R&E-SOURCE http://journal.ph-noe.ac.at
Open Online Journal for Research and Education
Special Issue, December 2017, ISSN: 2313-1640

Increasing IT skills in algorithmization for students of applied informatics

Jana Jurinová[1]

Abstract

In this paper, we present possibilities for creating and using purposefully designed interactive applications in order to understand the problematic parts taught within the course "Algorithms and Data Structures I" and the acquisition of the required IT skills of students in informatics. The reason is steadily decreasing percentage of students, despite the measures that have been taken and proved to be effective. These were specified in details by the author in another paper (Jurinová, 2016). However, their effectiveness is still not satisfactory, and therefore we continue to support teaching methods and forms of education by developing other customized interactive applications.

Keywords:	*Schlüsselwörter:*
Algorithm	Algorithmus
Algorithmization	Algorithmisierung
IT Skills	Informatikkenntnisse
Interactive Application	Interaktive Anwendung
Nassi-Schneiderman Diagram	Nassi-Shneiderman Diagramm

1 Introduction

From various studies (Herout, 2016; Avancena, Nishihara and Kondo, 2015; Urquiza-Fuentes and Velázquez-Iturbide, 2009), as well as from own experience and the carried out research (Jurinová, 2015; Jurinová, 2013), we have found out that the more diversified and entertaining forms of learning by using interactive multimedia applications and e-materials developed with the purpose to influence the specific knowledge and skills, have positive impact on learning. Students can better remember the subject matter and are more motivated to study the topic and improve knowledge. Basics of algorithmization are the basis and the fundamental subject for students of IT field. Abstract nature of learned concepts, required analytical thinking and lack of interest from students are the main causes of this failure. In addition, other institutions in the world face similar problems on a global scale. Similarly like us, they are trying to create various applications (Avancena, Nishihara and Kondo, 2015; Diehl, 2015), which would facilitate the process of understanding and acquiring the required knowledge and skills. The research presented in paper (Jurinová, 2016) was oriented for this type of students and carried out at the author's workplace. Assuming that the division of the course "Algorithms and Programming" with a very broad concept and with the time allotment 2 hours of lectures and 2 hours of exercise per week into two mutually cooperating subjects "Algorithms and data structures I" and "Programming I" with the time allotment of 2 hours of lectures and 2 hours of exercise per week for each of them will help to reverse the decline in student success has proven to be inadequate. This step, however, led to a new reality monitored in the academic year 2015/2016. Based on statistics from the Academic Information System (AIS), the percentage of students in "Programming I" improved significantly, while their success in "Algorithms and Data Structures I" deteriorated. These facts are illustrated in Fig. 1.

[1] Department of Applied Informatics and Mathematics, Faculty of Natural Sciences, University of Ss. Cyril and Methodius in Trnava, Slovakia. E-Mail: jana.jurinova@ucm.sk

