Programming courses reflecting students' aptitude testing and implementing learning style preferences research results

Eva Milková, Karel Petránek

Abstract An essential part of studies at faculties preparing students in the area of computer science is the development of student's programming skills. Despite a heroic academic effort and many different theoretical researches which deal with the question of how to develop these skills, a substantial minority of students fail introductory programming course. We observed this fact at our university and noticed it is similar to other universities. To change this disadvantageous situation we aimed our attention at a research concerning students' programming capabilities evaluation with regard to the input Dehnadi's test focused on programming aptitude. Within research most often students' mistakes have been deeply examined and study materials have been innovated. The paper describes our research and introduces an innovative study material successfully supporting student´s programming skills development.

Keywords Algorithmic thinking and programming skills development, programming skills evaluation.

I. Introduction

Dehnadi et al. developed a programming aptitude test which predicts whether a person can become a programmer. The test does not require any prior programming experience which is an important property that allows student pre-screening and teaching adjustments before a programming course begins. We have pursued a research project focused on programming capabilities evaluation using Dehnadi's approach at the Faculty of Informatics and Management and at the Faculty of Science, University of Hradec Králové since 2011.

During the research we examined the most common mistakes students make, and we prepared new study materials according to research results based on learning styles preferences.

In the paper we briefly introduce the Algorithms and Data Structures course, which is our introductory programming course. Secondly, we discuss Dehnadi's test and present the main results of evaluating student programming capabilities using the test. We conclude this paper with a summary of study materials innovations which successfully support programming skills and have been made with regard to most often mistakes made by students.

II. Algorithms and Data Structures

There are various research studies which deal with the question of how to start developing algorithmic thinking in students.

The education at secondary schools and colleges in the area of computer science has changed in the Czech Republic and students have only learnt a user approach. For many today's students, the algorithmic approach is almost unknown. In university departments that are training students in computer-related disciplines, the creation of algorithms is still taught mostly within courses dealing with a programming language.

There are still long discussions regarding what kind of programming language is suitable for beginners (cf. [1]). In the late 1960s Niklaus Wirth designed, especially for educational purposes, Pascal programming language, which was initially intended to teach students a structured programming, see e.g. the textbook [2]. Over the years it has been enhanced to many variants including Delphi, the object oriented version of Pascal.

Works written by Seymour Papert [3] covering children's programming languages and similar works based on his approach (e.g. [4], [5]) have given another possibility of how to develop programming skills of beginners.

There are also teachers using spreadsheets as an access to the introduction of basic algorithms (e.g. [6]).

We went through all above-mentioned possibilities at the university. However, the children's programming language seemed unsuitable for adult students. And using a program language, students concentrated more on language syntax and on options available in an environment rather than on algorithms.

Since the late 1990s most students starting at the university have had almost no concept of how to create an algorithm. We therefore decided, in 1998, to build the Algorithms and Data Structures (ALGDS) course aimed to basic algorithm construction skills. (cf. [7] and [8])

ALGDS is placed into curriculum before other courses which deal with algorithmic and programming skills. The approach that we have been using is based on an idea of a box of bricks, where only several basic shapes are available from which children are able to create impressive buildings. We do not use any programming language in the course. Students

write algorithms on paper in Czech meta-language. The Czech meta-language is nothing more than the basic commands of Pascal programming language translated into Czech.

This process of algorithm design is reflected in the course structure. Thus, when we lead students through their first steps in the creation of algorithms we explain to them that it is like building interesting objects out of just a few basic shapes. In ALGDS it means that we start our teaching with *basic algorithmic structures* (= basic elements from a box of bricks: *blocks of commands, incomplete and complete branching and loop construction*) and *typical algorithmic structures* (= a few parts made out of these elements: structures concerning *count* a value, *find* and *determine* an element with the given property and its position, *move*, *add* and *remove* an element) and then we let students into the secrets of making whole algorithms (building whole constructions).

A. Lectures and seminars

All basic algorithmic structures are explained in the lectures. The idea of each presented algorithm is illustrated by a practical situation at first, and the whole procedure is demonstrated step-by-step.

