Algorithmic Thinking as a Prerequisite of Improvements in Introductory Programming Courses

Mario Konecki
Faculty of Organization and Informatics, University of Zagreb, Croatia
mario.konecki@foi.hr

Abstract

There are many persisting problems present in introductory programming courses. Students are not able to deal with abstract and complex programming concepts and in many cases they have troubles in understanding even the most basic programming examples. One of the means of making programming more understandable is the usage of various visualization tools and video materials, but maybe the most important factor that results in greater or lesser ability of students to follow programming lectures and to solve programming tasks is the adoption of an appropriate way of thinking that can be called algorithmic thinking. It enables introductory programming students to analyze given problems and construct appropriate algorithms in order to solve various tasks. In order to promote algorithmic thinking, a series of lectures and examples have been designed and presented to introductory programming students to find out about the effect of algorithmic thinking on the students' ability to follow the introductory programming courses.

Keywords: Introductory programming courses, Students, Algorithmic thinking, Problem-solving skills

Izvleček

Algoritmično razmišljanje kot pogoj za napredovanje študentov pri predmetu uvod v programiranje

Pri poučevanju programiranja se srečujemo z več perečimi problemi. Študenti imajo težave z razumevanjem abstraktnih in kompleksnih konceptov programiranja, številni celo težko razumejo primere preprostih programov. Pripomočki, ki lahko olajšajo razumevanje programiranja, so različna orodja za vizualizacijo in video gradiva. Najpomembnejši dejavnik, ki vpliva na sposobnost sledenja predavanjem in zmožnost reševanja programerskih nalog, je usvojitev algoritmičnega razmišljanja. To študentom omogoči, da analizirajo dane probleme in sestavijo primerne algoritme, s katerimi jih rešijo. Da bi preverili vpliv sposobnosti algoritmičnega razmišljanja na sposobnost sledenja uvodnim predavanjem iz programiranja, smo zasnovali zaporedje lekcij in primerov, ki spodbujajo algoritmično razmišljanje, ter jih uporabili pri študentih.

Ključne besede: tečaj programiranja za začetnike, študenti, problemi, algoritmično razmišljanje, sposobnost reševanja problemov.

1 INTRODUCTION

Both students and teachers agree that introductory programming courses are difficult. Students find abstract programming concepts hard to understand and a rather large number of different syntactic rules makes introductory programming even more challenging. Many students tend to stop following lectures in an active manner and become unable to do their homework tasks. This situation has a negative effect on the motivation of students and it also creates fear and unwillingness of students to learn programming. There are many reasons for this kind of state, but maybe the most important reason is the lack of one critical prerequisite for any programming activity, namely a very specific way of thinking that is required to analyze and solve problems. This kind of thinking can be named problem-solving skills or simply algorithmic thinking and it is not a part of students' habitual way of reasoning, which is the main reason that students lack this kind of insight into the analysis of given problems and into the construction of corresponding algorithmic solutions.

Students actually perform all necessary activities that are needed in order to develop proper algorithmic thinking on a daily basis, but they do all these activities in such a quick way that makes these activities virtually unconscious. These activities include the analysis of problems and the formation of a series of steps for their resolution. The person who wants to get a glass of water knows that he/she needs to find a glass, check whether it is empty, find a bottle of water, check if there is any water in it, open it if necessary, pour the water, etc. All these actions and conditions are something that the human brain processes automatically without the person being aware of every single step. In the case of programming, students have to become aware of every single step and the conditions needed to solve a problem. There is a gap between the intuitive way of students' thinking and the way of thinking that is needed to perform programming activities.

Students already have some knowledge about programming concepts and structures. For example, students experience troubles in understanding how arrays function (Milne & Rowe, 2002; Lahtinen et al., 2005) but they know perfectly well how trains look and that there are trains with different numbers of wagons, which is analogous to arrays and the way they function. Analogy and metaphors can be used to make programming concepts more familiar and comparable to already known elements from everyday life.. Another possible approach that can help in making programming concepts more understandable includes visualization and various video materials. In this way students are presented with means that enable them to see programming concepts dynamically and to connect these processes with analogous processes from real life by usaging various metaphors which promote constructivism and building students' programming knowledge on already known concepts from their everyday life.