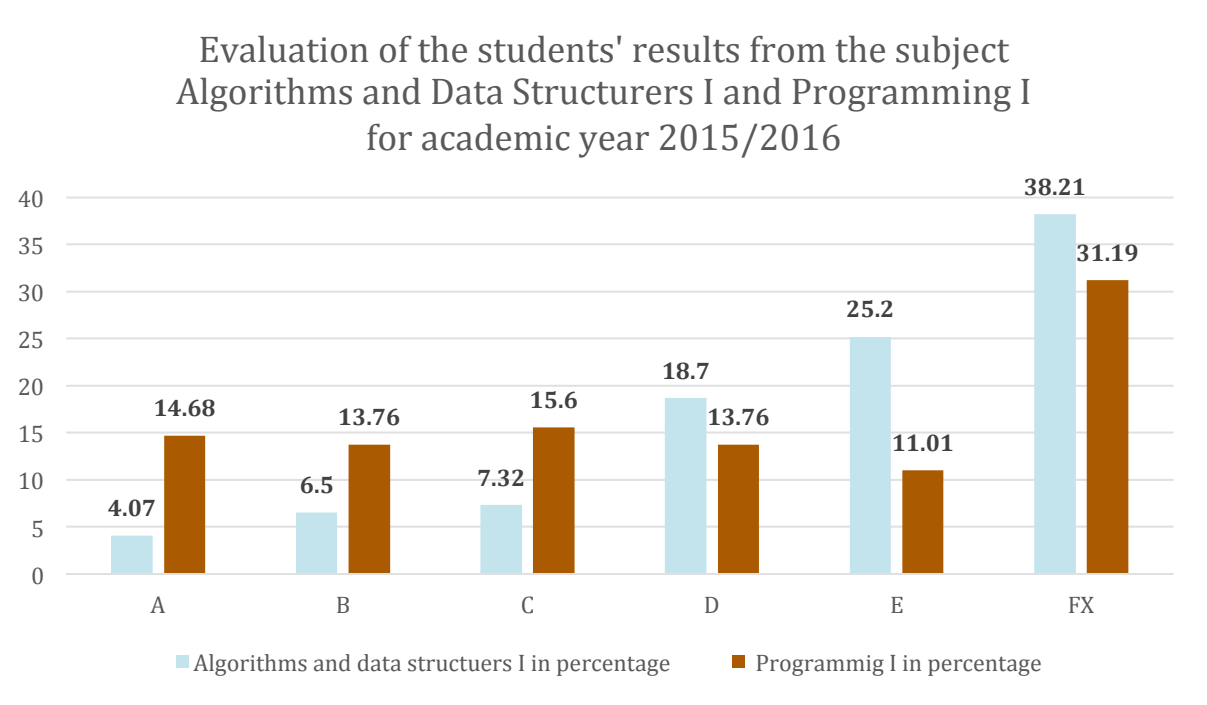

Fig. 1: Evaluation of the students' results from the course Algorithms and Data Structures I and Programming I for academic year 2015/2016

Based on this fact, it can be said that the acquisition of programming skills is not as complicated for students as the analysis and writing of algorithms. The reason is also the fact that programming can be understood as an engineering activity. On the other hand, the algorithms design is known as a creative and intuitive process. This is evidenced by the fact that algorithmization is the science studying algorithms, while algorithm theory is mathematical science studying mathematical models of algorithms. The concept of creating an optimal algorithm for solving the problem therefore requires students to have extensive knowledge from a number of areas (mathematics, programming, algorithmization, the area for which the algorithm is designed), as well as their own experience. However, programming of the already proposed algorithm expects students only to have acquired certain programming language and the syntax of individual commands. On the other hand, programming output control is based on functional testing (most commonly in the form of black-box, grey-box or white-box testing) of the created application, i.e. the student has almost immediate feedback on whether his/her program is working properly correctly or not. The student does not have such feedback after completing the algorithmization of the task. Because the only feedback control is level of the student's knowledge or the control directly by the teacher/tutor or by the person with acquired the given knowledge and skills. The mentioned fact represents the problem we were trying to solve and we present in this paper.

2 Materials and methods

If the student's knowledge is insufficient to correctly design the algorithm, how can one expect a proper check of the result of his/her activity? On the other hand, it cannot be expected that the teacher is able to check each student's output at any time of the student's study (we mean, in particular, his/her self-study).

There are several ways to write algorithms: verbal (natural language), pseudocode, graphic, mathematical, etc., each of which has its pros and cons. The subject of our interest in this paper is the graphical representation of algorithms, among which we include, beside others, a flowchart diagram and a Nassi-Schneiderman diagram (NS diagram). NS diagrams are an alternative notation for process flowchart. It is a diagrammatic approach to algorithm design but is not as bulky to draw as flowcharts. Students are guided to use both forms of this graphical representation of algorithms. Argumentatively and critically, they are also acquainted with a rigorous analysis of their suitability for plotting particular factors. It is not always useful to use a flowchart algorithm representation or a NS diagram. It depends on the use of control structures and data structures. In general, however, it can be stated that the use of NS diagrams leads to fewer errors in the algorithm design that students do when using a flowchart. It results from the nature of the individual diagrams.

The flowchart is represented as an oriented graph, so that the plotting options against the NS diagram represented by the block composition are multiple, i.e., NS diagrams are more structured than flowcharts. Although the use of flowchart is more common, due to the above-mentioned reasons the use of NS diagrams is more appropriate for us.

