New Approaches and Tools in Teaching Programming

Danijel Radošević, Tihomir Orehovački, Alen Lovrenčić
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{danijel.radosevic, tihomir.orehovacki, alen.lovrencic}@foi.hr

Abstract. *Teaching programming at university beginner's level has some specific problems like wide diversity of student's previous knowledge, fear of programming, problems with programming language syntax etc., as shown in many previous researches and our on-line questionnaire. There are already some approaches and tools developed to make programming concepts easier to understand for students, like different visual tools, tutorials, video lectures and even new programming languages, developed to learn programming concepts. Our approach is based on development of specific learning interface to standard programming languages, like C++, instead of standard IDE-s. That interface should prevent many mistakes students often do in learning programming and make easier for teachers to help their students before they collect to many syntax and logical errors.*

Keywords. programming, teaching software, teaching approaches

1 Introduction

This paper analyzes specificities and issues of teaching programming at university beginner's level and suggests our solutions in organization of teaching process and also the software tool that supports that process and helps students in easier achievement of needed programming knowledge and skills.

During last several years, we periodically conduct web questionnaire on a pattern of students on Faculty of Organization and Informatics (1. year of study, at Programming 1 course). Questions are about student's previous programming and general informatics knowledge and what they find as a problem in their understanding of programming. Their answers helped us to find out their difficulties and problems in learning and our biggest issues in teaching too. Firstly, it's obvious that teaching programming is faced to big variety of student's previous knowledge, together with their different attitudes toward programming. That is the big issue and one of the main specificities in teaching programming, toward teaching some other disciplines where the previous knowledge level doesn't vary in such degree. Furthermore, some students feel fear of programming as something that is "very hard". Some students starts with "it's easy" approach, but, when exercises become harder, change it into "it's too hard for me". We usually compare that situation with the fitness room when new trainees arrive. They are often very enthusiastic about their training, but sometimes don't consider warnings about their practice seriously (e.g. start with too heavy weights). Consequences, like pain, injuries and even hernia are possible and have their mental analogies when students try to achieve programming skills. There different approaches to deal with that problem, and we referenced some of them in our Related work chapter.

Our approach is based on development of specialized learning interface to standard programming languages, like C++, instead of standard IDE-s. That learning interface should prevent students from some bad habits in programming, like writing code without syntax and logical checking and learning program code by rote. There also advantages for teachers and teaching process like:

- It's much easier now to help students before they swamp into errors, and
- It's easier to prevent students from unallowable acts like copying programs (programs have to be written during exercises)

There are some other possibilities for students like different code analyses (e.g. usage of curly brackets and program structures) and debugging. After testing period, we plan to continue with development of our learning interface, by adding some new features like time measuring, testing theoretical knowledge and uploading code on the Internet.

2 Related work

Programming is one of the basic subjects of information science curriculum but for most students is also one of the most difficult. For several years, a significant number of students had problems with the successful passing the examinations of programming and related subjects and they are often the reason for cancellation of the study or transferring to another course. It is an unpopular trend that is not represented only at our faculty but also at other universities worldwide where similar studies are taught. Thus the results of research conducted at Monash University showed that first-year students considered programming as the most difficult and least interesting subject in all Computing courses [13]. Reasons for this may be diverse. Firstly, programming cannot be learned only from books as is the case in some other subjects; in order to learn the basic concepts of programming and develop the algorithmic approach in problem solving process, students have to invest a lot of time and practical work. However, the basic problem with the learning of programming lies not in overcoming programming concepts but in their implementation [18]. Furthermore, in order to solve the well-defined programming tasks and problems, students must have the appropriate level of mathematical knowledge [5][11] and the logical and abstract thinking which is often not the case primarily because the students who enroll Informatics course mutually differ by which secondary schools they come from. Finally, students lack of motivation to learn programming, because they have the image of the programming as something that is very difficult even before classes begin. The main reason for these arises from the fact that senior students who had problems with programming are spreading negative connotations toward novice students.