Dealing with the given topic we start to solve easier tasks and consecutively proceed to more difficult ones. We carefully discuss mutual relations among algorithms. Let us demonstrate it in the following example:

Example

Let us suppose that we are working with a finite numerical sequence ($a_1, a_2, \dots, a_n$) of n terms being already saved to the array **a** of the length n, $n \geq 1$. Let us create algorithms solving the following tasks.

- Remove the first three terms a_1, a_2, a_3 from the sequence.
- Remove the fifth till night terms, i.e. $a_5, \dots, a_9$, from the sequence.
- From the sequence remove the terms $a_p, \dots, a_q$, supposing $p < q$.

The students apply their knowledge during seminars to a variety of tasks. They work in groups of two or three and each group is responsible for solving one of the given tasks. The students are given some time to prepare their solutions on a piece of paper. Then the task is illustrated and presented on the blackboard by two or three students, each from a different group. The class discusses and compares the given solutions. This approach is beneficial both for the students who are pushed to try and find more solutions to a task, and for the teacher who has an opportunity to open a discussion if there is a problem with the solutions.

B. Study materials

The whole area explained within ALGDS is introduced in the *textbook* [9], where more than 150 problem assignments, questions and exercises are presented. The accuracy of a solution can be verified with the help of the *Algorithms program* (see below in the text) which is included together with solutions of all the textbook provided tasks, on an accompanying CD.

Students can view the electronic version of the textbook as well as they can download the *Algorithms* program in the virtual study environment used at the university. They can also access other study materials corresponding to a lecture such as additional lecture notes, problem statements of tasks solved in seminars, presentations and animations used at the lecture. There is also a detailed plan of lectures available as well as credit conditions and sample of credit test, which contains not only standard type of task but also the following ones:

I)

Complete the algorithm solving the following task. In the sequence of n integers saved in the array **a** determine the first minimum value and then sum all integers behind the found minimum value.

```
begin
   minimum := a[1];
   sum := ..........;
   for i from 2 to n do
   begin
      sum := sum + ..........;
      if a[i] ... min then
      begin
         minimum := ..........;
         sum := ..........;
      end;
   end;
end.
```

II)

There are n integers saved in the array **a** (see the table). Determine the values in the array **a** after finishing the following algorithm. Write them to the table below.

```
begin
   n:=6;
   x:=a[1];
   i := 2;
   while i ≤ n - 1 do
      begin
         if a[i] > x then
            begin
               a[1]:= a[i];
               a[i]:= x;
            end;
         i := i + 1;
      end;
end.
```

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
11	8	19	7	16	17

C. Multimedia applications

Education scientists have been discussing "changing the education paradigm" pointing out that the power of technology has caused fundamental changes in all aspects of our lives, including the educational process [10]. Multimedia applications provide teachers with an excellent opportunity to demonstrate and visualize the subject matter more clearly and comprehensibly, as well as to enable them to prepare a study material that optimizes students' study habits. Multimedia

animations and presentations are usually used to describe a topic and illustrate it using a visualization of objects and processes. They mostly enable to test the coursework using several prepared exercises. Multimedia programs offer not only the options as presentations, but their biggest advantage is that they can provide an infinite number of more needed examples. (cf. [11] - [13])

Along with large multimedia software products developed by a team of professionals there are also various smaller presentations and multimedia applications appropriate to course subject matter created by students themselves based on a script given by the teacher.

One of such programs is the *Algorithms* program. It is developed in the Borland Delphi environment and available on http://lide.uhk.cz/prf/ucitel/milkoev1/en_index.htm, the page ALGDS / Lectures / Students presentations). Several animations prepared for the subject ALGDS are available there as well.

The program is an excellent tool for all students attending ALGDS. The program is user-friendly and provides entire graphical support for users. Its options are designed to be intuitive and at the same time to remind professional editors and debuggers of well-known programming languages [14].

Using the program, students can place their solution(s) to the given task, written in Czech meta-language, into the program and the program shows them step-by-step how their algorithm works and if it is correct or not. In each step of the algorithm process the program shows current values of variables.

III. Dehnadi's test and research results

When introducing programming to freshmen students, there seems to be a clear division between students who find programming easy and those who have a hard time grasping even the basic concepts. Dehnadi and Bornat [15] observed that the level of mental model consistency that learners apply when faced with a novel, seemingly nonsensical problem could be a good predictor of programming capabilities. Saeed Dehnadi designed a test focused on discovering future programmers before they enter their first programming class [16].

