PROGRAMMING NOVICES' MENTAL MODELS

Nikolina Bubica1, Ivica Boljat2

1 Mokosica Elementary School, Dubrovnik (CROATIA)
2 Faculty of Natural Science, Split (CROATIA)

Abstract

Achieving success or failure at the beginning of the introductory programming course often determines whether the student will continue to track content related to computer science. Despite numerous studies, it is still not entirely clear why some students of introductory programming courses learn to program faster and better than others, and why, for some, programming is funny while others see it as an arduous struggle in an effort to pass one course. In an effort to discover the factors that influence the achievement of success in introductory programming course, a growing number of researchers are studying the importance of students creating consistent mental models of fundamental programming concepts. This paper aims to give an overview of the relevant literature that examines the impact of novice programmer creating consistent mental model to his learning programming in general, lists the advantages and disadvantages of the application of certain teaching techniques with respect to the student's engagement, emphasizes the importance of student's high self-efficacy for knowledge adoption and transfer, compares the effectiveness of individual tools, visual or Command Line, which aims to assist students in creating consistent mental models in learning programming and understanding how computer program really works and finally gives some pedagogical implications.

Keywords: mental models, novices, teaching programming, notional machine.

1 INTRODUCTION

Learning is a part of every person daily routine. Some changes in our behavior are the result of the maturation of the person, and some are the consequences of learning. What really affects the success in learning? Learning could make new forms of person’s behavior, provided that his or her abilities are at a sufficient level of development, naturally if the person is sufficiently motivated and if they are favorable external opportunities for learning. The absence of at least one of these factors leads to the impossibility of gaining experience in any shape.

Computer science, especially programming, is seen as very difficult to learn and teach. This is why it is important to understand the way students learn, before we try to teach them. The traditional teaching methods have not proved particularly successful in computer coding and problem solving area because those skills are best acquired through experience and practical work. The current education in computer science seems very inefficient. Well-known multinational McCraken’s group research conducted in 2001 studied the students’ programming ability after introductory programming course and found out that most of the students achieved a lot worse result than their teachers expected [1]. We find encouraging the results of the similar more recent research conducted in 2013 which indicated that there is still a significant number of teachers who show a certain resistance to necessary changes in the teaching process, although progress has been made in understanding and expectations related to the performance of students in programming, also in application of strategies that require greater involvement of students in the work and pay special attention to any form of interaction and collaborative engagement [2].

Teaching introductory programming courses is considered as fundamental skill necessary for progression of students within the Computer Science colleges. A large number of researchers report about the withdrawal from the introductory programming course or falling off that course and that these rates vary by up to 30% of the total number of enrolled students. Learning programming is definitely a personal experience for each student and it is still not clear which difficulties encounter students while they are trying to write a small program.

Numerous researches concerning factors important for achieving success in initial courses of programming highlight several factors that could be powerful enough to influence a novice programmer to successfully adopt basic programming concepts. This work particularly deals with the mental models of fundamental programming concepts which develop students after an introductory programming course. It will highlight the impact of mental models on the effectiveness of learning in
the introductory programming course and display the results of different studies that have attempted to answer the question whether a particular teaching strategy, teaching design or the application of various tools in the classroom could positively influence the development of valid mental models of fundamental concepts by programming beginners.

2 CONSTRUCTIVISTIC APPROACH OF TEACHING PROGRAMMING

The cognitive approach to learning, which dates back to the first half of the twentieth century, is largely based on the theoretical principles about the development of cognitive schemes and thinking structures which represent certain external or internal phenomena or processes. Learning is related to acquiring new schemes and adaptation to new needs. Effective teaching using cognitive theories requires first to determine the type and level of development schemes that students already possess and then to shape the process and the content in a way that allows further development, changing and complementing existing schemes. The schemes that an individual possesses during their development are becoming richer and more complex.

