The Interaction between ICT and Didactics

V. Dagdilelis
Aristotle University of Thessaloniki Faculty of Preschool Education, Florina

M. Satratzemi
University of Macedonia, Dept. of Applied Informatics

G. Evangelidis
University of Macedonia, Dept. of Applied Informatics

Paper presented at the European Conference on Educational Research Edinburgh, 20-23 September 2000

1. Introduction

Nowadays, ICT is being used in education more and more. The invasion of ICT in education is happening from kindergarten to life long adult education in many varied ways. It is thus logical to conclude that this continued invasion of technology in education will lead to a situation where technology and education coexist. This interaction has also caused an increase in both individual and joint research in this field. However, the relevant research often focuses on technology rather than on teaching. We believe that technology is not an end in itself but an intermediary. To be more exact, technology must not only be a intermediary but should become the most appropriate tool for teaching. In other words, the didactic situations must determine the use of technology and not vice versa.

In this paper, we argue that the development of new teaching environments and tools must be subjected to didactic analysis in order to contribute substantially to teaching. As examples, we mention the concept of *recordability*, i.e. the ability of a system to record the behavior of a user and *the systems of automated evaluation*, i.e. environments that automatically validate student output to given problems. In the first case a teacher can examine not only the final output of a student, but also his entire course. In the case of an automated evaluator, specific sets of data can determine some of the student's misconceptions. Thus, both can give us valuable results concerning those misconceptions.

2. Didactic and Technological Rationale

"No matter what technology is used, insights obtained from traditional courseware design both from a pedagogical and content/form point of view must not be ignored", Maurer, 1997 [8].

Maurer's position summarizes our beliefs regarding the relationship between ICT and didactics. The role of teaching is to educate, or according to the classic theory of Piaget, to help the student adapt to an ever changing environment and absorb new information. Despite that, one often gets the feeling that the processes of teaching and educating that rely heavily upon the new technologies, miss the point and become marginal in favor of other objectives.

There are many researchers and intellectuals that point out the fact that the usage of a certain technology as the foundation for teaching a course does not automatically imply the didactic effectiveness of the given course. The hopes of the educational community for a better education are often based upon the advent of the technological discoveries, like radio, television, or video. But, a quite large number of studies [11] conclude that these hopes never come true, since there is no evidence that there is a difference among students educated using new technologies and those educated in the traditional way.

Although not impossible, it is very difficult to point out any one common characteristic in a series of events over a 50-year period. However in the case of the usage of computers in education, the latest link in the chain of technological discoveries that have greatly affected our civilization, we believe that the common characteristic is the lack of a *didactic questioning*.

As an example of the Greek practice, which is usually attuned to international practice, we mention that the calls for development of educational software describe technical rather than didactical specifications. These calls often contain explicit requirements for multimedia software [12]. There is neither justification nor analytical data however, to support the belief that multimedia educational software is didactically more effective than other types of educational software, for example, software based on open *microworlds*.

We believe that didactic rationale and didactic needs should guide the development of educational software. Then, the software can open up new possibilities and redefine to some degree the factors affecting the didactic situations. This is fundamentally different than the practice of assigning the responsibility for essential teaching to technology alone. For example, the *dynamic nature* of the mathematical objects in open microworlds (numbers in Excel and geometric shapes in Cabri-Geometer [1]) allows the development of didactic activities with mathematical content that were inconceivable in the traditional setting of pencil and paper, where numbers and shapes are static. Relevant research shows that microworlds become didactically effective only when they deal with "classic" didactic problems, such as the distinction between a drawing and a geometric shape.

In the following, we describe two collaborating environments used to teach programming that were designed taking into consideration mainly didactic problems that usually come up when teaching programming. We believe that the capabilities of these environments can become a model for designing educational environments that consider didactic needs.

3. Two didactic problems

A usual didactic problem that is related to student activities in Mathematics, Computer Science, Physics and other sciences, is the ability to record the history of the student activities, or in other words, the path followed to a solution or no solution by the students for a given problem. In the traditional classroom, the teacher can simply "see" *snapshots* of the path to the solution rather than the whole path. A rare exception occurs when the teacher can have access to the draft pages used by the students in written exams, or when there is an experimental setup where the actions of the subjects in this case students, are recorded in detail by an observer either a human or a machine.

