

Evaluating objectKarel - an educational programming environment for object oriented programming

S. Xinogalos*,¹, M. Satratzemi¹, and V. Dagdilelis²

¹ Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

² Department of Educational and Social Policy, University of Macedonia, Thessaloniki, Greece

A common experience of those who teach introductory programming courses is that learning to program is difficult. So, special programming languages and educational environments have been devised with the aim of making the learning process easier. Unfortunately, in many cases the true value of these methods was not evaluated. Based on previous research findings and our experience in teaching object-oriented programming (OOP), we developed an environment called objectKarel. objectKarel is being used for three years in our University and this paper presents a systematic analysis of its didactic effectiveness based on evaluations from students themselves. Besides the evaluation of the effectiveness of the environment, this research allowed us to state some more general proposals regarding the design of programming languages and educational environments for an introduction to programming.

Keywords educational programming environments; mini-languages; microworlds; object-oriented programming; design principles; evaluation

1. Introduction

Computer programming is being taught in various levels of education for four decades. A common phenomenon in these teachings is that learning to program is difficult. In order to face this problem, a series of teaching methodologies, special programming languages and educational environments that take advantage of Information and Communication Technologies have been devised with the aim of making the learning process easier. Unfortunately, in many cases the true value of these solutions was not evaluated. Significantly big was the number of special languages and educational environments developed especially for supporting novice programmers, without being always accompanied with an objective evaluation of their didactic effectiveness [1].

Based on previous research findings and our experience in teaching object-oriented programming (OOP) to novices, we designed and implemented an educational programming environment, called objectKarel [2], based on a special object-oriented mini-language. The environment is based on the previous microworlds of Karel and Karel++ [3], which simulate the movement and action of one or more robots in a rectangular region (of the screen). The environment of objectKarel, besides the OOP language, incorporates many features that have proven to be important in analogous environments, such as: (i) a structure editor for supporting students in developing programs and avoid focusing on the syntactic details of the programming language [4]; (ii) the ability of executing programs step-by-step for supporting students in understanding the semantics of various programming structures and flow of control. Besides this, however, objectKarel is characterized by some novel features too, such as: (i) a series of lessons and hands-on activities incorporated in the environment with the aim of supporting students in comprehending OOP concepts prior to using them for implementing programs; (ii) the ability of explanatory visualization [5], that is the presentation of messages in natural language explaining the semantics of the statement being executed.

objectKarel is being used for three years in our University and this paper presents a systematic analysis of the didactic effectiveness of the environment – according to the evaluations of the students themselves. The related data were collected from different audiences, in different time periods. The data were

* Corresponding author: e-mail: stelios@uom.gr, Phone: +30 2310950613

analyzed qualitatively and quantitatively and give a complete picture of the effectiveness of the language and the environment of objectKarel. Besides the evaluation of the effectiveness of the Karel++ language and the environment in general, this research allowed us to state some more general proposals and principles regarding the design of programming languages and educational environments for an introduction to programming.

2. Empirical data for the effectiveness of objectKarel

The evaluation was based, mainly, on questionnaires given to 2 groups of students after a series of 5 two-hour lessons, which were carried out at a department of Applied Informatics. The data used for evaluation were - mainly - collected from questionnaires with open and closed type questions, but data collected from studying the consecutive versions of students' programs were used too.

The two groups of students have several similarities. The two groups consist of 19 and 24 volunteer students that attended the 5 lessons, carried out by the same teacher and under the supervision of the same research group. The lessons given in the two groups were not identical, since our experience from the first series of lessons showed that it was necessary to make some changes, but in general similar. Although, all the students faced difficulties with programming (most of them had failed the exams of programming courses), there was a significant difference between the two groups: the students in the second group were studying in higher semesters (4th to 8th) by those in the first group (2nd-3rd) and were more experienced in programming paradigms, languages and environments. What is important to state about an evaluation of this kind, is the fact that the effectiveness of the language and the environment is not independent of the users' profile. For example, experienced users have different demands from the system, and at the same time they are more exacting reviewers: they observe and criticize elements that are overlooked by novices. However, the opposite is true too, in some way: experienced users have established habits that cannot be easily shifted: for example, it is difficult for them to accept a common symbol (i.e. a specific form of the cursor) with a different meaning/functionality than that they are used to. The data collected, do not show important differences between the evaluations of the two groups. The most experienced users make more and more precise observations, but without expressing doubts regarding the general usefulness of the language and the environment.