In the last two decades, a new approach to teaching is reflected through teaching methods arising from the theory of constructivism. Constructivism is a learning theory which claims that students construct knowledge, and not just receive and store knowledge transferred by the teacher. This theory argues that knowledge is rather actively constructed by students than passively absorbed from textbooks and lectures. Building of knowledge happens recursively through the existing knowledge which student already possesses. That knowledge consists of facts, ideas and beliefs. Each student constructs his own version of knowledge. Such teaching techniques should be more successful than traditional techniques, because inevitably lead to the construction of knowledge [3]. Ben-Ari [3] states that the novices don’t have an effective computer model. Efficient computer model is every cognitive structure that the student can use to create sustainable construction knowledge. Such structures are formed on the basis of sensory experiences such as reading, listening, lectures and work on the computer. The task of student beginners is to build from scratch an idea of how the computer works. In programming, in particular, that building is associated with foresight and understanding of what happens during execution of a program. It is extremely important for teachers to recognize the mental models of their students, respectively the ideas and knowledge that form their students according to concepts being taught. The research of mental models is needed especially for teachers to properly prepare for classes, to apply effective teaching strategies and to adopt teaching materials to their students [4].

2.1 What are mental models?

Mental models are part of almost all human activities. We form, share, change, develop and use such models in all, both private and business communities to understand ourselves and the world around us. We use mental models to analyze and understand a phenomenon and then to design and create our own masterpieces. According to Craik [5] our brain creates “reality models of small scale” that could be used to predict or understand the events and support these events. Norman [6] stated that mental models meet three basic conditions: mental models reflect the beliefs about the observed system of the person who owns them, the system can be observed and this system cope prediction. Further research of mental models recognized theirs common characteristics independent of a concept to which they relate to. Mental models are unfinished and simplified, variable, have blurred boundaries, are non-scientific and sparse and often contain biases, also there is certain time lag in their changing. Although there are many studies dealing with human mental models of natural phenomena that surround us there are very few studies that have explored specifically mental models of novice programmers [7].

Problems that students have in learning introductory programming are discussed for more than 30 years, but very little has changed in that time. Much research work studies the difficulties of students in reading and explaining the very small pieces of code, but the general conclusion is that many students in their first, second and even third course do not understand the semantics of the assignment statement or sequence of instructions [8, 9]. Learning programming includes creation of sustainable mental models of fundamental programming concepts. Sustained mental model is a model that can be integrated in the general model of programming and must always conform to the actual model which is showing [9]. Students with sustained mental models manage significantly better the programming tasks than those with unsustainable mental models [7, 9]. An approach to teaching programming that emphasizes constructivism instead objectivism is proposed to improve students’
mental models. Constructivism supports the assumption that many students, before turning to the subject of the introductory programming, have already deeply rooted ideas about some computer concepts such as assignment statement or sequence of instructions. When teachers begin teaching novice programmers they believe that they all start from scratch, but their students actually came with already formed ideas about some of programming concepts which were formed on some other course like mathematic or another real-time experience. All technical literature agrees that learning and teaching programming is difficult. Bornat [10] pointed out that problem in learning programming start very early, with the assignment statement (Fig. 1). Students who manage to overcome this concept meet the next obstacle in the Loop and recursion structure. It looks absurd to teachers and professional developers both that students bend precisely on the concept of assignment, but this is really happening and many of them do not overcome this obstacle at the end of their introductory programming courses.

![Figure 1: Difficulties in learning programming through introductory programming course](image)

2.2 How to recognize student's mental models?

Great number of researchers have noted, Mayer [11] and Bonar & Soloway (1983) probably the first, that novice programmers develop predictable categorized errors. On this track Dehnadi [12] created a survey to test the development of novices’ programmer mental models on their first programming course and discovered a surprising consistency in students’ responses and the use of specific mental models for solving tasks related to assignment statement. Their first thought was that they discovered the programming aptitude test which could predict the success of novice programmers on the basis of their mental models of basic programming concepts. The existence of a test that could unambiguously predict success of novice programmers at the same beginning of the course and to set aside those that are considered to have the talent for programming could be of a great interest for teachers in any introductory programming course. Nevertheless, the various studies that have followed have failed to confirm Dehnadi argument with the same vigor [13] but we can say that Dehnadi survey really shows some power to predict success based on students mental models [14, 10, 13]. On the other hand, the test has proven to be ineffective for students who have previous experience of programming and students who have poor knowledge of mathematical algebra [14]. Creation of online version of Dehnadi questionnaire and a tool for automatic analysis made a great progress in creating a reliable measuring instrument for recognizing mental models of novice programmers and extremely facilitated the implementation of experiments related to the research of mental models programmers and analysis of its results. The use of this measuring instrument showed that Dehnadi experiment was incomplete if interview wasn’t applied after the implementation of online questionnair e. Also it showed statistically significant correlation of viable model models recognized by Dehnadi questionnaire with the success on the final exam introductory programming and nonviable models and failure on final exam introductory programming [15]. Application of interviews in experiment revealed presence of some so far not recognized answers representing mental models which were formed by novice programmers.