Several software solutions make it possible to carry out drawing of the designed algorithm. Based on marketing surveys or the experience of the authors (Duffy, 2017; Fairbanks, 2017; MeraBheja 2017), we can include: Lucidchart, SmartDrawCloud, Creately, Draw.io, Microsoft Visio Pro for Office 365, Gliffy, Edraw, etc. Our goal is not to assess the suitability or choice of specific software. The aim is to document the fact that there are enough software options for creating a flowchart. However, none of these applications support publication of the plotted algorithm directly into the programming language. This is not their primary function. Unlike a wide range of software for drawing flowcharts, the range of software solutions for NS diagrams is very narrow. Edraw and SmartDraw belong to the most known software of this type. Application Structorizer (Structorizer, 2017) is a little tool which is released under the terms of the General Public License (GPU) as published by the Free Software Foundation. This application is constantly upgraded and developed, especially by its original author Bob Fish. Besides creating a diagram, it also allows to generate a source code.

Based on the algorithmization of a task that ends in an appropriate representation of the algorithm, the student continues by its implementation, that is, by programming. Diagrams are an important part of every software development documentation and help students better understand the structure of the code itself, as well as easier to understand the syntax of the control structure of the language architecture. As reported by the Shrag (1978), the programmer can more easily see the link between the operations in the designed algorithm, because the graphical form is much more comprehensible than the code itself. Our main goal is to increase the success of students and help them better understand algorithmization. We see a solution in providing immediate feedback, independent of the place, time and a competent person (teacher/tutor). Therefore, following a thorough analysis of the suitability of existing software solutions, we have decided to supplement Structorizer application with a modulus that, after developing NS diagrams and their subsequent generation into the C programming language, will allow run it through a compiler. Thus, the student receives immediate feedback whether the designed algorithm is functional or not. As with programming, testing can lead to the use of a trial and error method in order to achieve the correct outcome and not a targeted modification of the algorithm based on knowledge. We also consider this method appropriate during training. In the end, the student gets the right result by a suitable modification of the algorithm, and in this process, he/she reveals the solved connections and facts in an interactive manner.

3 Results and discussion

Structorizer is all written in Java. When designing the application, the model-view-controller (MVC) design pattern was partially used, which means there is a hint of a split structure for the graphical interface and application logic. Because the project is open-source, several different companies and professionals worked on it, providing many packages. Everyone who designed a modulus developed a custom package with its classes and own implementation. As a result, this project has become quite unclear after a certain time. Each method is implemented within the related class. Existing implementation of our C language generator is again only one class, in which all the methods necessary for proper code generation are implemented.

As specified in the documentation for Strutorizer (2017), the GUI (Graphical User Interface) is quite minimalist and easy to use (see Fig. 2). It is built of:

- a Toolbar offering shortcuts to features and functions,
- the Menu, which provides most of what the Toolbar does and some more features,
- the Work Area, which is where you create your NS diagram,
- the Report List where the Analyser component (if activated) writes warnings on dubious diagram contents, and
- the Arranger Index, which lists all diagrams currently held in the Arranger tableau in lexicographic order (main programs first, then subroutines)".

The user interface of Structorizer application was changed only slightly. The "Generate C" button is added to the diagrams placed in the Arranger. This button starts generating the code for all diagrams and joins them into one code (Fig. 3). The code is then displayed in a new window (Fig. 4).

In modeling diagrams in Structorizer and subsequent export of the code, we encountered a number of factors that are not perfectly implemented. It is not the subject of this paper to detail these because their modification would mean rewriting the entire application, but we want to point out that we do not see this as a

negative fact because of our goal. After the code has been generated, the student upon his/her knowledge has to read and modify the code into its executable form, he/she ultimately learns again - specifically C syntax.

We illustrate the functionality of the application and its work with a simple algorithm of calculating a quadratic equation roots on a set of real numbers. In the main body of the program (function “main”), quadratic equation coefficients A, B and C are read. The next step is calculating the value of the discriminant. Solution is dependent on a discriminant. If the discriminant is positive, then there are two distinct roots. If the discriminant is zero, then there is exactly one real root, sometimes called a repeated or double root. If the discriminant is negative, then there are no real roots. Rather, there are two complex roots. The user is asked to define the data type of variable only in the resulting code. This call is in the form of a comment that is generated at the beginning of each function. This comment can be seen in Fig. 4. It begins with the word “TODO:” and below are listed all the variables that need to be declared. The control variables required for the "for loop" will not join the above mentioned variables. This may also be because compilers based on the C99 standard support the definition of the variable directly in the “for loop”.