In the past few years, a large number of tools and methodologies are developed in order to support students when learning programming. Some scientists believe that the problem solution lies in choosing the most appropriate programming language, methodology and tool for teaching and content that will be taught [10] [21]. So at the top of list of the most popular languages for teaching programming are the main representatives of object-oriented paradigm: Java and C + + [30] while some authors as an alternative propose multimedia language Actionscript [8]. Reason why C + + is so frequent in use arises from the fact that it is a general purpose language that contains exactly all the elements that are necessary for beginners to understand basic programming concepts like control structures, mechanisms of aggregation, etc. In addition, after the syntax of programming language C + + is learned by students, it would be much easier for them to master other modern languages with C-like syntax such as Java, PHP, JavaScript, etc. [20].

After a suitable programming language for teaching programming has been selected, it is necessary to choose the appropriate teaching methodology and tool that will facilitate students understanding of basic programming concepts. Choosing the most appropriate teaching method in the field of information and computer science was the subject of research of large number of scientists [17][21][31] but for our research the most interesting was tool oriented method. According to the characteristics of that method (for details see [28]), there are two main groups of tools: mini-languages [4] and visualization tools [17].

The basic idea of mini-languages is that the student controls some actor in microworld and thus learns programming concepts like control structures, functions, recursion, etc. The development of mini-languages was significantly inspired by programming language Logo [27]. But Logo itself is not considered to be representative of generations of mini-languages mainly because the actor (turtle) does not have interaction with its microworld and does not support basic control structures like if and while. However, the basic set of commands by which the student controls the actor in microworld and thus solves given problem was taken from Logo. The most important representative of a mini-language group is Karel the Robot [29] in which student controls actor (robot) with four main actions and through interaction of actor and its' microworld learns basic control structures. However, Karel the Robot has a limitation in the sense that it does not support variables, types and expressions. Nevertheless, Karel Genie [22], an integrated software environment for the original Karel has been used, as teaching programming tool, for many years in high schools and prestigious universities across the U.S. Development of mini-languages Karel the Robot, initiated the development of tools for similar purposes, such as Josef the Robot [32], Martino [26], Marta [6], Turingal [3], Darel [16], Karel-3D [15] and Guido von Robot [33] (a Python version of Karel the Robot; see Figure 1) but none of them did not achieve success as the original.

The second group consists of visualization tools that are a combination of multimedia elements which main purposes are to help students in understanding the basic concepts of programming, facilitate the development of software applications, and motivate them for the process of learning programming. This group includes two types of tools: demonstration tools and virtual worlds.

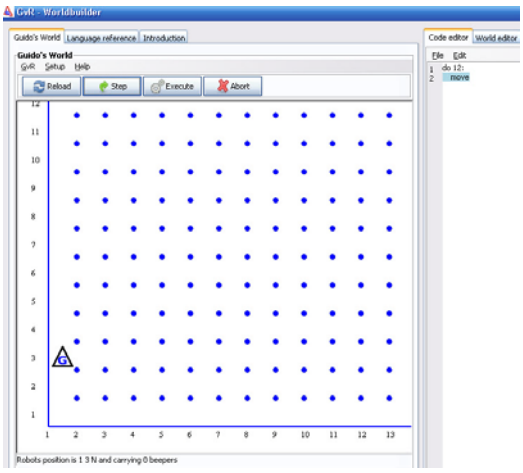

Figure 1. Guido von Robot

Demonstration system for teaching purpose facilitates learning in the way that it divides the course material into sequence of smaller logical entities (learning objects) that are easier to understand. Learning object is small, substantive and reusable media resource which contains high quality information and it's used during technology supported learning [1][25].

The main representatives of this group of tools are AnnAnn and AnnAnn.NET that enable iterative and incremental program development. Namely, for learning new programming concepts, the teacher starts with a known program segment (e.g. declaration of variables) and with sequential changes in the code (e.g. the introduction of control structure or initialization of array) creates a completely new program that addresses the pre-set a problematic task [14]. OOP-Anim [9] works in a similar way but unlike as AnnAnn and AnnAnn.NET that are intended for teaching any type of programming languages, it is specialized for learning object-oriented concepts.