By decreasing the number of obstacles that students encounter in their introductory programming education, their motivation to learn programming would be increased and the fear of programming would be decreased. It is also important to make students aware of the importance of programming for their future career. When talking about visualization, there are many different tools available (Price et al., 1992; Stasko, 1992). These tools include a variety of elements, from static images to animations, video and/or audio media. Regardless of existing efforts, problems in introductory programming courses are still present and persistent with high reported failure rates (Gomes & Mendes, 2007b). In this paper a number of existing visualization approaches and tools are presented and discussed along with the research on the effect of the development of algorithmic thinking in introductory programming students on their passing rates and the quality of their programming skills.

2 PROBLEMS IN INTRODUCTORY PROGRAMMING COURSES

The attitude that denotes programming as hard and challenging to learn is commonly accepted by many authors (Gomes & Mendes, 2007a; Smith & Webb, 2000; Robins et al., 2003). A number of new abstract programming concepts as well as rather large number of various syntactic rules that need to be adopted make introductory programming problematic for students to understand. Learning programming is very similar to learning any spoken human language since it consists of syntactic rules and semantics. The main difference is that programming is used for describing problems that are usually not a part of everyday life compared to various spoken languages which describe more familiar life situations and this makes programming rules and syntax more challenging to learn. There is a need to address these detected difficulties in all programming courses but especially in introductory programming courses because programming represents a vital part of any computer expert's education and, as such, is a crucial part of all computer science studies (Gomes & Mendes, 2007b).

In programming there is a need to master not only syntax but also many abstract concepts and principles of programming, and this fact results very frequently in a situation in which students tend to deal more with either syntax or concepts, which in the end results in a lack of knowledge or skills in either of the above mentioned parts. The need to put a lot of effort into two major parts makes programming even more difficult. The fear and skeptical attitude towards programming also affect the number of students that are willing to study computer science (Bennedsen & Caspersen, 2007). Along with the fear of programming, a lack of motivation and poor understanding of the role of programming in their future professional career, there are also other aspects that affect students and are part of the causes that make programming problematic for students. Some of these aspects are (Gomes & Mendes, 2007a):

- Programming demands a high level of abstraction.
- Programming calls for a good level of both knowledge and practical problem-solving techniques.

- Programming requires a very practical and intensive study, which is quite different from what is required in many other courses (which are based more on theoretical knowledge, implying extensive reading and some memorization).
- Usually teaching cannot be individualized due to class size.
- Programming is mostly dynamic, but usually taught using static materials.
- Teachers' methodologies often do not take into consideration students' learning styles. Different students have different learning styles and can have different preferences in the way they learn.
- Programming languages have a very complex syntax with characteristics defined for professional use and not for pedagogical motivations.

The discussion about the reasons for problems in introductory programming courses is still ongoing and it is very difficult to draw clear conclusions about the reasons of all reoccurring problems. Some of the responsibility for this kind of state lies with teachers, but some of it also lies with students. Programming is a skill and because of that it requires a somewhat different approach than most other courses which require memorization and reinterpretation of some facts. Programming as a skill also requires a longer time of constant practice. All of this is not a part of students' learning habits. Students are used to memorize facts and to learn in less time rather than in a prolonged way. The way of learning and practicing needed to develop a skill is, along with algorithmic thinking, something that students need to learn as a prerequisite for successful programming. Visualization tools and video materials are some of the means of making programming concepts and principles simpler and clearer for students because most of them are visually oriented (Hu, 2004).

3 VISUALIZATION AND VIDEO IN INTRODUCTORY PROGRAMMING COURSES

Various visualization techniques as well as the usage of video materials aimed at making abstract programming aspects more understandable and easier to imagine to students compared to traditional static materials which leave a lot of space for students' own understanding and constructions of presented programming concepts which are in many cases inaccurate and flawed. Animations make it possible for students to gain a deeper insight into the processes and dynamics of algorithms which is something that is much harder to explain by using static materials. Although there are many visualization tools available, most of introductory programming courses still mainly use traditional means of education without the possibilities offered by contemporary technology. The reason for this could be the nonexistence of specific tools to suit every particular teacher's needs and the nonexistence of proper development environments or special skills that teachers would have to posses in order to develop such tools (Naps et al., 2005). Unwillingness of teachers to deal with visualization tools and to create such tools is another reason for the absence of this kind of tools along with the lack of time that is needed for the development or preparation of special visual presentations or video materials. Another fact that needs to be considered is that many teachers are simply used to traditional ways of teaching and are not eager to adopt new technologies or change their habitual ways of work. Usefulness and possibilities of various visualization techniques are something that should be made better known and available to teachers in order to make programming more interesting and clearer for their students.