A. Dehnadi's test

Dehnadi's test [17] is based on assessing mental model consistency in the assignment operation. The test consists of twelve similar questions. Each question gives a sample C-like program, declaring two or three variables and executing up to three variable-to-variable assignment instructions.

The test is evaluated according to an answer sheet and a mark sheet which together assign mental models to each answer and combine them into a total score. There are 11 mental models described in [17], slightly modified in [18], see Fig. 1. Except for the classical assignment operation model known from C or Java, Dehnadi identified 10 other models that were used by the applicants. The common ones include the left-to-right assignment (as opposed to the standard right-to-left), comparison or assignment without transfer to the next line. If a person is able to choose any of these 11 models and use it consistently through the whole test, he or she is considered a good candidate to become a programmer. The consistency threshold was set at 8 consistent answers out of 12. Dehnadi and Bornat [15] report that 44 % of the applicants showed consistent models, 39 % used several models inconsistently and 8 % have refused to fill the answers. The remaining 9 % of students are not discussed in the paper. [19]

M1. Value moves from right to left (a ← b and b ← 0 - eighth line in figure 1).

M2. Value copied from right to left (a ← b - fourth line of figure 1, and the 'correct' answer in Java).

M3. Value moves from left to right (b ← a and a ← 0 - third line of figure 1).

M4. Value copied from left to right (b ← a - first line of figure 1, a reversed version of the 'correct' answer).

M5. Right-hand value added to left (a ← a+b - second line of figure 1).

M6. Right-hand value extracted and added to left (a ← a+b and b ← 0 - tenth line of figure 1).

M7. Left-hand value added to right (b ← a+b - ninth line of figure 1).

M8. Left-hand value extracted and added to right (b ← a+b and a ← 0 - fifth line of figure 1).

M9. Nothing happens (sixth line of figure 1).

M10. A test of equality (first and fourth lines of figure 1).

M11. Variables swap values (seventh line in figure 1).

Fig. 1 Anticipated mental models of assignment a=b [18]

B. Dehnadi's test applications

Dehnadi's test promises to bring the holy grail of programming education and aptitude testing – with a simple test we should be able to decide whether a student will be a fast or slow learner. This brings several possibilities to enhancing education and certification:

- Students can be split into classes based on their expected proficiency
- The teachers can direct their focus better by knowing the audience
- Companies could easily filter applicants for a programming position
- The students can adjust their focus according to their test results

A test of the qualities described in Dehnadi's research would thus be a great improvement of programming education. Given the small sample size in the original paper and due to some of its criticisms [20] we decided to verify Dehnadi's claims on our freshmen at University of Hradec Králové on a bigger number of students than the original paper.

C. Research data

We first decided to use Dehnadi's test in academic year 2011/2012 at the Faculty of Informatics and Management to test our freshmen at the beginning of the first programming ALGDS course. We hoped that Dehnadi's test could serve as an orientation resource both for students and teachers. Students who belong to the inconsistent group should devote a more thoughtful attention to the subject; they should study regularly, discuss their solutions with the teacher or experienced students and practise their programming skills (preferably every day). Teachers would be able to use the Dehnadi's test results to divide students into appropriate groups when explaining and practising algorithm design. During academic years 2011/2012, 2012/2013, 2013/2014 and 2014/2015 data of 473 students who both filled Dehnadi's test and participated in the introductory programming course were analysed.

Minding the fact that there are students starting the university who had already a prior programming experience before attending the first programing course we divide the students to two groups: *students with no prior programming experience* and *students with prior programming experience*.

D. Results

To avoid human errors during test evaluating, we prepared automated software [21] that evaluates each student's test based on the grading sheets released by Dehnadi. Having the possibility to use the automated software we are able to evaluate hundreds of students' test results.

Dehnadi's test targets students with no previous exposure to programming. The original test is supposed to be applied primarily to non-exposed group of students. At the same time, it is supposed to give biased results of students who were previously exposed to programming. Our aim is to verify these results – we compared the results of successful and unsuccessful students and their performance in Dehnadi's test for both groups. We used Wilcoxon rank sum test (at significance level 5 %) to check for significant differences in expected score of successful and unsuccessful students.