In an effort to help students to learn programming it is essential for teachers to perceive and recognize incorrect conceptualization formed by students about basic programming concepts. Since there is neither easy and very accurate way of testing their mental models, nor it is possible for each student to be interviewed in order to at least partially reveal difficulties he encounters, besides the various tests that are being developed we should mention an interesting approach of creating an inventory of
programming concepts [16]. The basic idea is to identify hard concepts that could make it difficult for students and interview students to better understand their misconceptions. Collected results are used to develop testing and evaluation instrument of concept inventory. The survey revealed a much larger number of misconceptions about memory models than it was expected, and the lack of basic understanding of object concept. Using such approach should give teachers a signal what they should address more in their teaching and what are critical points of the programming context. The use of software tools for improving mental models will be more discussed in section 4.

2.3 Previous programming experience – does it make a difference?

Students in introductory programming courses come with different prior knowledge and perceptions of fundamental programming concepts. Some were introduced with programming in junior high or high school, and some have first contact with this kind of abstract thought at the university. Many studies have analyzed predictors of programming effects of early experience programming on student success in his introductory programming course and the results indicate that previous experience of programming has not shown the strength of the prediction of success or impact on the success of novice programmers [17, 18, 19, 20, 21] but it does not mean that the existence of programming experience is irrelevant. Student’s earlier high-school programming experience influences the perception of his abilities even in the introductory programming course. The study of mental models of experts and novice programmers have shown that beginners are prone to different mental models of experts and novices tend to develop mental representations consisting of specific information while experts develop mental models that contain functional information [22]. Beginners tend to create mental representations with less interconnection of experts [23]. What student knows he presents with his mental models and what he believes about himself he presents with his self-efficacy. Self-efficacy is defined as an individual assessment of his or her ability to perform the task within a specific domain and is known to affects his or hers performance during course. Student’s mental models of programming concepts influence his self-efficacy for programming and finally it influence the success at the end of introductory programming course [14].

2.4 Applying Notional Machine for teaching introductory programming

To be successful in programming, student must be able to create a valid and consistent model of the machine that is running his commands. Misconceptions as well as limited understanding of programming concepts are usually cause of problems and frustrations to novices. Dynamic side of computer program that exists at runtime is very challenging for some novice programmers to grasp. So, it is reasonable to wonder whether the students have problems with programming because they don’t understand how computer really works and how it executes program instructions. The creation of a model that explains it is not simple and the community of computer education has not yet found a common abstract model. Students often form ad-hoc models which don’t guarantee consistency so we could say that both teachers and students could benefit from the existence of some common consistent mental model or notional machine which helps during learning programming. Notional machine is an abstraction created to offer a model to understand the constructs of a programming language or program execution [24, 25]. This machine doesn’t have to accurately reflect exact properties of real machine, but it represents higher conceptual level, which is considered to be easier to understand than the actual machine. Today, the most common technique in teaching program execution is still drawing diagrams by hand or on a board. In recent times there are attempts to use educational software system for presentation and animation of notional machine [24, 25]. Since most currents notional machines apply in first programming years their focus just on the content covered in these courses are basic concepts of programming such as assignment sequence of instructions so software tools used to help students cover mostly basic programming concepts. This could be their advantage as well as disadvantage.

3 TEACHING STRATEGIES AND DESIGNS THAT COULD INFLUENCE STUDENTS TO CREATE Viable MENTAL MODELS OF PROGRAMMING CONCEPTS

Studies of novice programmers’ mental models have shown that many of them are not able to accept the knowledge in a way that is expected of them, and one of the reasons may be that particular student’s prior knowledge of a particular concept, regardless of whether it is true or false belief, really affects him while building new knowledge. Existing students’ beliefs are often at odds with scientific
beliefs. While teaching students a scientific belief he can continue to use his existing unscientific understanding to interpret new information provided by the teacher and he would likely provide new meaning to information different or in conflict with scientific information. It is possible that a student doesn’t notice that his understanding is inconsistent with those displayed by the teacher. On the other hand, teachers aren’t often aware of the inappropriate misunderstanding that occurred with students (Nussbaum & Novick, 1982). This is especially true in the area of programming where, because of the invisibility of program execution, beginners in programming often mistakenly use their prior knowledge or intuitive models to understand the concepts of programming. When a student is satisfied with the previously adopted concept he doesn’t want to accept new concepts and it would be desirable to bring him into a situation to be dissatisfied with his existing knowledge or we could say we should bring the student out of his comfort zone.

