lazarus (free at [www.lazarus.freepascal.org](http://www.lazarus.freepascal.org)). The application is in Fig. 1.

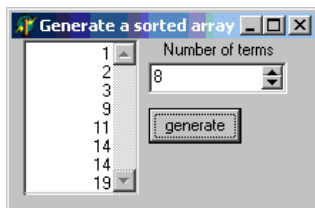


Fig. 1. The Delphi application

## 2 The applications

In BubbleSort (Fig. 2), each term of the array is compared with the next term. The terms are swapped if the first one is bigger than the second one.

In SelectSort (Fig. 3), maximum Max of the unsorted part of the array is found. Then it is swapped with the last term in the unsorted part. MaxI is the index of Max.

In InsertSort (Fig. 4), a number is inserted into a sorted array so that the array stays sorted. The method is convenient when the numbers are being generated.

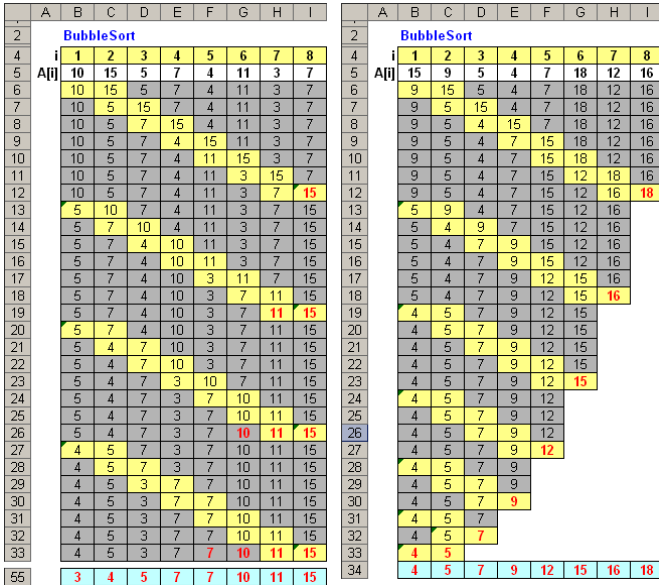


Fig. 2. BubbleSort; the simpler (left) and the improved solution (right)

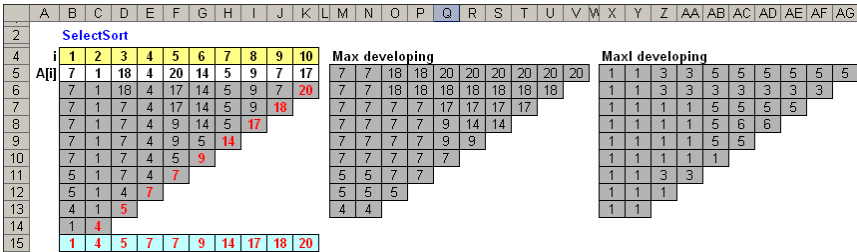


Fig. 3. SelectSort

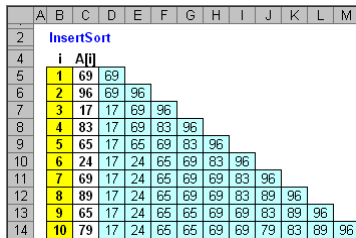


Fig. 4. InsertSort

**Acknowledgement.** The author is a member of the research team of project PRIMAS (Promoting Inquiry in Mathematics and Science Education across Europe) funded by the EU 7th Framework Programme, grant agreement 244380.

# Problem-solving strategies must be taught implicitly

Raising the awareness of in-service and pre-service computer science teachers to problem-solving strategies

Noa Ragonis

School of Education, Beit Berl College, Doar Beit Berl, 44905, Israel

Department of Education in Technology and Science, Technion – ITT

`noarag@beitberl.ac.il`

**Abstract.** Problem solving is one of the central activities performed by computer scientists as well as by computer science learners. Whereas the teaching of algorithms and programming languages is usually well structured within a curriculum, the development of learners' problem-solving skills is largely implicit and less structured. Students at all levels often face difficulties in problem analysis and solution construction. The basic assumption of the workshop is that without some formal instruction on effective strategies, even the most inventive learner may resort to unproductive trial-and-error problem-solving processes. Hence, it is important to teach problem-solving strategies and to guide teachers on how to teach their pupils this cognitive tool. Computer science educators should be aware of the difficulties and acquire appropriate pedagogical tools to help their learners gain and experience problem-solving skills.