The main result from our research is that Dehnadi's test cannot predict whether a person will be a proficient programmer or not (judged by getting a credit for our introductory programming course) when the person has not been exposed to programming before. This contradicts Dehnadi's findings and thus we cannot confirm the conclusion of his papers. See Fig. 2 for a visual comparison of student scores for students who have no prior programming experience. Obviously, Wilcoxon rank sum test does not reject the null hypothesis that the two groups come from the same population (p-value 0.274, significance level 0.05).

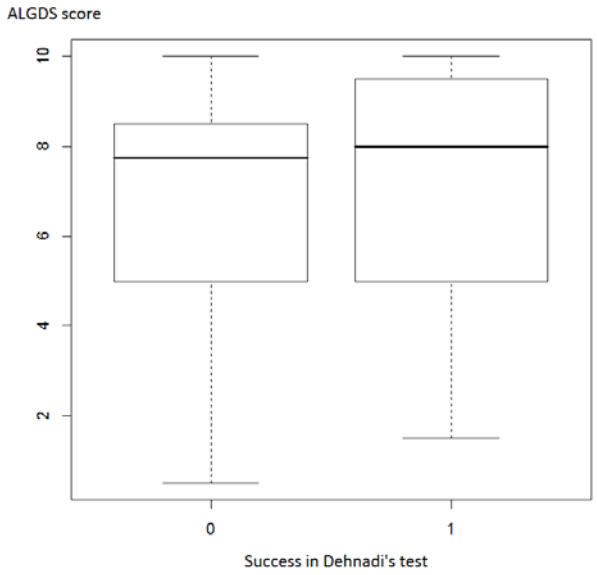

Fig. 2 Box plot showing the distribution of scores from our introductory programming course ALGDS based on their success in Dehnadi's test (1 means successful). Students needed 6 points to get the credit. Only students with no prior programming experience are included in this plot.

However, when Dehnadi's test is applied to students who have already been exposed to programming before, we show that the test actually does have a discriminative power. If a student with previous programming experience does not reach a full score (12 points) in Dehnadi's test, he or she will likely fail the course.

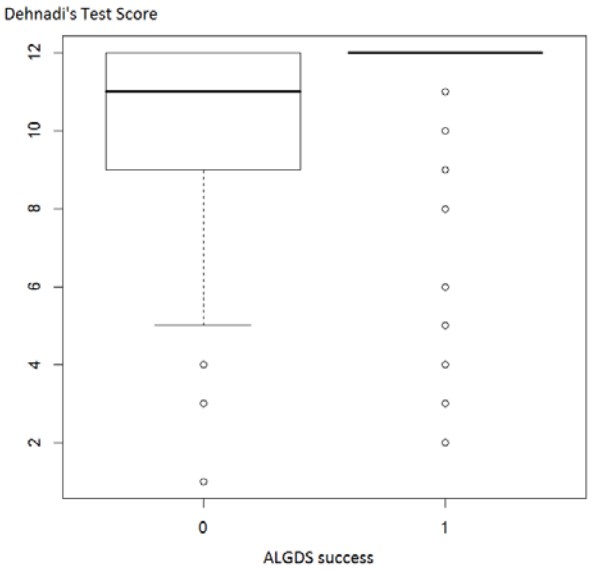

Fig. 3 Box plot showing the distribution of Dehnadi's test scores for students with previous programming experience that were unsuccessful (left) and successful (right) in our introductory programming course ALGDS.

This result also makes an intuitive sense – Dehnadi's test is based on simple assignment tasks and if a student is unable to comprehend these after a previous programming course, it is unlikely he or she will perform better in our course. The Wilcoxon rank sum test also reports a statistically significant difference with p-value 0.003 given significance level 0.05. Fig. 3 shows two distributions of Dehnadi's test scores, both for students who succeeded in our introductory course and those who failed the course. We can observe easily that there is only a handful of students with previous programming experience who did not reach a full score and yet passed the course. (cf. [22])

E. Applications and discussion

Contrary to the claims in [15], results of our research have shown that the Dehnadi's test *cannot* predict success in the ALGDS course for *students with no prior programming experience* (cf. [20]).

Nevertheless, using the test *we are able to predict success in* ALGDS subject of *students with prior programming experience*. Supposing that about a half students attending ALGDS subject have prior programming experience, the Dehnadi's test can serve as useful information for students not achieving a full score to devote a considerable care to the subject.