One of the fundamental teaching strategies based on constructivism is precisely the strategy of cognitive conflict. It was basically developed on the assumption that students’ prior knowledge and current beliefs affect the way students learn new knowledge and create new beliefs. We bring student is a state of cognitive conflict, a condition in which the student observes the discrepancy between his cognitive structure and the external environment and between components of his cognitive structures (Lee & Kwon, 2001). The cognitive conflict strategy provokes learner's existing ideas in a way that encourages him to identify a problem in his understanding and motivates him to create a proper understanding (Scott, et al, 1992). In general, cognitive conflict strategy includes: research of student's prior knowledge and existing beliefs, challenging students with contradictory information and evaluation of conceptual changes between the student's previous and new ideas and beliefs (Limon, 2001). Analysis of the use of cognitive conflict strategy in working with novice programmers has shown promising results [26] [7] [9]. The problem of this approach is that some students do not manage to realize that they have a problem with their existing knowledge, they are not disgruntled, therefore they will not gladly accept new knowledge or modify the existing one. The strategy of cognitive conflict alone could not be sufficient to achieve change from unsustainable model in sustainable so student needs strong support. Good results were demonstrated in the application of cognitive conflict strategy with program visualization through the application of a visually oriented environment [26].

In an effort to assist their students in adopting demanding concepts of programming, teachers tend to use a variety of techniques, but also create new ones. Hertz and Jump presented such techniques called Program memory traces [27]. This technique represents a new approach to tracking code through lectures which model everything happening in memory while program is executing using traces. Tracking code is the key learning activity of this technique. The basic idea is to surround students with viable memory models and putting the focus of teaching and learning on tracing. Application of this method recommended certain pedagogical changes of the existing course like adding active learning exercises and demonstrating characteristics of working with finished sample code or coding live. The learning activities always involved problems that required tracing the finished code, or tracing code after making your own code examples. More than half of each lecture was organized as work on learning activities in small groups. Such teaching techniques based on code tracing showed very good results. Students pointed out that teacher's code tracing demonstration as well as their own tracing as the greatest help for their learning. The application of this method showed statistically significant improvement through improved students’ grades, improved students’ programming skills, reduced dropout and failure rates [27]

Another very interesting and lately often used method in teaching programming is pair programming. Pair programming is a technique which two programmers work together on a single keyboard, showed significant positive results, especially in the field of the student's sense of satisfaction and self-esteem [28, 29]. Given that such strategy has already shown the power of improving the quality of student work and increasing retention of students in programming course [30], also as the increase of student self-confidence [31] the question is if the way of assigning students in pairs affects their success. Recent results showed no statistical significance but the results have indicated that it would be more desirable to pair two students both with inconsistent models than pairing students with consistent and inconsistent mental model of the concept being taught to [31, 32].

Following the idea of CS popularization especially programming with K12 learners approaches like Model based thinking and practice is proposed. Such approach focuses on linking mental models (of real or imaginary world) and computer models of built-in computer systems. Model based thinking and practice advocate broadening of computer science issues and proposes an emphasis on the use of models and modeling, as well as skills that are considered to be generally useful, such as reading,
writing and math skills. Such approach could better clarify the role of models in computer science in relation to other areas and is believed to be able to strongly broaden the participation in computing, as well as to allow more students to become active creators through programming [33]. Computer models are a powerful tool for the creation of new processes and products and directly support thinking and practice-based model because the model becomes clear, tangible and interactive through some modeling environment like Netlogo.

Most of these strategies and techniques emphasize the achievement of better results by using a graphical tool or tool of program visualization. In selecting the appropriate teaching method it is good to be aware of learning styles that are more prevalent among students. The influence of learning styles on the success in the introductory programming courses so far has not shown the strength of success predictor, but some studies suggest that there are methods more suitable for some learning styles, and less for some other. The usage of flow-model visual programming language with method that encourages communication about program solutions showed promising results with possibility of addressing even difficult topics for novices especially those who prefer visual thinking [34].