One of the limitation is the need to write mathematical operations as they are written in the programming language. If the mathematical functions are used, the header files are not generated. This error arises because the export counts only the header file "stdio.h", which is added automatically.

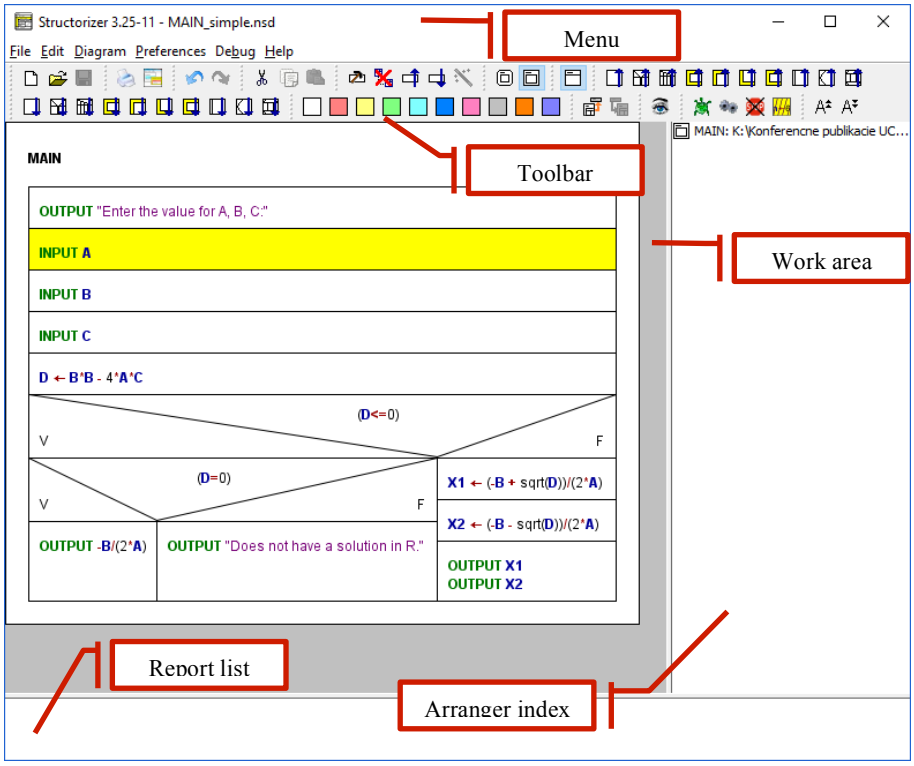

Fig. 2: Structorizer Graphics Interface

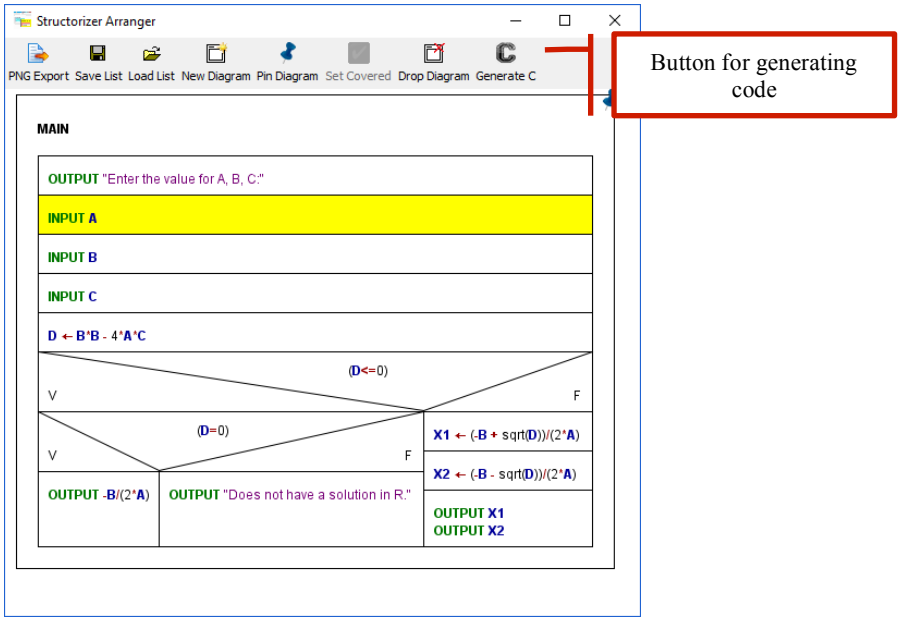

Fig. 3: Arranger of the diagrams for calculating the roots of the quadratic equation