**Keywords:** Problem solving, Problem solving strategies, Teaching problem solving strategies

## 1 Introduction

The workshop is dedicated to pedagogical tools that can scaffold learners in solving problems in the discipline of computer science. The focus will be on algorithmic solutions that can be implemented in different programming languages. Several main problem-solving strategies will be presented; the advantages and disadvantages of each will be discussed and ways of imparting them to in-service and pre-service teachers will be addressed.

## 2 Workshop topics

- Define the various required stages in the problem-solving process: problem understanding, solution design, solution examination, reflection.
- Present main problem-solving strategies: focus on variables (rules of variables [2, 7, 12]); focus on problem decomposition (stepwise refinement [1]); focus on algorithms (algorithmic patterns [3, 4, 8, 9]).
- Discuss advantages and disadvantages, limitation of usage and needed attentiveness when using each strategy.
- Discuss whether there are differences between solving problems in different programming paradigms or different topics within CS such as computational models [5].
- Discuss ways to impart problem-solving strategies to in-service and pre-service teachers [6, 10, 11].

The workshop will be based on various participant activities and group work.

## References

1. Batory, D., Sarvela, J. N., Rauschmayer, A.: Scaling stepwise refinement. *IEEE Trans. Softw. Eng.* 30(6): 355–371 (2004)
2. Ben-Ari, M., Sajaniemi J.: Roles of variables from the perspective of computer science educators. Univ. Joensuu, Depart. Comput. Sci., Technic. Report, Series A-2003-6 (2003)
3. Ginat, D.: Interleaved pattern composition and scaffolded learning. *Proc. 14th Ann. ACM SIGCSE Conf. on Innov. and Technol. in Comput. Sci. Edu. - ITiCSE '09*, pp. 109–113, Paris, France (2009)
4. Ginat, D.: Algorithmic patterns and the case of the sliding delta. *SIGCSE Bull.* 36(2), pp. 29–33 (2004)
5. Haberman, b. and Ragonis, N.: So Different Though So Similar? – Or Vice Versa? Exploration of the Logic Programming and the Object-Oriented Programming Paradigms. *Issues in Informing Science and Information Technology* 7, pp. 393–402 (2010)
6. Hazzan, O., Lapidot, T., and Ragonis, N.: *Guide to Teaching Computer Science, An Activity-Based Approach*. Springer Science+Business, London, UK (2011)
7. Laakso, M J., Malmi, L., Korhonen, A., Rajala, T., Kaila, E., Salakoski, T.: Using roles of variables to enhance novice's debugging work. *Iss. in Informing Sci. and Inf. Technol.* 5, pp. 281–295 (2008)
8. Muller, O., Ginat, D., Haberman, B.: Pattern-oriented instruction and its influence on problem decomposition and solution construction. *ACM SIGCSE Bull.* 39(3), pp. 151–155 (2007)
9. Ragonis, N.: Integrating the teaching of algorithmic patterns into computer science teacher preparation programs. In *Proceedings of the 17th ACM annual*

- conference on Innovation and technology in computer science education (ITiCSE '12), pp. 339–344 (2012)
10. Ragonis, N., and Hazzan, O.: A Reflective Practitioner's Perspective on Computer Science Teacher Preparation. Proceedings of The 4th International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP), Zürich, Switzerland, pp. 90–106 (2010)
  11. Ragonis, N. and Hazzan, O.: A tutoring model for promoting the pedagogical disciplinary skills of prospective teachers. *Mentoring & Tutoring: Partnership in Learning*, 17(1), pp. 50–65 (2009)
  12. Sajaniemi, J.: Roles of variables and learning to program. Proc. 3rd Panhellenic Conf. Didactics of Informatics, Jimoyiannis A (ed) University of Peloponnese Korinthos, Greece (2005) [http://cs.joensuu.fi/~saja/var\\_roles\\_abstracts/didinf05.pdf](http://cs.joensuu.fi/~saja/var_roles_abstracts/didinf05.pdf) (last checked 1/31/2013)





# .NET Gadgeteer Workshop

Sue Sentance<sup>1</sup> and Steve Hodges<sup>2</sup>

<sup>1</sup> Anglia Ruskin University, Chelmsford, Essex, UK

sue.sentance@anglia.ac.uk

<sup>2</sup> Microsoft Research, Cambridge, UK

steve.hodges@anglia.ac.uk

.NET Gadgeteer is a platform for creating your own electronic devices using a wide variety of hardware modules and a powerful programming environment [1]. Students with little or no Computing background can build devices made up of components that sense and react to its environment using switches, displays, motor controllers, and more. Components are plugged into a mainboard and programmed to make them work together<sup>3</sup>.