4 APPLICATION OF PROGRAM VISUALIZATION FOR CREATING VIABLE MENTAL MODELS

Ben-Ari [4] emphasizes that program visualization has the potential to create learning environment that is comfortable and convenient. Visualization techniques are used for over 20 years and still have not proved successful and effective as expected. In recent years, all the more highlights that the student's interaction in the learning process is an important factor in effective use of visualization. For this reason, there are more and more studies of the program visualization application together with some pedagogical approach in order to create learning environments that could support students in creating viable mental models, but also to correct unviable models. Promising results were obtained by application of the model based on the strategy of cognitive conflict with the use of Jeliot, visually oriented environment, that helped students understand that there was a problem in their current understanding, and support them in correcting their nonviable mental models [26, 7].

Another interesting approach that developed and investigated Sorva [35] called Visual program simulation is a pedagogical technique for introductory programming education where student uses a visualization of an abstract computer. That visualization, ‘a notional machine’, is an aid to illustrate what computer does as it processes the program. The main goal of this technique is to help students to learn about programming in general and about specific programming concepts and for that purpose software system Uuhistle was used. Early investigation of suggested approach announced that Visual program simulation is a promising technique but also needs to be used thoughtfully. Technique was helpful in teaching about expression evaluation and function call, but not about reference semantics. Findings suggest that effectiveness of this technique depends greatly on what students have to reason about as they choose simulation steps [35].

In an effort to assist novice programmers in understanding programming and its dynamics there appear to be new software-systems offering improved design and implementation of notation machine. Application of Novis, a real-time dynamic visualizer notation machine integrated in BlueJ, offers a visualization that creates automatic and animated versions of notation machine diagram. It is integrated in BlueJ main interface, but also it represents a static notation machine diagram in the selected phase implementation of the program, or animates currently executing in real time. Novis system offers great simplicity in the presentation and the possibility to choose the level of clover display which is extremely important because this system could be adapted according the student's current knowledge or even for the use in subsequent courses of programming with more complex concepts and a higher level of clover. The system has been tested on a small number of students with promising results, but there is no formal studies that addressed this [24, 25].

Another tool designed for helping students in the active construction of mental models of computation and the evaluation of its use in the case of introductory programming is OTS. OTS is extensible and programming language neutral tool. Exercises in the tool appear as XML files rendered by the tool into the used language. Tracing program allows the student to step through each statement that is executed. Preliminary reports stated that the use of the tool led to great increase in completion of the course and that students believed that the tool really helped in understanding programming. Although students didn’t express great enjoyment with the use of tool, they agreed that it helped them understand program execution and build a mental model of computing [36].
Although there is no scientific verification, it is important to highlight, as an interesting pedagogical practice, developing free online learning materials that are open to teaching community, easy to approach for students of different ages. Such materials facilitate and make more accessible teaching and learning programming. An example of this approach is seen at Khan Academy Computing [37], which developed an interesting free online environment for “Talkthrough” learning programming through JavaScript programming language, using the method of flipped classroom, online video tutorials, discussion of task solutions, interactive materials and interactive programming environment (Fig. 2).

```
1. // Wuton's handsome features don't need outlines
2. shape4();
3. var myColor = '#000;
4. var x =
5. // face
6. fill(255, 255, 0);
7. ellipse(x, y, 80, 80);
8. ellipse(x+40, 80, 40);
9. ellipse(x+120, 120, 60, 60);
10. ellipse(x+160, 160, 40, 40);
11. // eyes
12. fill(164, 96, 65);
13. ellipse(x+120, 120, 60, 60);
14. ellipse(x+160, 160, 40, 40);
15. // mouth
16. fill(255, 60, 60);
17. ellipse(x+120, 120, 60, 60);
18. ellipse(x+160, 160, 40, 40);
```

Figure 2: Khan Academy Computing – More on variables

While there are a lot of researches that demonstrate the successful use of graphical environments in introductory programming courses it is interesting to examine what really students adopt faster and easier while using graphical environments, and what advantages the work in Command-Line environments has, if any. Evidently, standard Command-Line environment features are not exactly helpful for novice to learn programming as he must learn the syntax and language semantics while adopting file system management tools and compiling. In contrast, graphical environments allow novices to do the same with almost one mouse click. Research of learning programming with Python programming language through IDLE or VIM environment has highlighted the interesting and perhaps surprising results. Although they are simple for usage, consistency and affordance of graphical environments may force novices to develop wrong programming experience. Novices could impose inability to adapt to an environment which is less user-available. In contrast, the Command-Line environments, although challenging and perhaps even unattractive to novices, allow them to develop better mental models of programming and because of its limited features even easier transfer to a different environment. The differences in the characteristics of environments could affect the student’s learning curve. Graphical environments impose a lower learning curve while Command-Line environments require more time from beginners to get them adopted [38].