Since 2014 we have had a fully automated on-line version of the test where students see their score right after submitting the test. While we do not count Dehnadi's test results toward the final score, students are required to fill it during the first week of the semester and include information about their previous programming experience. This serves two purposes – the students who fail Dehnadi's test experience the psychological effect of a failed test and the teachers can see the scores and experience of all their students and adjust their lectures. While a failed test does not affect the final grade, it still motivates many students to devote extra time to the course. Especially students who had a previous programming course and fail Dehnadi's test feel compelled to "make up" for this failed test – exactly the effect we want to achieve as this group is most susceptible to fail the course. Based on many years' experience we are sure that the existence of students who successfully passed Dehnadi's test but did not succeed in ALGDS can be explained as follows: Students with prior programming experience usually think that their previous programming skills are enough to pass the ALGDS subject and do not devote the necessary attention to the subject matter. We therefore use Dehnadi's test as a wake-up call for these students.

On the other hand, students with no previous programming experience are able to pass ALGDS subject thanks to a systematic development of algorithmic thinking. Dehnadi's test does not play a big role for this group because of its non-existent discriminative power. Also the students with no previous programming exposure tend to take the failed test more lightly expecting that the course is supposed to teach them how to answer questions like that.

IV. LEARNING STYLES

The individual learning styles model is one of the concepts that is postulated by scientists to determine the different students´ approaches to receiving, processing and further presenting obtained information. In this regard extensive global research has been undertaken, the results of which are applied in different models of learning styles, see e.g. an overview in [23]. This can then be integrated into all aspects of the educational process.

People learn and process information in different ways. Learning style assessments provide learners as well as teachers an opportunity to learn how to respond under different circumstances and how to approach information in a way that best addresses students' particular needs.

Out of plenty of learning style models we have chosen the Felder-Silverman learning style model and on this model based questionnaire called the Index of Learning Styles.

Felder [24] defines individual learning styles as follows "The ways in which an individual characteristically acquires, retains, and retrieves information are collectively termed the individual's learning style".

The Felder-Silverman learning style model was created by Richard M. Felder on Dr. Silverman's expertise in educational psychology and his experience in engineering education.

Felder and Silverman [25] propose that a student learning style may be defined by the answers to five questions:

1. What type of information does the student preferentially perceive: sensory (external) – sights, sounds, physical sensations, or intuitively (internal) – possibilities, insights, hunches?

2. Through which sensory channel is external information most effectively perceived: visual – pictures, diagrams, graphs, demonstrations, or auditory – through words or sounds?

3. With which organization of information is the student most comfortable: inductive – where facts and observations are given, and underlying principles are inferred, or deductive – where principles are given, and consequences and applications are deduced?

4. How does the student prefer to process information: actively – through engagement in physical activity or discussion, or reflectively – through introspection?

5. How does the student progress toward understanding: sequentially – in continual steps, or globally – in large jumps, holistically? (cf. [26] and [27])

Felder´s model includes four dichotomous learning style dimensions which indicate students´ preferences for certain poles of the dimensions:

Sensing or Intuitive - this spectrum determines how we perceive or take in information,

Visual or Verbal, this spectrum determines how we prefer the information to be presented,

Active or Reflective - this spectrum determines how we prefer to process the information

Global or Sequential - this spectrum determines how we prefer to organize and progress understanding information.

A. The Index of Learning Styles

The Index of Learning Styles (ILS) was developed by Richard M. Felder and Barbara A. Soloman and may be used cost-free for non-commercial purposes by individuals who would like to determine their own learning style profile and by educators who would like to use it for teaching, advising, or research. For more information see [28] - [29].