Usability evaluation. The usability evaluation of the environment is positive, since, students rated the system - according to the usability heuristics of Nielsen [6] -with 4,4 (1st group) and 4,5 (2nd group) in the range of 1 to 5 (5=excellent). The usability heuristics that students were asked to rate, their ratings and the problems mentioned by the 2nd group (of the more experienced users) are all summarized in Table 1. Moreover, in a question regarding the advantages of the environment, 50% of the students of the 1st group and 42% of the 2nd group explicitly stated that one of the system's advantages is its usability.

Table 1 Usability evaluation based on the usability heuristics of Nielsen

Usability heuristic	Group1	Group 2	Problems (Group 2)
Provide feedback	4,3	4,6	Lack of "undo" –
Simple and natural dialogue	4,6	4,8	works only for the last
Good error messages	4,5	4,3	action.
Minimize user memory load	4,2	4,5	Lack of "copy",
Provide clearly marked exits	4,1	3,5	"paste".
Be consistent	4,4	4,8	Lack of the ability to
Prevent errors	4,6	4,5	"delete" arbitrary
Provide shortcuts	4,8	4,9	number of lines.

Of course, the usability heuristics have a more general character, since they refer to the interface and the functionality of the environment in general. So, the majority of our questions explored students' evaluation in relation to the effectiveness of the environment as a *didactic tool*. In this context, we present students' evaluations regarding the special characteristics of the language and the environment.

e-lessons and hands-on activities evaluation. Our hypotheses regarding the importance of incorporating theory and hands-on activities in the environment were verified. This feature helped students not only during the lessons, as could be expected, but it helped them during problem solving. In a relevant question the vast majority of students answered that went back to the theory and hands-on activities module of the environment during program development, as shown in Table 2.

Table 2 Replies to the question “did you go back to the theory/activities during program development?”

	Group1	Group 2
Yes	79%	71%
No	21%	29%

Structure editor evaluation. Both the development and the correction of programs with the use of the structure editor are considered to be easy by the vast majority of students. The exact percentages of students are presented in Table 3.

Table 3 Replies to the question “was the development and correction of programs with the structure editor easy?”

	Easy development		Easy editing	
	Group 1	Group 2	Group 1	Group 2
Yes	100%	83%	95%	75%
No		17%	5%	25%

In a question where students were asked to make proposals for the improvement of the structure editor, the 2nd group, which consists of more experienced students, made proposals that in fact show what the drawbacks of a structure editor are. As a matter of fact, these problems are those referred in the question regarding usability evaluation (and presented in Table 1). Students’ answers are summarized in Table 4. In this Table and those that follow the percentage of students that included a specific element in his/her answer is presented, while the percentage is calculated accordingly to those students that answered the question.

Table 4 Replies to the Question “Make proposals for the improvement of the structure editor”

	Group1	Group 2
Combination of a structure – text editor	17%	27%
Include “Cut”, “Copy”, “Paste”, “Undo”		27%
Easier correction of programs (editing)		18%
Presentation of the structure editor’s statements in the screen (instead of using a menu)		9%
Extend for supporting program development in a conventional language	33%	
No improvement is needed (or no proposal was made)	68%	54%

Finally, in Table 5 we present students replies to a question regarding the problems that the structure editor of objectKarel helped them overcome. No matter what the problems of a structure editor referred by a number of experienced users are, it seems that the help provided to novice programmers is much more important. Of course, the use of a mixed type of a structure – text editor seems to be the best choice.

Table 5. Replies to the question “what kind of help was provided by the structure editor?”

	Group1	Group 2
No need to memorize the syntax of the statements	50%	52%
Avoiding syntax errors	28%	29%

Guidance during program development	28%	
Easier comprehension of concepts (such as class, inheritance), the way a program functions, the syntax of selection & repetitive structures	11%	19%
Writing programs much faster	11%	14%
Focusing on avoiding logical errors	6%	
No need to use the keyboard for writing step by step	6%	5%

Program animation – explanatory visualization evaluation. Students' replies to a series of questions regarding whether they used program animation and explanatory visualization or not, as well as in what way these features helped them, confirmed the importance of program animation for novices and made clear that the feature of explanatory visualization, which is not so widely known and used, is of great importance too. As a matter of fact, the feature of explanatory visualization, which was used by 68% of the students in the 1st group and 63% in the 2nd, seems to be much more important to novices than expected, since it provides an alternative way of comprehending concepts, program flow and debugging. In Table 6 we present students' replies to questions regarding the use of various program animation techniques and explanatory visualization, while in Tables 7 and 8 we present the help provided by these features, program animation and explanatory visualization respectively.