Microsoft Research has launched .NET Gadgeteer as open source software/hardware, and .NET Gadgeteer kits are now available from a variety of hardware vendors. Gadgets can be constructed by connecting the modules with cables, then programming the events triggered when using the device using Visual Studio. Either Visual Basic or Visual C# can be used.

.NET Gadgeteer has great potential in schools due to the fact that it can be used to teach students computer programming, simple electronics and also some computer-aided design. It is also very motivating for young people to be able to build their own gadgets. A digital camera can be built in about half an hour! Other devices that students can learn to build are a stop watch, traffic lights, various games, a temperature logger and a music player. The possibilities are endless! Through using .NET Gadgeteer, students learn about handling events, and are introduced to key programming concepts that are taught in schools including selection, iteration, arrays and functions.

We have held pilots in schools in the UK and the USA and students have been very motivated by the opportunity to create physical devices. Students work in groups and enjoy the opportunity to be creative as well as learning to program [2]. In this hands-on workshop, we will demonstrate .NET Gadgeteer and a variety of modules that are available. Participants will have the opportunity to build and program a small gadget or device of their own, using a range of modules and Visual Basic .NET.

## References

1. Hodges, S., Villar, N., Scott, J., Schmidt, A.: A New Era for Ubicomp Development. IEEE Pervasive Computing **11**(1) (2012)
2. Sentance, S., Schwiderski-Grosche, S.: Challenge and Creativity: Students Experiences of .NET Gadgeteer. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education (2012)

---

<sup>3</sup> <http://www.netmf.com/gadgeteer>



# Using Teachers' TryScience to support educators and improve teaching

Carol Berry<sup>1</sup> and Peter Kusterer<sup>2</sup>

<sup>1</sup> IBM Europe, Corporate Citizenship, Education Programme Manager  
carol\_berry@uk.ibm.com

<sup>2</sup> IBM Deutschland, Corporate Citizenship & Corporate Affairs manager  
kusterer@de.ibm.com

**Abstract.** The challenge is providing teachers with the resources they need to strengthen their instructions and better prepare students for the jobs of the 21st Century. Technology can help meet the challenge. Teachers' Tryscience is a non-commercial offer, developed by the New York Hall of Science, TeachEngineering, the National Board for Professional Teaching Standards and IBM Citizenship to provide teachers with such resources. The workshop provides deeper insight into this tool and discussion of how to support teaching of informatics in schools.

**Keywords:** science, teacher, collaboration, teaching material, instruction, lesson, social networking

## 1 Wanted: Great science teachers

Building the base of scientists and engineers and preparing the next generation of innovators requires great science teachers with the skills and knowledge to educate, inspire and motivate students. But the demand for science teachers continues to far outweigh the supply.

In many countries this is especially true for teachers on informatics. While informatics itself is rarely a compulsory part of today's school curricula, latest with the Web 2.0 information technology is being used not only as part of many lessons (e.g. preparing presentations) but pupils are becoming more and more versatile in using these tools, peer learning is prevalent. Formal teachers' education struggles to keep pace with this development.

The challenge is providing teachers with the resources they need to strengthen their instruction and better prepare students for the jobs of the 21st Century, many of which will increasingly be in STEM (science, technology, engineering and math) fields.

## 2 Enablement through the Web

Therefore continuously sharing of best practices is key for today's advancement of education – even more in the dynamic field of informatics.

While currently focusing on environmental science, using Teachers' TryScience, teachers, primarily at the middle school level, are able to improve their instruction

in project-based learning. Teachers' Tryscience provides free and engaging standards-based lessons, integrated with teaching strategies and resources, which are designed to spark students' interest in science, technology, engineering and math (STEM).

The site also provides social networking tools that enable educators to comment on and rate the lessons and resources; submit their own teaching materials; and form public and private groups to engage in focused discussions with colleagues in the same district or around the globe.

What distinguishes Teachers' TryScience is the integration of lessons with instructional supports. There are literally thousands of lessons on the web. Teachers' TryScience features some of the best and then helps teachers implement them effectively in the classroom by giving them the real tools to do so.