The main representative of virtual worlds is Alice [7], 3D programming environment that through the creation of simple animation or video games teaches students the basic programming constructs. Using the interactive interface, students drag and drop 3D objects into the virtual world and thus create a series of instructions that is program. What is most interesting is that each instruction in Alice is equivalent to the statement of the most popular object-oriented programming languages such as Java and C + +. Therefore, students can very easily, during the development and testing of their programs, identify a correlation between the behavior of the object in the animation and certain program statement and thus learn the basic programming concepts. Besides Alice (see Figure 2), this group includes also LogoBlocks [2] which uses graphical objects labeled with words and Obliq 3D [23] which is more for experts than novice students beginners.

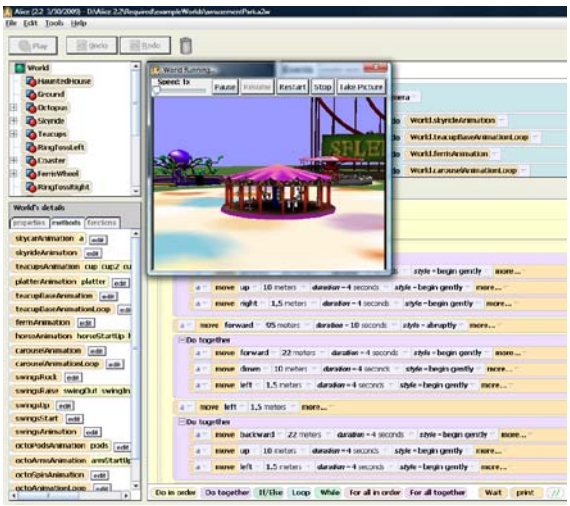

Figure 2. Alice

Mention teaching concepts and tools have a positive impact on students learning [19][24], but still, the main problems with which students are facing when learning programming are still remaining unsolved. Due to these reasons, we decided to develop our own tool for teaching programming.

3 What and how we teach computer programming

There are many questions regarding teaching of computer programming pointing teaching methods as well as technology used.

Although the good programming course should be independent on programming language used in it, the good praxis to teach concepts of computer programming in relation to some programming language. So, the choice of programming language is very important, if not crucial for the programming course organization. As it can be seen from [34], the programming languages used today converging to C-like programming languages. More than 50% of programming code used in USA today is written in the one of the languages very close to C programming language – C, C++, C# and Java. The only two other significantly used programming languages are PHP with almost 10% and Visual basic with 8.5% of used code. There are some parameters we used in programming language choice:

- Programming language usage
- Popular OS platform support
- Availability of all important programming concepts.

The first parameter in the list clearly points to one of the C-like languages. Although Java is the most popular programming language today, and it is platform independent, the lack of some important

concepts in the language, such as pointers, makes it questionable as a choice. It would not be so big problem for the programming course for the students which main field of interest in study is not computer software design and development. However, programming course for computer and information science students have to introduce all of main programming concepts, which includes pointers and dynamic memory allocation mechanisms.

C programming language can be, and in the many cases is used for the fundamental programming course, after which some other similar languages like C++ or Java is used for advanced courses. Our opinion is that it is not necessary to start with C programming language and that other C-like languages that provide higher programming level are more convenient for beginners in computer programming. That is the reason that we decided to use only one programming language for all levels of programming courses.

C# is the newest of four most popular C-like programming languages. It provides support for all main programming concepts, including some concepts very interesting for teaching, such as safe pointers, possibility of manual or automatic memory reallocation, and so on. The drawbacks of C# are relatively small percentage of use (about 4.3%) and weak Linux platform support.

So, the language we chose is C++, which is widely supported in all platforms, provides support for all programming concepts (although its concept of unsafe pointers is not particularly good for beginners).