ILS is a self-scoring web-based instrument that assesses preferences on the *active* / *reflective*, *sensing* / *intuitive*, *visual* / *verbal*, and *sequential* / *global* dimensions. The forty-four multiple choice questions in the questionnaire reflect the psychological and behavioural characteristics of four dichotomous dimensions of learning styles mentioned above. After submitting their answers, students are provided with Learning Style Results (see Fig. 4), where if their score on the scale is 1-3, they are considered fairly well balanced on the two dimensions of that scale. If their score on the scale is 5-7, they have a moderate preference for one dimension of the scale and will learn more easily in a teaching environment which favours that dimension, and if their score on the scale is 9-11, they have a very strong preference for one dimension of the scale and are classified as purely single-style learners, which may cause struggling and suffering when learning in an environment which does not support their preference. [27]

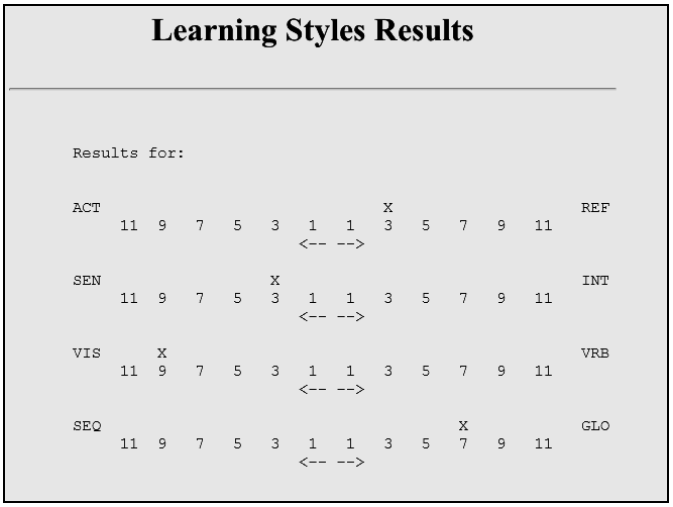

Fig. 4 ILS graph showing student preferences

B. Research Data and Results

The main results concerning learning styles preferences of a student attending ALGDS during three previous academic years 2012/13, 2013/14 and 2014/15 can be summarized as follows.

From our general observation of students most of them are visual learners. The results show that in fact 87% are *visual learners*, whereas 56% are *strong visual learners*.

About 72% students belong to *sensing dimension*, whereas 34% of them to the moderate till strong sensing dimension.

Concerning the other two dimensions, active/reflective and sequential/ global, most students belong to the mild level of these dimensions more of them to the „left side", which means to the *mild active* and *mild sequential dimension*. Surprisingly, almost no students belong to the strong „right" side of any dimension.

C. Study material for ALGDS reflecting learning style preferences

Our aim has been to prepare lectures, lessons and suitable study materials in a way that support all dimension described above. This fact guarantees optimal study conditions for all students. On one hand each student can find a study material suitable for his/her learning style. On the other hand each student is slightly pushed to enhance also his/her opposite learning style dimension.

Let us explain it in detail.

Thanks to the way that students create an algorithm to a task given at a seminar (see section II. A) both, active and reflective dimension is supported.

Learners belonging to the sensing dimension remember and understand information best if they can see how it is connected with the real world. Therefore, we explain the idea of each algorithm, that is presented in the lecture in a practical situation and we usually demonstrate it with the help of students.

At lectures and seminars we devote enough time and space to discussions concerning mutual relations among algorithms that is a valuable support to both intuitive learners who prefer to discover possibilities and relationships and to learners belonging to the sensing dimension who need enhance innovative thinking. (Remark: According [25] *„Everybody is sensing sometimes and intuitive sometimes.* To be effective as a learner and problem solver, you need to be able to function both ways. If you overemphasize intuition, you may miss important details or make careless mistakes in calculations or hands-on work; if you overemphasize sensing, you may rely too much on memorization and familiar methods and not concentrate enough on understanding and innovative thinking.").

Using the *Algorithms* program both sequential and global students can find their own way to support their self-study, how to revise a subject matter and understand it more properly and accurately.

Algorithms are described in words, which is the method that verbal students prefer. However, there are several ways to support visual students as well. For example, each algorithm introduced in the first few lessons is also described using a flowchart. Small animations visualizing basic algorithmic structures and the *Algorithms* program (see section II. B) visualizing changing values in variables during the whole process, belong to another helpful method.

A new significant support that has been provided recently is introduced in the following section.

V. Additional new study material for ALGDS

In section II we mention that students can download the textbook [9] and the *Algorithms* program from the virtual study environment.

Using the results gained from the analyses of student

learning styles preference we made the following changes.