Interaction is another important aspect for capturing students' attention and their proper programming skill development (Pears et al., 2007). Interaction is important because it promotes active involvement in the learning process and it also promotes better focus. Static and dynamic means are beneficial in bringing programming closer to students and these means include tools and techniques such as (Hu, 2004):

- Static:
  - code lists
  - flowcharts
  - diagrams
  - pictures
- Dynamic:
  - one-way presentations
    - movies
    - PowerPoint slides
    - Flash animations with audio explanation and music
  - Two-way interactive tools:
    - executable flowcharts
    - algorithm animations
    - program visualizations

There are many existing visualization tools available to teachers such as: XTANGO (Stasko, 1992), MRUDS (Hanciles et al., 1997), JavaVis (Oechsle & Schmitt, 2002), JHAVE (Grissom et al., 2003), BlueJ (Kölling et al., 2003), Jeliot3 (Moreno et al., 2004), TRAKLA2 (Malmi et al., 2004), Raptor (Carlisle et al., 2005) and ALVIS LIVE! (Hundhausen & Brown, 2007). Visualization tools can be beneficial to students in many aspects of programming (Sorva et al., 2013), but the usage of this kind of tools still shows variations in research results (Bennedsen & Caspersen, 2007; Clancy et al., 2001) and there is a need for more attention and testing of this kind of tools. Animation of programming elements along with interaction promotes better understanding and greater involvement of students in their own learning process, which supports constructivism as a valid approach that captures students' attention and provides a beneficial approach for acquiring of some particular skill, in this case programming.

Another way of making programming concepts clearer is the usage of video materials which can be used as standalone online materials or as part of classroom lectures. In the case of online video materials, there is an advantage of constant availability and the possibility of viewing these materials at one's own pace and at a desired time. Video materials that are used in the classroom are shown to clarify certain programming concepts but are less flexible regarding time and pace. The main advantage of this kind of approach in the usage of video materials is the presence of students' peers as supporters and the teacher as a moderator and facilitator of more focused student activities (Ward & Newlands, 1998), as well as a greater amount of presented practical program examples because more time is available for practice, since a part of the theoretical knowledge is covered by video materials. Online video materials, on the other hand, promote a greater amount of students' individual work. A very important aspect when talking about video materials is the proper design of these materials, which in the end results in different impacts on students. Research has shown that simple visualization tools that include some form of interaction can make programming more interesting and programming concepts clearer (Konecki & Mrkela, 2014). Research has also shown that well-designed video materials can promote greater understanding and motivation of students to learn programming (Konecki, 2014).

4 PROMOTING ALGORITHMIC THINKING IN INTRODUCTORY PROGRAMMING COURSES

Algorithmic thinking includes the whole process of constructing a solution for a given problem. It consists of a proper decomposition and analysis of a given problem and the construction of a proper algorithmic solution that addresses the given problem as a whole. In order to conclude about the effect of algorithmic thinking on students' ability to understand programming concepts and construct proper algorithmic solutions, an appropriate study has been conducted. The study included 121 students who have enrolled into an introductory programming course and have gone through its entire curriculum. All students have gone through 12 hours of lectures, which have been designed to teach students about algorithmic thinking. Practical examples and interaction with the students were a major part of these lectures, since the lectures consisted of many practical problems that were analyzed and decomposed into smaller parts, which were described and connected with their appropriate algorithmic solutions in order to compose a larger algorithm that addressed the whole problem given to students.

An important aspect of the conducted study is the way in which various problems were presented and solved. Rather than just stating the problem and coding its solution, the process of solving the problems was conducted using several steps:

- State the problem, the main input and the main output of the problem.
- Think about the problem parts and their outputs.
- Draw problem parts and their connections.
- Draw an algorithmic solution.
- Design the code for solving the problem.