The second important matter that has to be considered is methods of teaching that should be used in programming course. Although most of common teaching methods work, the matter teacher deals with in programming course are in many points very specific. In programming, like in mathematics, it is very important that student follows the schedule of the course very closely, because teaching units In the most of cases hardly depend on previous ones. Because of that continuous learning is one of the greatest goals in programming teaching. On the other hand, skills students achieve shall be improved only by extensive work in programming. The way to achieve that we chose is the greater number of blitz-exams and laboratory exercises that are graded. This two sorts of exams insure that a student learns both practical skills and theoretical knowledge continuously.

The goal of blitz-exams is to drive students into continuous learning of theoretical knowledge and concepts of programming, while laboratory exercises sharpen their programming skills. To support this two ways of learning and examining, we introduced the 3-level teaching process:

- Lectures
- Auditory exercises and
- Sillabuses

In lectures the concepts are introduced, mostly using generalization and abstraction methods. We found that these two methods that provides bottom-up approach to teaching greatly overheads the classical, formal, top-down approach. The reason for that is that this approach provides more usable examples and connects every introduced concept to some of them, which ensures that student immediately see the purpose of the concept.

The auditory exercise provides technical knowledge and skills needed for syllabus; introduce top-down approach to computer program development from idea to the algorithm and computer program.

At the last, syllabus is reserved for student work. The problem in this part of teaching process is that as course advance the time needed for development of algorithm and program rises as well. That is the reason why it is necessary to publish problems that students have to solve in syllabus in advance. But, that is also the reason why the plagiarism has to be considered as a possibility. The system we developed and that is introduced in the rest of the paper is one of the results of our plagiarism concern.

4 Students' questionnaire

Our students' web-form questionnaire has been conducted in 2009. on a population of 182 examinees (students at the University of Zagreb, Faculty of Organization and Informatics, 2nd semester, at Programming I course). Similar questionnaire have been conducted in 2004 (206 examinees), so the results are compared.

Answers about students' previous programming experiences show that the number of students with previous programmers experience is slightly lowered in 2009. toward 2004. as shown in Table 1.

Table 1. Students' previous experience in programming

	2004.	**2009.**
programs up to 1000 lines	40%	37%
programs larger than 1000 lines	8%	4%

There is also a shift among the most popular programming languages, as shown in Table 2.

Table 2. The most popular programming languages among students

Interest lowered	**2004.**	**2009.**
Basic	32%	20%
Visual Basic	16%	8%
Pascal	27%	13%
Interest increased	**2004.**	**2009.**
C++	16%	32%
Java and C#	2%	9%
PHP and Perl	2%	9%

Furthermore, the questionnaire shows that students declared good basic informatics knowledge in both surveys, and there is now some previous knowledge of multimedia networking and computer security (Table 3).

Table 3. Elements of general informatics knowledge

	2004.	2009.
writing text	98%	96%
using Internet	96%	98%
using spreadsheet	71%	74%
multimedia tools	50%	62%
networking	unknown	48%
computer security	unknown	23%

It seems that students are well equipped by computers and link to the Internet (Table 4).

Table 4. Computer equipment and average usage of the Internet

Computer	**2004.**	**2009.**
have their own computer	90%	99%
share computer with somebody	7%	1%
borrow computer occasionally	3%	0%
own more than one computer	unknown	35%
Internet	**2004.**	**2009.**
more than once daily	23%	92%
once a day	18%	6%
several times weekly	34%	2%
once weekly	13%	0%
less than once weekly	9%	0%

4.1 Students' standpoints about programming

According to their wishes about future job, some students want to have programming as one of the major working tasks, and somebody would rather do something that not include programming (Table 5; results are for 2009.).

Table 5. Students' wishes about their future job

	% students (2009.)
want their job include programming	33%
if can't avoid	29%
don't want programming	25%
undecided	13%

About problems in learning programming, students extracted several (Table 6).