Table 6 Replies to the questions regarding the “use of program animation – explanatory visualization”

	Group 1	Group 2
Step-by-step execution	100%	88%
Tracing	22%	17%
Run	11%	8%
Explanatory visualization	68%	63%

Table 7 Help provided by “program animation (step-by-step execution, tracing, run)”

	Group1	Group 2
Step-by-step execution was used for:		
Detecting and correcting the errors	56%	59%
Watching the consecutive execution of statements – watching what each statement does	33%	14%
Better comprehension of statements – the way that selection and repetition statements work – the way each statement works	17%	14%
Seeing the explanation of each step (without time limit)	17%	
Executing with my own pace – stopping when I wanted	11%	
Making sure that I am in the right “path”	6%	
Tracing was used for:		
Detecting errors in my classes	11%	
Watching the complete program executing	11%	14%
Direct implementation of a program and gradual watching of its progress		5%
Run was used for:		
Executing when I was (almost) sure for the correctness of the program	11%	5%
Direct implementation of a program and gradual watching of its progress		5%

Table 8 Help provided by “explanatory visualization”

	Group1	Group 2
Better comprehension of the functionality of a program	42%	20%
Better understanding of statements – detailed understanding of each step	33%	13%
Easier detection of errors	33%	47%
Helped me correct errors or improve my program	17%	7%

3. Conclusions

objectKarel is an educational programming environment based on the mini-language of Karel++ and the microworld approach to teaching programming. The environment was designed according to established design principles [7, 8], and uses technologies that have proven to support the learning of programming. However, objectKarel has some novel, or at least not so common, features too: incorporation of e-lessons and hands-on activities, a structure editor that is not as strict as usual, and explanatory visualization. The features of objectKarel were evaluated by two groups of volunteer students from a department of Applied Informatics that attended 5 two-hour lessons.

The analysis of the questionnaires showed that *students use the system according to its design principles*, since students used all the features and evaluated them positively. Step-by-step-execution, incremental testing with immediate feedback and visualization are features that should be definitely taken into account when designing educational programming environments. However, our study allowed us to propose some *design guidelines* that should also be considered:

- *Incorporate brief and concise theory and hands-on activities to the environment*, in order to support students in comprehending programming concepts and problem solving (assist students during program development).
- *Use a combination of a structure – text editor*. The use of a structure editor that is not as strict as more structure editors are, proved to be a good choice, but it is not free of problems. An editor that combines the support of a structure editor (no need to memorize syntactic details, avoidance of syntax errors, focus on acquiring programming concepts and problem solving abilities) with the established features of a text editor (cut, copy, paste, undo functions, easier editing) seems to be the best choice.
- *Use explanatory visualization in combination with program animation*. Explanatory visualization is not just an alternative way of watching a program executing, as one might think. Explanatory visualization – text explanations in natural language - supports students in comprehending better the semantics and the functionality of a program and should not be overlooked.

Acknowledgements This research is being funded by the Greek Ministry of Education and the European Union as part of the project "Pythagoras II- Funding of research groups in the University of Macedonia".

References

- [1] L. M^oIver, Evaluating Languages and Environments for Novice Programmers, *Fourteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002)*, Middlesex, UK, 100-110, (2002)
- [2] S. Xinogalos, M. Satratzemi and V. Dagdilelis, An Introduction to object-oriented programming with a didactic microworld: objectKarel, *Computers & Education*, Volume **47**, Issue 2, 148-171, (2006)
- [3] J. Bergin, M. Stehlik, J. Roberts and R. Pattis, *Karel++ - A Gentle Introduction to the Art of Object-Oriented Programming*. 2nd edn. , Wiley, New York (1997)
- [4] P. Miller, J. Pane, G. Meter and S. Vorthmann, Evolution of Novice Programming Environments: the Structure Editors of Carnegie Mellon University, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213-3890 (1994)
- [5] P. Brusilovsky, Program Visualization as a Debugging Tool for Novices. ACM, INTERACT '93 and CHI '93 conference companion on Human factors in computing systems, pp. 29-30, (1993)
- [6] J. Nielsen, J.: Enhancing the Explanatory Power of Usability Heuristics, *Proceedings of CHI 1994*, pp. 152-158, (1994)
- [7] J.F. Pane and B.A. Myers, Usability Issues in the Design of Novice Programming Systems, Technical Report CMU-CS-96-132, School of Computer Science, Carnegie Mellon University (1996)
- [8] J. Stasko, J. Domingue, M. Brown and B. Price (eds), *Software Visualization: Programming as a Multimedia Experience*, The MIT Press, Cambridge (1997)