It is also important to mention that a number of problems were simple everyday problems such as sorting the students' tests, inspecting a car's fueltank, etc. in order to promote algorithmic thinking and problem-solving skills in which students are required to think step by step, to include all necessary steps, to include all conditions, to repeat certain actions and to include all other aspects that are commonly found as constructs in programming.

The traditional way of teaching mostly relies on theoretical lectures with some examples and on practical tasks that are presented to students, as well as on the tasks that students need to solve by themselves. The problem is that this kind of teaching does not address the need of students to understand the process of decomposition and analysis of problems, as well as the process of constructing the overall solution from smaller elements and constructs. Students are presented with various programming concepts and constructs which are abstract in their nature. This abstract nature makes students struggle with the presented concepts and most students simply cannot track and understand these concepts along with syntax alone.Moreover, they cannot simultaneously focus on the concepts, syntax, the process of problem decomposition, the analysis and the construction of a proper algorithmic solution. There are simply too many aspects involved in programming for most students to understand and learn them all at once. In the conducted study, the students were trained in algorithmic thinking, which promotes problem--solving skills: problem decomposition, problem analysis and constructing the solution step by step. In this way students learn how to design algorithmic solutions and are later able to be more focused on programming concepts and syntax which they then incorporate in their already designed solutions. In this way students learn in an incremental way and are not overwhelmed, which in the end should produce better results and greater understanding of programming.

12 hours of lectures were divided into 4 parts that were presented to students with a one-week break between each part. At the beginning of each part, the students were given several programming examples and they were asked to describe and explain certain parts of these examples, as well as to change certain parts in order to implement some particular change in the way that programs function. After the 3-hour long lecture with practical tasks and visualization elements, students were again given several programming examples and they were again asked to describe, explain, or change certain parts of these examples. The results of these tasks were compared to conclude whether there were any improvements in the results after the conducted lectures. These comparisons were made in all 4 parts of the conducted lectures. The results represent an objective evaluation of the effects that have been achieved by the conducted lectures. The students were also given a questionnaire after the last conducted lecture in order to asses their opinion and attitude towards the conducted lectures, as well as to assess their perception of these lectures and whether education on the algorithmic reasoning has been beneficial and useful for the development of programming skills and better understanding of their programming course curriculum materials. In this way an objective and subjective study was conducted in order to obtain a more realistic conclusion about the effectiveness of such education.

The objective analysis was conducted through the tests that were given to students before and after every conducted lecture. All tests were composed of questions from a particular lecture area and the lecturer was not aware of the exact questions and tasks that were included into these tests. The tests used before and after every lecture were designed in a way which ensured that they were different but as even as possible in terms of their difficulty. All students were tested simultaneously so there was only one set of questions for every particular test. The results of the objective study have shown that the understanding of programming concepts and the students' ability to deal with programming tasks had increased as a result of the conducted lectures on algorithmic thinking and that the style used in these lectures was efficient. The study has shown an absolute increase of 24% or more in the accuracy of the tests when considering the maximum possible number of points which represents a relative increase of 65% to 100% after every part of the conducted lectures compared to the students' prior skills and knowledge in the area that was covered by a particular part of the conducted lectures. The results of the objective study are shown in Table 1.

Table 1. **Accuracy percentage of the tests before and after every of the four conducted lectures**

Test	**Lecture No. 1**	**Lecture No. 2**	**Lecture No. 3**	**Lecture No. 4**
Before lecture	43%	37%	34%	24%
After lecture	71%	69%	62%	48%

A paired-samples t-test was conducted to compare the students' test scores before and after the conducted lectures (each test had a maximum score of 10 points). There was a significant difference in the scores before the first lecture (M=4.30, SD=0.95) and after the first lecture (M=7.10, SD=1.15); t(120)= -21.299, p=0.000. Also, there was a significant difference in the scores before the second lecture (M=3.70, SD=1.26) and after the second lecture (M=6.90, SD=1.47); t(120)=-17.496, p=0.000. Further on, the difference was also statistically significant between the average score before (M=3.40, SD=1.09) and after the third lecture (M=6.20, SD=1.32); t(120)=-17.647, p=0.000. The difference in scores was also statistically significant before (M=2.40, SD=1.07) and after the fourth lecture (M=4.80, SD=1.24); t(120)=-16.325, p=0.000. These results suggest that the conducted lectures do indeed facilitate a better understanding of programming concepts and improve the quality of students' knowledge and skills.