Table 6. Students' problems in learning programming

	% students (2009.)
lack of previous knowledge	65%
preoccupation by other duties on their study	58%
algorithm for solving problem	26%
using curly brackets	17%
fear of programming	**13%**
C++ syntax	12%
C++ seems too hard	**9%**
C++ operators	9%
iterations	9%
C++ instructions	7%
using semicolon	2%

Some other researches, e.g. Gomes and Mendes [12] also show that algorithm for solving problem is the harder problem than programming language syntax. But, that could lead to wrong conclusion that teaching process should be oriented to algorithms only, because programming syntax is "easy". The real question is "Why students have difficulties in passing from syntax level of thinking to algorithm level of thinking in programming?". Possible answer is that the problem lies in the connection between these two levels, as shown on Figure 3.

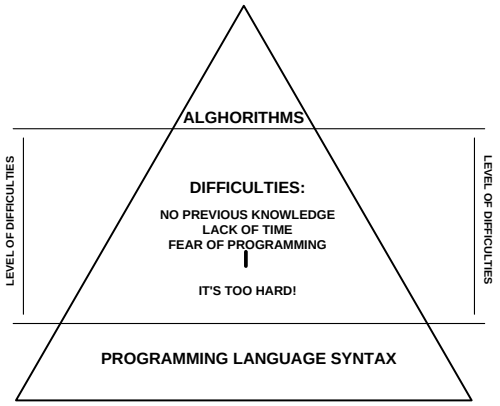

Figure 3. Difficulties in learning programming

So, the lack of programming language syntax knowledge, together with lack of problem solving abilities that many students show leave the gap for difficulties in programming. While the lack of previous knowledge could be the biggest problem when starting with programming, it's much harder problem when it outgrow to conclusion that programming is "too hard". In that situation the students may even give up from learning programming skills and from engaging in the activities defined in the syllabus for the course.

Our approach to reduce the level of difficulties includes using learning programming interface that prevent students to swamp into syntax problems together with constant work on the improvement of the teaching process.

5 Learning interface to standard programming languages

During years of teaching experience, we found some bad programming habits of our students. These start from trying to write program code by rote, without syntax and logical checking. After some time, when exercises become harder, such students are faced to fact that they don't really understand the programming code they wrote which can easily finish in a conclusion that programming is "too hard" for them. There were also attempts to copy program code from colleagues.

Our learning programming interface should prevent these bad programming habits and turn back our students on a "right way" in programming.

5.1 Control points in writing programs

The first idea of our learning interface is to push students to make "control points" in their programs. That means that each program starts from the easy one which has to be checked, so the development can continue toward next control point, until program has all the needed functionalities (Figure 4).

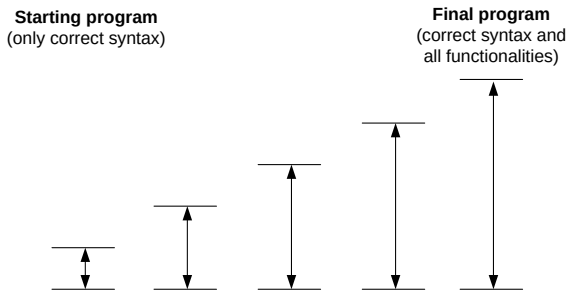

Figure 4. Program development from the easy one to fully functional final

The number of programming lines between control points has not to be too high, because of possible collecting errors. On the other hand, there is not always possible to compile program after adding one single programming line, because all program structures have to be closed (e.g. there have to be right curly bracket '**}**' for each left curly bracket '**{**').

5.1.1 Using semaphore in programming

Our learning interface introduces semaphore (in a form of "traffic light") which contains number of new programming lines after last compilation (Figure 5).

Semaphore works in a way that each new programming line and each semicolon are counted. Values 0-4 are inside green light, 5-10 inside yellow and values bigger than 10 inside red light. If the light is red, program can't be compiled and programmer has to reduce number of lines (or put something into comment - comment are not counted). When the program is compiled, the semaphore value returns to zero. Of course, it is not possible to load program into learning interface, or to copy it into editor.