5 PEDAGOGICAL IMPLICATIONS

Although we still know very little about how the inside of our head works but one thing is certain. Students shouldn’t be allowed to rush to more interesting parts of introductory programming course until they fully adopt the basic idea that the computer is a mechanism [10]. While preparing lessons and teaching materials, teacher must put special attention on the simplest concepts such as assignment statement and sequence of instructions [8, 16]. Investing extra time in teaching concepts such as assignment statement is a good recommendation since that concept belongs to one of those causing the first bigger obstacle in learning programming [10]. On the other hand, holding on to some fundamental conception, at the beginning of the course, will certainly demand from teachers to modify their course curriculum what we could all agree is not so easy task for them. The teacher must decide whether to sacrifice some complex concepts which fight of opportunity for deeper study of simple concepts that are not so easy for students.

No matter which programming languages are taught, building good mental models can be achieved by teaching strategy that engages students in experiential learning tasks which includes monitoring of program flow. Task such as debugging and program modification often includes tracking and would likely contribute to the development of mental models. Generally tasks like active involving in
understanding and creating programs strengthen the mental models through thinking about the consequences like adding a new module program which includes interaction with other parts of the program [14]. Since promising result are achieved in application of different tools of program visualization teachers should find a way to fit them well in their teaching methods.

6 CONCLUSION

Analysis of results achieved by novice programmers after introductory programming courses showed that many novices have trouble in adopting basic programming concepts such as assignment statement and that it affects their motivation to learn programming, but often also causes the abandonment of these courses. Although there are slightly less studies just of programming novice's mental models, recently there are more and more experiments of new ways for detecting and recognizing students' mental models especially of basic programming concepts such as assignment statement, loop and recursion. Existing measuring instruments indicate the richness and diversity of understandings that develop students develop and they are able to identify their valid mental models but we can’t say that they have the power to recognize the “real talent” for programming. Several studies have shown that previous experience of programming couldn’t predict success in programming, but positive previous programming experience affects the creation of valid mental model and student’s self- efficacy. Teaching methods that emphasize student's involvement put an accent on monitoring and understanding of the programming code and along with some tools for program visualization have proved to be successful in developing viable mental models. For teacher it is extremely important to know mental models that develop their students to be able to adopt his teaching method, as well as existing teaching materials. Further research in the area of mental models and measuring instruments for their recognition is certainly an important part of computer science education research.

REFERENCES

[8] Simon, »Assignment and sequence: why some students can’t recognise a simple swap,« u Koli Calling ’11, Koli, Finland, November 17–20, 2011.


V. Ramalingam, D. LaBelle i S. Wiedenbeck, »Self-Efficacy and Mental Models in Learning to Program,« u ITiCSE'04, Leeds, United Kingdom, June 28-30, 2004.


B. Adelson, »When novices surpass experts: The difficulty of a task may increase with the expertise,« Journal of Experimental Psychology: Learning, svez. 10, br. 3, 1984.


M. Berry i M. Kölling, »The design and implementation of a notional machine for teaching introductory programming,« u WiPSCE '13, Aarhus, Denmark, November 11–13, 2013.


M. Hertz and M. Jump, "Trace-Based Teaching in Early Programming Courses," in SIGCSE’13, Denver, Colorado, USA, March 6–9, 2013.

Corney, M.; Teague, D.; Thomas, R. N., »Engaging Students in Programming,« u 12th Australian Computing Education Conference (ACE2010), Brisbane, Australia, 2010.


[33] P. Nowack i M. E. Caspersen, »Model-Based Thinking and Practice: A Top-down Approach to Computational Thinking,« u Koli Calling ’14, Koli, Finland, November 20 - 23 2014.