The results of the subjective questionnaire survey have shown that students find the conducted lectures about algorithmic thinking useful, interesting, motivating and effective. The style, methods and techniques used in the conducted lectures have made programming more understandable to the students and their motivation to deal with programming has increased. Students have reported that the conducted lectures have increased their understanding of abstract programming concepts and that they would like to have the same lecture style in their programming courses. Students have also stated that they would recommend this kind of lectures to their colleagues. The results of the conducted questionnaire survey are shown in Table 2.

Table 2. **Results of the conducted questionnaire survey**

Questionnaire item	**Mean**	**Std. dev.**
I find the lectures on algorithmic thinking useful.	4.20	1.19
The lectures on algorithmic thinking have made programming more understandable for me.	3.78	1.21
The style, techniques and methods of teaching used in the lectures on algorithmic thinking are more engaging and interesting for me compared to traditional teaching.	4.21	1.11
I have found the lectures on algorithmic thinking to be of little use to me.	1.60	0.80
I prefer traditional methods compared to the teaching that was used in the lectures on algorithmic thinking.	1.57	0.86
I would recommend the lectures on algorithmic thinking to my colleagues.	4.14	1.13
I think that I have a greater chance to pass my programming exams after the lectures on algorithmic thinking.	4.10	1.17
I would like to have the same style of teaching that was used in the lectures on algorithmic thinking in my programming course because this kind of style makes programming more understandable to me.	4.25	1.02
The way of teaching that was used in the lectures on algorithmic thinking increases my motivation for learning programming.	3.76	1.23

The results of the conducted objective and subjective studies have shown that algorithmic thinking is indeed an important aspect and a prerequisite of improvements in programming courses and something that needs further research and attention.

5 CONCLUSION

Although programming is gaining importance in the modern business world, it is perceived as hard to teach and learn. There are many reoccurring problems and difficulties that are part of introductory programming education. Students experience fear and a lack of motivation to deal with programming because of their inability to cope with its abstract nature and the inability to perform well on programming tasks. Programming includes many aspects that are not intuitive and require a special kind of knowledge and skills as a prerequisite of successful solving of programming tasks. Various visualization tools and video materials have been developed and have shown to be beneficial for clarifying certain programming concepts in the sense that these tools and materials make these concepts simpler to imagine and understand as well as to apply.

A set of skills and knowledge that has been termed algorithmic thinking has been recognized as an important prerequisite of students' successful results in introductory programming courses. Some authors refer to algorithmic thinking as the base of programming (Milkova, 2005). Algorithmic thinking has been recognized and mentioned by other authors who have presented their own efforts to facilitate this kind of thinking, such as posing similar problems of increasing size (Burton, 2010), or training students in algorithmic thinking by letting students play algorithms themselves (for example to find a way to identify the oldest student) (Futschek & Moschitz, 2010). Algorithmic thinking is important because students have shown that they have more problems with analyzing problems and constructing algorithmic solutions than with remembering the syntax because memorization is something that most students have become accustomed to during their past education. In order to conclude about the effectiveness of algorithmic thinking as a prerequisite of improvements in programming courses, a study has been conducted in which 121 students went through 12 hours of lectures which were divided into 4 parts. These lectures were designed to promote algorithmic thinking and problem-solving skills, which was achieved by using several teaching elements: a number of understandable examples, visualization of presented examples and intensive interaction with students. The students were tested before and after each conducted lecture to objectively asses their ability to deal with programming tasks. These results were compared to conclude whether lectures on algorithmic thinking had any positive effect. The students were also given a questionnaire at the end of the lectures in order to conclude about their subjective perception of this kind of education. The results of the conducted study have shown that algorithmic thinking is indeed an important aspect that promotes a better understanding of problems and enables students to perform better in their programming tasks, which is consistent with the results of previously mentioned research. The study has also shown that students perceive this kind of education as useful and interesting. Further research about the aspects that should be included into this kind of education and further testing of its effects will be a part of future research.

6 LITERATURE

[1] Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. ACM SIGCSE Bulletin, 39(2), 32–36.

[2] Burton, B. A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

[3] Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: A visual programming environment for teaching algorithmic problem solving. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pp. 176–180, St. Louis, Missouri, USA.