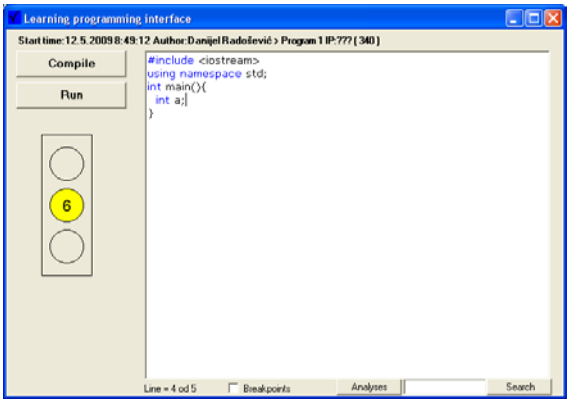

Figure 5. Learning programming interface with semaphore

5.2 Personalization of programs

The next idea of learning programming interface is to personalize the programs in a way that each program has it's owner. So, before starting with programming, students have to fill appropriate data form (Figure 6).

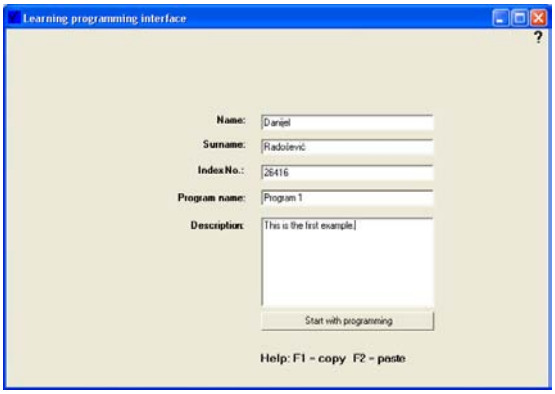

Figure 6. Program development from the easy one to fully functional final

The data entered in a form, together with some other program attributes, finish in program code in a form of comments, e.g.:

```
// MD5:xvdsRpxpRWC3jPjvyIfKyA==
// Learning programming interface
// Program:Program 1
// Description:This is the first example.
// Author:Danijel Radošević
// Start time:12.5.2009 9:05:44
// Final time:12.5.2009 9:06:13
// IP: ( 340 )
// #:#include<iostream>,
// Succesfull/unsuccesful compilings:1/0
      #include<iostream>
      using namespace std;
      int main(){
            int a;
}
```

The code are automatically saved on computer desktop, after compiling (if passed the compilation!) and contains the MD5 checksum as a guarantee that program is really written in educational programming interface (that can be checked).

5.3 Additional possibilities and helping tools

The learning programming interface has some additional possibilities to help students in programming:

- Marking error containing code in red light after compilation
- Code analyses:
  - Using of curly and round brackets
  - Open program structures on cursor position
  - Review of all program structures
  - Marking of unreferenced program structures (functions, methods, classes and structs)
- Using breakpoints in program

There also some restrictions in writing programs:

- It is not allowed to put more than five empty rows
- Each program line may contain only one semicolon (except the ***for*** iteration and ***break***)
- It's not allowed to go too deep into "red" - there is a warning when the semaphore value is 18 or bigger

6 Preliminary results

The learning programming interface has been used by our students at Programming I course for their laboratory exercises during one semester. Previously, they used standard programming interfaces like *Visual C++* and *DevC++*. Our first experiences are as follows:

- The passing rate at the end of semester has increased from 50,0% (151 out of 302) in 2008. to 70,2% (179 out of 255) in 2009.
- It was easier for teachers to control their students during exercises and exams (e.g. from copying programs)
- The most positive students' comments are from the repeaters (did not pass the exam in previous year): "If there were learning programming interface last year, I would never be a repeater!" - said one of them
- Negative students' comments:
  - 10 new lines before compiling is too few and slow down the programming (the most often negative comment)
  - Some usual possibilities are still not implemented (e.g. undo)
  - Some bugs reported (e.g. problems with editor and semaphore)

We noticed that students often make some typical syntax and logical mistakes (e.g. despaired curly brackets or usage of '=' operator instead of '==' in logical tests) and waste lot of time in trying to solve the problem. Moreover, compiler messages sometimes confuse them. Some of these situations could be detected automatically and reported to students, helping them to find the solution. That could compensate some of the time waste caused by frequent compiling.