Teachers' Tryscience is a non-commercial offering, developed by the New York Hall of Science, TeachEngineering (a collaborative project between faculty, students and teachers associated with five universities and the American Society for Engineering Education), the National Board for Professional Teaching Standards and IBM Citizenship.

### **3 Objective of the workshop**

First, to introduce into Teachers' Tryscience and the underlying concept for educators to understand and more effectively use science, technology, engineering and math (STEM) learning, design-based lessons, and summative and formative assessment strategies. Second, in common work group discussions the participants will identify the potential for use on lessons on information technology and how to potentially to utilize the Teachers' TryScience website ([www.teacherstryscience.org](http://www.teacherstryscience.org)) as an instructional resource.





## Author Index

- Arslan, Okan 77
- Bellettini, Carlo 139
- Benacka, Jan 153
- Berry, Carol 161
- Cho, Shinya 13
- Dagiene, Valentina 63
- Février, Loïc 111
- Grgurina, Nataša 119
- Grossmann, Wilfried 151
- Gujberová, Monika 53
- Gülbahar, Yasemin 77
- Hodges, Steve 159
- Hofuku, Yayoi 13
- İlkhan, Mustafa 77
- Izutsu, Katsunobu 89
- Jevsikova, Tatjana 63
- Kanemune, Susumu 13
- Kilis, Selcan 77
- Kim, Seungyon 35
- Kusterer, Peter 161
- Lessner, Daniel 99
- Linck, Barbara 25
- Lonati, Violetta 139
- Malchiodi, Dario 139
- Miled, Mahdi 111
- Monga, Mattia 139
- Morpurgo, Anna 139
- Nakano, Yoshiaki 89
- Neuwirth, Erich 151
- Nishida, Tomohiro 13
- Ortiz, Pascal 111
- Park, Seongbin 35
- Planteu, Lukas 151
- Ragonis, Noa 155
- Reffay, Christophe 111
- Schulte, Carsten 63
- Sentance, Sue 63, 159
- Standl, Bernhard 151
- Thota, Neena 63
- Tomcsányi, Peter 53
- Torelli, Mauro 139
- Vaníček, Jiří 41
- Weise, Martin 127





## Previous Publications in this Series:

- 1 Andreas Schwill (Hrsg.): Hochschuldidaktik der Informatik. HDI2008 – 3. Workshop des GI-Fachbereichs Ausbildung und Beruf / Didaktik der Informatik 2008  
2009 | ISBN 978-3-940793-75-1
- 2 Stechert, Peer: Fachdidaktische Diskussion von Informatiksystemen und der Kompetenzentwicklung im Informatikunterricht  
2009 | ISBN 978-3-86956-024-3
- 3 Freischlad, Stefan: Entwicklung und Erprobung des Didaktischen Systems Internetworking im Informatikunterricht  
2010 | ISBN 978-3-86956-058-8
- 4 Engbring, D., Keil, R., Magenheimer, J., Selke, H. (Hrsg.): HDI2010 – Tagungsband der 4. Fachtagung zur „Hochschuldidaktik Informatik“  
2010 | ISBN 978-3-86956-100-4
- 5 Forbrig, P., Rick, D., Schmolitzky, A. (Hrsg.): HDI 2012 – Informatik für eine nachhaltige Zukunft  
2013 | ISBN 978-3-86956-220-9
- 6 Diethelm, I., Arndt, J., Dünnebier, M., Syrbe, J. (Eds.): Informatics in Schools – Local Proceedings of the 6th International Conference ISSEP 2013 – Selected Papers  
2013 | ISBN 978-3-86956-222-3





This series publishes conference proceedings and selected research reports in the field of computer science and informatics education from primary school to university level.

---

The International Conference on Informatics in Schools: Situation, Evolution and Perspectives – ISSEP – is a forum for researchers and practitioners in the area of Informatics education, both in primary and secondary schools. It provides an opportunity for educators to reflect upon the goals and objectives of this subject, its curricula and various teaching/learning paradigms and topics, possible connections to everyday life and various ways of establishing Informatics Education in schools. This conference also cares about teaching/learning materials, various forms of assessment, traditional and innovative educational research designs, Informatics' contribution to the preparation of children for the 21st century, motivating competitions, projects and activities supporting informatics education in school.