By academic year 2013/14 we had first explained the one-dimensional data structure. After that we proceeded to two-dimensional array problems. In 2015 we decided to change this process. We explained each typical algorithmic structure for both one-dimensional and two-dimensional arrays. We found out that the students are able to comprehend better the array index manipulation this way and thus better understand the whole subject matter. Compared to the previous years the fear of working with two-dimensional array indices almost disappeared.

To support more and more visual learners we have prepared another significant innovation. The texts describing the typical array manipulation techniques were complemented by colours emphasizing structures used in the given algorithm. We demonstrate this method on an extract of the chapter Shift of values within array published in the textbook [9], namely on the algorithmic structure *Inserting a new array element*. The demonstration follows.

Inserting a new array element

Let us introduce a typical algorithmic structure used in algorithms for adding a new value x after the k-th term of a numerical sequence $(a_1, a_2, \dots, a_n)$, $1 \le k \le n$, stored in an array a.

```
read(x);
read(k);
i := n;
while i ≥ k + 1 do
begin
   a[i + 1] := a[i];
   i := i - 1;
end;
a[k + 1] := x;
n := n + 1;
```

Example

Let us create an algorithm which adds –1 after the last minimum element of a numerical sequence $(a_1, a_2, \dots, a_n)$, $1 \le k \le n$, and writes out the resulting sequence.

```
begin
   read(n);
   for i := 1 to n do
      read(a[i]);
   min := a[1];
   indexMin := 1;
   for i := 2 to n do
      if a[i] ≤ min then
      begin
         min := a[i];
         indexMin := i;
      end;
   i := n;
   while i ≥ indexMin + 1 do
   begin
      a[i + 1] := a[i];
      i := i - 1;
   end;
   a[indexMin + 1] := -1;
   n := n + 1;
   for i := 1 to n do
      write(a[i]);
end.
```

VI. Conclusion

We present a detailed list of improvements to our introductory programming course at University of Hradec Králové that result from our research in programming aptitude testing and learning style analysis.

We show that the programming aptitude test cannot be used blindly with all students. Good discriminative results can be achieved with students with a prior programming experience.

The learning styles preferences research shows that majority of students tend to mild preference to the four dimensions of learning styles. However, there are moderate and strong preferences especially to sensing and visual dimensions. The optimal condition is that teachers can help students acquire the ability to use their less preferred style modalities when appropriate and make those learners with strong preference to certain learning styles move toward a position of greater balance [30].

We have found out that devoting enough time explaining mutual relationships among solved problems and carefully, together with students, recognizing the differences among almost the same algorithms is very helpful for beginners and sometimes reveals hidden connections even for experienced students.

The learning styles preferences research leads to a technically simple, but an extremely helpful extension of the textbook that uses colour coding to match sentences from an exercise with corresponding lines of its algorithmic solution.

Results achieved from the credit tests show that our improved pedagogical approach used in the first programming course, Algorithms and Data Structures, is successful.

Acknowledgment

This research has been supported by Specific Research Project of the University of Hradec Králové, Faculty of Science No. 2113.

References

[1] Guniš, J., Šnajder, L. (2012) The model of algorithmic thinking – dimensions and levels. In: *Information and Communication Technology in Education* (ICTE 2012), Ostrava: University of Ostrava, Rožnov pod Radhoštěm, Czech Republic, September 11–13, 2012, 69–78.

[2] Wirth, N. (1975) *Algorithms + Data Structures = Programs*, Prentice-Hall, New Jersey.

[3] Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas*, Basic Books, Inc.

[4] Blaho, A., Kalaš, I. (1995) Playing, developing and computing with images in Comenius Logo for Windows. *EuroLogo Proceedings*, pp.15–19.

[5] Hubalovský, Š., Musílek, M. (2013) Modeling, Simulation and Visualization of Real Processes in LOGO Programming Language as a Method of Development of Algorithm Thinking and Programming Skills. *International Journal of Mathematics and Computers in Simulation*. Vol. 7, No. 2, p. 144–152.

[6] Lovaszova, G., Hvorecky, J. (2005) Using spreadsheet calculations to demonstrate concepts of programming. *International Journal of Continuing Engineering Education and Life Long Learning*, 15(3), pp. 162–184.

[7] Milková, E. (2011) Multimedia Tools for the Development of Algorithmic Thinking. *Recent Patents on Computer Science*. 4(2), 98–107.