7 Conclusion

Teaching programming at university beginner's level faces to some specific problems as shown in our student's questionnaire. The lack of previous knowledge, together with problems in understanding of program code and even fear of programming sometimes leads students to the "wrong way". Bad programmer's habits like writing code without syntax and logical checking and learning program code by rote cause considerable problems when exercises become harder and often lead students to conclusion that programming is "too hard" from them.

Our approach to deal with that problem include learning programming interface to standard compilers, which prevent students to copy programs mutually and force them to make control points during development of their programs. There also some possibilities which help students in analyzing program code and debugging. Each program written in learning programming interface is personalized by containing the data entered in a form on the beginning of programming session, together with some other data and MD5 checksum which guarantee that program is really written in learning programming interface.

There are advantages for students and advantages for teaching process from using learning programming interface. Students are pushed to check their programs during development process so they can't swamp into errors. Also, the tools for program analyses and debugging help them to find the cause of their errors. For teaching process, it's important that students can't copy their programs from outside. Also, teachers can help their students about their syntax and other errors before they collect too much of them.

In the future development of learning programming interface, we plan to introduce some new possibilities like better explanation of syntax and logical errors, time constraints (for exams) and questions about the program to check the student's understanding.

References

[1] Abernethy, K., Piegari, G., Reichgelt, H., Treu, K.: **An Implementation Model for a Learning Object Repository**, Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Ostober 24-28, Vancouver, Canada, 2005, pp. 2-7.

[2] Begel, A.: **LogoBlocks: A Graphical Programming Language for Interacting with the World**, Boston, MA, MIT, 1996.

[3] Brusilovsky, P. L.: **Turingal - the language for teaching the principles of programming**, Proceedings of Third European Logo Conference, 27-30 August, Parma, Italy, 1991, pp. 423-432.

[4] Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., Miller, P.: **Mini-languages: A Way to Learn Programming Principles**, Education and Information Technologies, Vol. 2, No. 1, pp. 65-83.

[5] Byrne, P., Lyons, G.: **The Effect of Student Attributes on Success in Programming**, Proceedings of 6th Conference. on Innovation and Technology in Computer Science Education, June 25-27, United Kingdom, 2001, pp. 49-52.

[6] Calabrese, E.: **Marta - the "Intelligent Turtle"**, Proceedings of Second European Logo Conference, 30 August - 1 September, Gent, Belgium, 1989, pp. 111-127.

[7] Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K.: **Alice: lessons learned from building a 3D system for novices,** Proceedings of the SIGCHI conference on Human factors in computing systems, April 01 – 06, The Hague, The Netherlands, 2000, pp. 486 – 493.

[8] Crawford, S., Boese, E.: **ActionScript: a gentle introduction to programming**, Journal of Computing Sciences in Colleges, Vol. 21, No. 3, pp. 156-168.

[9] Esteves, M., Mendes A.: **OOP-Anim, a System to Support Learning of Basic Object Oriented Programming Concepts**, Proceedings of the 4th International Conference Conference on Computer Systems and Technologies, June 19-20, Rousse, Bulgaria, 2003, pp 573 - 579.

[10] Giangrande, E.: **CS1 Programming Language Options**, Journal of Computing Sciences in Colleges, Vol. 22, No. 3, pp. 153-160.

[11] Gomes, A., Carmo, L., Bigotte, E., Mendes, A.J.: **Mathematics and programming problem solving**, Proceedings of the 3rd E-Learning Conference – Computer Science Education (CD-ROM), September 7-8, Coimbra, Portugal, 2006.

[12] Gomes, A., Mendes, A. J.: **Learning to program - difficulties and solutions**, Proceedings of the International Conference on Engineering Education, September 3-7, Coimbra, Portugal, 2007.

[13] Hagan, D., Sheard, J., Macdonald, I.: **Monitoring and evaluating a redesigned first year programming course**, ACM SIGCSE Bulletin, Vol. 29, No. 3, pp. 37-39.