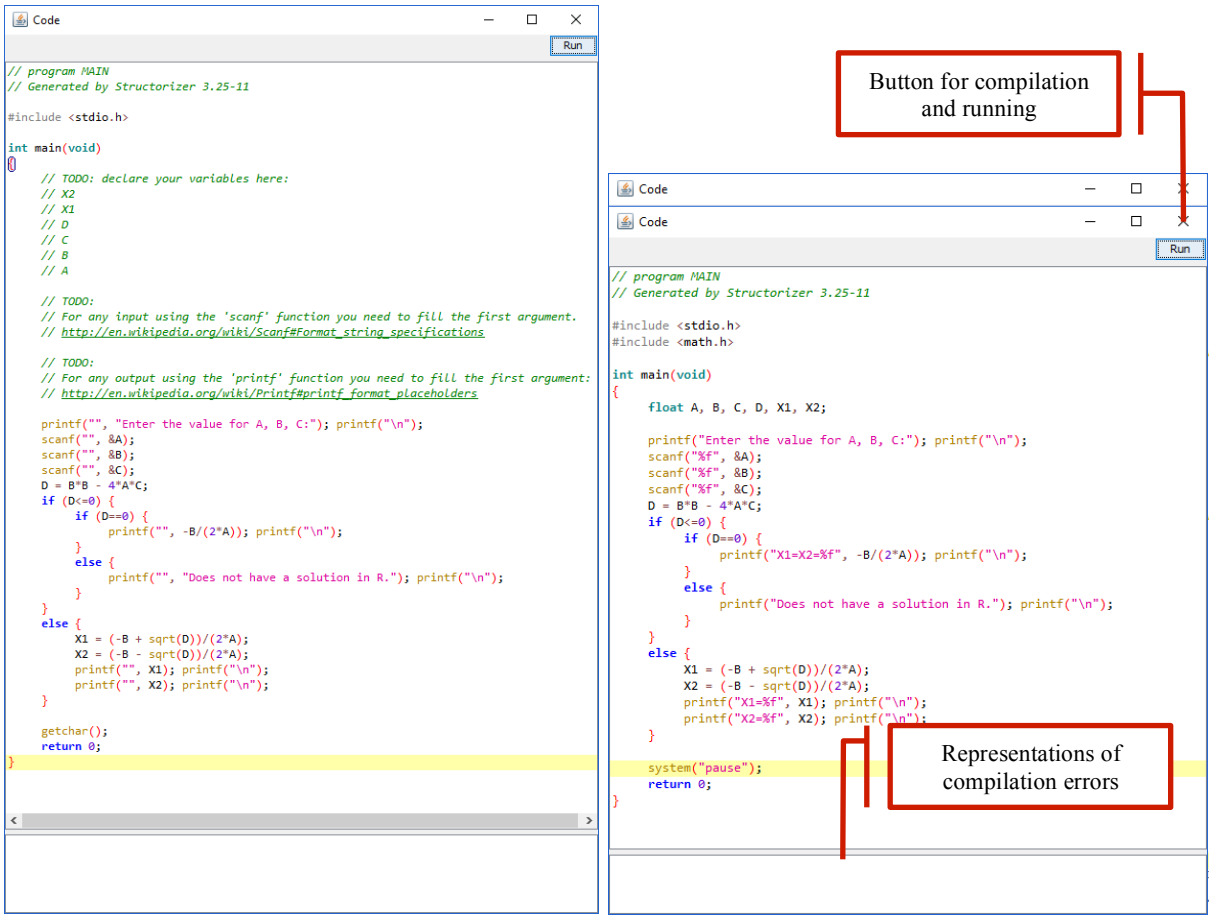

Fig. 4 Exported code based on diagrams and the code modified into executable form