[8] Milková, E. (2015) Multimedia Application for Educational Purposes: Development of Algorithmic Thinking. *Applied Computing and Informatics* Vol. 11, Issue 1, 76–88. doi: 10.1016/j.aci.2014.05.001
[9] Milková, E. et al. (2010) *Algoritmy: typové konstrukce a příklady*, Hradec Králové: Gaudeamus.
[10] Robinson, K. (2011) *Out of Our Minds: Learning to be Creative*, Capstone Publishing Ltd, Oxford.
[11] Sedivý, J. Multimedia support of parametric modeling. *Engineering education (EDUCATION 2012) : proceedings of the 9th WSEAS international conference*. Athens : World scientific and engineering academy and society, 2012. 5p.
[12] Michalík, P., Toman, J. (2013) Possibilities of Implementing Practical Teaching in Distance Education. *International Journal of Modern Education Forum*, Vol. 2, No. 4, pp. 77-83.
[13] Němec, R., Hubalovský, Š. (2014) Software Design of System SMPSL, *WSEAS transactions on computers*, Vol. 13, pp. 329-337.
[14] Voborník, P. (2011) Teaching algorithms using multimedia tools, *Proceedings of the 8th International Conference on Efficiency and Responsibility in Education*, Czech University of Life Sciences Prague, pp. 312–321.
[15] Dehnadi, S., Bornat, R. (2006) The camel has two humps (working title), Middlesex University, UK.
[16] Dehnadi, S., Bornat, R. & Adams, R. (2009) Meta-analysis of the effect of consistency on success in early learning of programming, *Psychology Programming Interested Group (PPIG) Annual workshop*.
[17] Dehnadi, S. (2006) Testing programming aptitude, *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, pp. 22–37.
[18] Bornat, R. et al. (2008) Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education*, Vol. 78. pp. 53–61.
[19] Milková, E., Petránek, K. & Janečka, P. (2012) Programming capabilities evaluation, *Proceedings of the 9th International Conference on Efficiency and Responsibility in Education*, Czech University of Life Sciences Prague, pp. 310–318.
[20] Bennedsen, J. & Caspersen, M. E. (2007) Failure rates in introductory programming, *ACM SIGCSE Bulletin*, Vol. 39, No. 2, pp. 32–36.
[21] Petránek, K., Janečka, P. (2013) Dehnadi Evaluation, *Software for automatic grading of Dehnadi's test*, available online at https://github.com/karelp/dehnadi_evaluation
[22] Petránek, K., Janečka, P. (2013) Testing programming aptitude: An in-depth analysis, *Efficiency and Responsibility in Education*, Czech University of Life Sciences Prague, pp. 497–502.
[23] Saeed, N., Yang, Y., & Sinnappan, S. (2009). Emerging Web Technologies in Higher Education: A Case of Incorporating Blogs, Podcasts and Social Bookmarks in a Web Programming Course based on Students' Learning Styles and Technology Preferences. Educational Technology & Society, 12 (4), 98–109.
[24] Felder, R. (1995) Learning and Teaching Styles in Foreign and Second Language Education, *Foreign Language Annals*, Vol. 28, No. 1, 1995, pp. 21–31.
[25] Felder, R. M,. Silverman L. K. (1988) Learning and Teaching Styles in Engineering Education, *Engr. Education*, Vol. 78, No. 7, pp. 674–681.
[26] El-Hmoudova, D., Milková, E. (2015) Variations and Frequencies in Learning Styles in a Group of Czech English as Foreign Language Learners. *Procedia - Social and Behavioral Sciences*, Vol. 182 (2015), pp. 60-66. doi: 10.1016/j.sbspro.2015.04.738
[27] El-Hmoudová, D. (2015) Assessment of individual learning style preferences with respect to the key language competences, *Procedia - Social and Behavioral Sciences*, Vol. 171 (2015), pp. 40–48. doi:10.1016/j.sbspro.2015.01.086
[28] http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Learning_Styles.html
[29] http://www4.ncsu.edu/unity/lockers/users/f/felder/public/ILSpage.html
[30] Felder, R. M., Spurlin, J. E. (2005). Application, reliability, and validity of the Index of Learning Styles, *Int. J. Engr. Education*, 21(1), pp. 103-112.