[4] Clancy, M., Stasko, J., Guzdial, M., Fincher, S., & Dale, N. (2001). Models and areas for CS education research. *Computer Science Education, 11*(4), 323–341.

[5] Futschek, G., & Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. In *Proceedings of the 2010 Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century*, pp. 1–10, Paris, France.

[6] Gomes, A., & Mendes, A. J. (2007a). An environment to improve programming education. In *Proceedings of the 2007 international conference on Computer systems and technologies*, pp. 88:1–88:6, ACM, New York, USA.

[7] Gomes, A., & Mendes, A. J. (2007b). Learning to program--difficulties and solutions. In *International Conference on Engineering Education–ICEE*, pp. 283–287, iNEER, Portugal.

[8] Grissom, S., McNally, M., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the ACM Symposium on Software Visualization*, pp. 87–94, San Diego, California, USA.

[9] Hanciles, B., Shankararaman, V., & Munoz, J. (1997). Multiple representation for understanding data structures. *Computers & Education, 29*(1), 1–11.

[10] Hu, M. (2004). Teaching novices programming with core language and dynamic visualization. In *Proceedings of the 17th NACCQ*, pp. 94–103, Christchurch, New Zealand.

[11] Hundhausen, C. D., & Brown, J. L. (2007). What you see is what you code: A 'live' algorithm development and visualization environment for novice learners. *Journal of Visual Languages and Computing, 18*(1), 22–47.

[12] Konecki, M., Mrkela, V. (2014). Algorithmic thinking and animated interactive presentation of sorting algorithms in education of students. In *Conference Proceedings of VIVID 2014 (Education in Information Society)*, pp. 105–112, Faculty of Organizational Sciences, Kranj, Slovenia.

[13] Konecki, M. (2014). Using video lectures in introductory programming courses. In *Proceedings of IAC-ElaT 2014*, pp. 256–260, Czech Institute of Academic Education z.s., Vestec, Czech Republic.

[14] Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13*(4), 249–268.

[15] Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*(3), 14–18.

[16] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. Informatics in Education, 3(2), 267–288.

[17] Milkova, E. (2005). Developing of algorithmic thinking: the base of programming. *International Journal of Continuing Engineering Education and Life Long Learning*, 15(3-6), 135–147.

[18] Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming – views of students and tutors. *Education and Information technologies, 7*(1), 55–66.

[19] Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 373–376, Gallipoli, Italy.

[20] Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R. J., Anderson, J., Fleischer, R., Kuittinen, M., & McNally, M. (2005). Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin, 35*(4), 124–136.

[21] Oechsle, R., & Schmitt, T. (2002). JAVAVIS: Automatic program visualization with object and sequence diagrams using the java debug interface (JDI). In *Lecture Notes in Computer Science, Vol. 2269: Software Visualization*, pp. 176–190, Springer-Verlag, Berlin, Germany.

[22] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39*(4), 204–223.

[23] Price, B. A., Small, I. S., & Baecker, R. M. (1992). A taxonomy of software visualization. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, pp. 597–606, IEEE, Los Alamitos, California, USA.

[24] Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Journal of Computer Science Education, 13*(2), 137–172.

[25] Smith, P. A., & Webb, G. I. (2000). The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers. *Journal of Educational Computing Research, 22*(2), 187–216.

[26] Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE), 13*(4), 15.

[27] Stasko, J. (1992). Animating algorithms with XTANGO. *ACM SIGACT News, 23*(2), 67–71.

[28] Ward, M., & Newlands, D. (1998). Use of the Web in undergraduate teaching. *Computers & Education*, 31(2), 171–184.

▪

Mario Konecki graduated in 2005 at the Faculty of Organization and Informatics where he has also earned his doctoral degree in 2013. Since 2005, he has been working at the Faculty of Organization and Informatics as an assistant and later as a senior assistant. His main areas of interest are programming education, graphical user interface design, the development of programming languages and the development of intelligent systems. During his work at the Faculty of Organization and Informatics, he has published over 30 scientific and professional papers and he has also worked on 8 scientific and professional projects, one of which was closely connected with his doctoral thesis which dealt with the development of a new programming language for the inclusion of the visually impaired in activities of graphical user interface design.