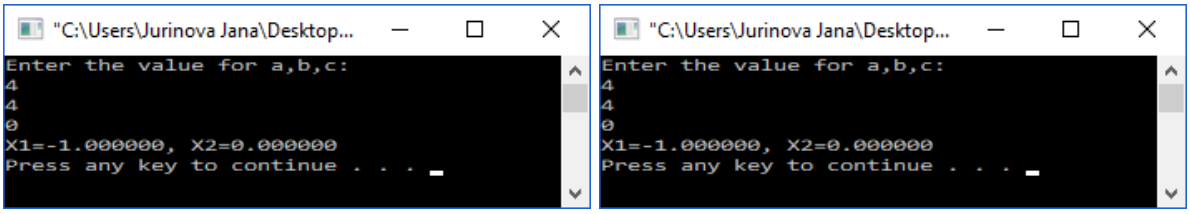

Fig. 5 Demonstration of functionality

4 Conclusion

The application was provided to students for pilot testing in the academic year 2017/2018. The main goal is to fine-tune the functionality of the application to a satisfactory state, as we already know that students have drawn on a few problems when using it. On the other hand, we have registered positive feedback on the application. Consequently, our goal is to carry out a pedagogical experiment and test whether the use of such an application will increase the IT competencies and skills of the students for the purpose of which the application was created.

Acknowledgment

The author gratefully acknowledges the contribution of the KEGA Grant Agency of the Slovak Republic under the KEGA Project 016UCM-4/2017.

Our special thanks go to Marcel Forgáč for technical support in development and testing the application.

References

Avancena, T., Nishihara, A. and Ch. Kondo. 2015. Developing an Algorithm Learning Tool for High School Introductory Computer Science. Education Research International 02/2015; (Article ID 840217). 11 pp. Available on: http://www.hindawi.com/journals/edri/2015/840217/

Diehl, S. 2007. Software visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer New York, 2007.

Duffy, J. 2017. The Best Flowchart and Diagramming Apps of 2017. [online]. [cit. 15.10.2017]. Available at: https://www.pcmag.com/article/349791/the-best-flowchart-and-diagramming-apps

Fairbanks, L. 2017. The Best Flowchart Software. [online]. ©2017 [cit. 15.10.2017]. Available at: http://www.toptenreviews.com/business/software/best-flowchart-software/

Herout, L. 2016. Elektronické studijní opory v prostředí terciárního vzdělávání. Praha: powerprint. ISBN 978-80-7568-016-7.

Jurinová, J. 2013. Rozvoj kognitívnych vedomostí za využitia multimediálnej učebnej pomôcky. In: *XXVI. DIDMATTECH 2013 NEW TECHNOLOGIES IN SCIENCE AND EDUCATION*. Gyor : University of West Hungary, Tribun EU, s.r.o., 2013. s. 136-141. 255 s. ISBN 978-963-334-184-1.

Jurinová, J. 2015. Instruction Videos for Psychomotor Skills Development. In: *R&E-source : časopis pre výskum a vzdelávanie*. Special issue 4, december (2015), pp. 129-133. ISSN 2313-1640.

Jurinová, J. 2016. Identifying the difficult thematic units for increasing the specific IT competences and skills in the field of algorithmization and programming. In : *Proceedings of International Scientific Conference SCHOLA 2016, Pedagogy and Didactics in Technical Education, 12th International Scientific Conference on Engineering Pedagogy*, Published: in 2017, CVUT Prague. Editors: Pavel Andres, Roman Hrmo et al. 2017, pp. 149-154, ISBN: 978-80-01-06112-1.

MeraBheja, 2017. 19 Best Free Tools for Creating Flowchart. [online]. ©2017 [cit. 15.10.2017]. Available at: http://merabheja.com/best-free-tools-for-creating-flowcharts/

Schrag, Cornelia M. Yoder and Marilyn L. 1978. Nassi-Shneiderman Charts: An Alternative to Flowcharts for Design. New York: ACM SIGSOFT/BIGMETRICS Software and Assurance Workshop.

Structorizer. [online]. ©2017 [cit. 15.10.2017]. Available at: http://structorizer.fisch.lu/

Urquiza-Fuentes, J. and J. A. Velázquez-Iturbide. 2009, A survey of successful evaluations of program visualization and algorithm animation systems. ACM Transactions on Computing Education (TOCE) – Special Issue on the 5th Program Visualization Workshop. Vol. 9, No. 2, pp. 1–21. Available on: http://www.researchgate.net/publication/220094505