[14] Hooper, C., Carr, L., Davis, H., Millard, D., White, S., Wills, G.: **AnnAnn and AnnAnn.Net : Tools for Teaching Programming**, Journal of Computers, Vol. 2, No. 5, pp. 9-16.

[15] Hvorecky, J.: **Karel the Robot for PC**, Proceedings of East-West Conference on Emerging Computer Technologies in Education, 6-9 April, Moscow, Russia, 1992, pp. 157-160.

[16] Kay, J., Tyler, P.: **A microworld for developing learning design strategies**, Computer Science Education, Vol. 3**, No.** 1, pp. 111-122.

[17] Kelleher, C., Pausch, R.: **Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers**, ACM Computing Surveys, Vol. 37, No. 2, pp. 83 – 137.

[18] Lahtinen E., Ala-Mutka K., Jävinen, H.: **A study of the difficulties of novice programmers**, Proceeding of the 10th annual SIGCSE conference on Innovation and technology in computer science education, June 27 – 29, Caparica, Portugal, pp. 14-18.

[19] Lawrence, A., Badre, A., Stasko, J.: **Empirically Evaluating the Use of Animations to Teach Algorithms**, Proceedings of the IEEE Symposium on Visual Languages,October 4-7, St. Louis, U.S.A., 1994, pp. 48-54.

[20] Lovrenčić, A., Konecki, M., Orehovački, T.: **1957 – 2007: 50 Years of Higher Order Programming Languages**, Journal of Information and Organizational Sciences, Vol. 33, No. 1.

[21] Matthíasdóttir, Á.: **How to teach programming languages to novice students? Lecturing or not?**, Proceedings of the International Conference on Computer Systems and Technologies, June 15-16, University of Veliko Tarnovo, 2006, Bulgaria.

[22] Miller, P., Pane, J., Meter, G., Vorthmann, S. R.: **Evolution of novice programming environments: the structure editors of Carnegie Mellon University**, Interactive Learning Environments, Vol. 4, No. 2, pp. 140-158.

[23] Najork, M.: **Obiq-3D Tutorial and Reference Manual**, DEC SRC Research Report #129, 1994.

[24] Nevalainen, S., Sajaniemi, J.: **An Experiment on Short-term Effects of Animated versus Static Visualization of Operation on Program Perception**, Proceedings of the International Workshop on Computing Education Research, September 9-10, Canterbury, United Kingdom, 2006, pp7-16.

[25] Nugent G., Soh L., Samal A., Person S., Lang J.: **Design, Development, and Validation of a Learning Object for CS1**, Proceedings of the

10th annual SIGCSE conference on Innovation and technology in computer science education, June 27 – 29, Caparica, Portugal, p. 370.

[26] Olimpo, G., Persico, D., Sarti, L., Tavella, M.: **An experiment in introducing the basic concepts of informatics**, Proceedings of Fourth World Conference on Computers in Education, 29 July - 2 August, Norfolk, U.S.A., 1985, pp. 31-38.

[27] Papert, S.: **Mindstorms: children, computers, and powerful ideas,** Basic Books, 1980.

[28] Papp-Varga, Zs., Szlávi, P., Zsakó, L.: **ICT teaching methods – Programming languages**, Annales Mathematicae et Informaticae, Vol. 35, pp. 163 – 172.

[29] Pattis, R.E.: **Karel the Robot: A Gentle Introduction to the Art of Programming**, John Wiley & Sons, 1981.

[30] Schulte, C., Bennedsen, J.: **What do teachers teach in introductory programming?**, Proceedings of the 2006 international workshop on Computing education research, September 9-10, Canterbury, United Kingdom, 2006, pp. 17-28.

[31] Szlávi P., Zsakó, L.: **Methods of teaching programming**, Teaching Mathematics and Computer Science, Vol. 1, No. 2, pp. 247–258.

[32] Tomek, I.: **Josef, the robot**, Computers and Education, Vol. 6, No. 3, pp. 287-293.

[33] ***, Guido von Robot, URL: http://gvr.sourceforge.net/, Retrieved: April 30, 2009.

[34] *** *TIOBE Programming Community Index for May 2009*, TIOBE Software