Knowing the path to the solution followed by the students is most of the times extremely valuable information to someone who wants to explore the conceptions of the students. We believe that the capability to systematically record the path to the solution followed by a particular student, can create interesting new possibilities for exploring the conceptions of students and for developing a clever on-line help in computer programming environments.

In an educational software environment one could have the computer systematically record the actions of a student, thus providing the teacher with invaluable information about the path to the solution followed by that student, the steps backward, the repeated tries, the mistakes, as well as the hesitations. Educational software can be designed to record and store what is didactically essential. The teacher can explore any snapshot of a student's solution while the student is using the software or at any time later.

A related issue that is of great importance is the verification of the final solution given by a student to a given problem. The approach to this didactic problem is relatively simple when the proposed problem is formulated in a way that it accepts a unique and predefined solution. All one has to do is ascertain whether the student's solution coincides with the correct one, as in the case of multiple choice or closed type questions. The problem, however, is much more complicated if the solutions of the students are "open", as in the case of the creation of a geometric shape or a computer program. In these cases, the answers are not unique and predefined but can nonetheless be described in a formal way. Therefore, there exists theoretically an automated way to determine the correctness of the student's solution. We used such a method in an environment for teaching Geometry and we had some enlightening results [5].

Based on the above didactic assumptions we developed two environments that can be of great assistance to teachers of programming.

4. The environments AnimPascal and Telemachus

The first environment is a programming environment called AnimPascal [10], which includes an editor, a compiler, and a visualizer of the source code. The essential feature of this environment is that it automatically saves a snapshot of the program code each time a user executes it. This feature was implemented based on the assumption that the student decides to execute the program code whenever he believes that an important step in the process of developing his program has been reached. The gathered program code snapshots could be useful in finding out the students' errors, and following that, their conceptions when they try to solve a problem of an algorithmic or programming nature.

The later environment is a software tool called Telemachus [9] that tests and grades student programs and provides reliable performance data that could give a reasonable gauge of student knowledge and in this way contribute to teaching programming skills. Many errors in student programs have an element of chance and are thus unpredictable [6]. There are some errors, however, which are more systematic and more persistent and since they are due to student misconceptions they can be predicted. Students produce programs that are correct for most of the cases but when these programs are tested for some data sets they beget incorrect results since students do not take into consideration all the cases. Telemachus validates student programs, running them against a number of predefined data sets rather than against random data sets, so as to detect logical errors. We choose adequate data sets in such a way that a program with logical errors will produce an incorrect output or it will have an incorrect performance (infinite loop). Therefore, some data sets will cause student programs to give incorrect outputs whereas other data sets will cause correct outputs. The combinations of the chosen data sets give valuable insight into student conceptions.

5. Using the two environments together

AnimPascal was used in an experimental setup with first year students of the department of Applied Informatics of the University of Macedonia, in an introductory course to programming. The course consisted of a 2-hour lecture and a 2-hour laboratory under the supervision of the teacher. In the laboratory course the students used AnimPascal. The students were asked to solve the binary search problem. The algorithm had been taught a month and a half before the laboratory course. The problem was formulated as follows:

You are given an array A with M elements (numbers) sorted in ascending order and a number K. You are asked to develop a program that locates number K. If K appears in the array the program should *return the index of the array position that holds it, otherwise, it should return 0. You should use the binary search algorithm.*

The binary search algorithm was chosen because it is an algorithm well known to the computer science community for its deceptive simplicity. While the algorithm appears to be simple, there are certain peculiarities one has to take care of when trying to program it and verify its correctness [2]. Lesuisse [7] showed that even published versions of the binary search algorithm contain errors, weaknesses, and special cases (for example they require M to be a power of 2).

The students submitted their completed programs to Telemachus. Telemachus tested all programs in an automated, systematic way with input data prepared by the teachers.

The automated analysis of the results generated by Telemachus, in combination with the findings of AminPascal, allowed us to get an overall idea of the student errors which we summarize in the following table:

1	did nothing at all	1,4%
2	solved it correctly	5,8%
3	some syntactic errors	4,3%
4	split the initial array into two parts, then searched each part sequentially	4,3%
5	did not use a loop	4,3%
6	wrong termination condition (related to point 9)	17,4%
7	computed the middle element once for the initial array but not for each resulting sub array	1,4%
8	presumed that the element existed in the array	8,7%
9	wrong computation of the index of the middle element	7,2%
10	wrong computation of the boundaries of each sub array	10,1%
11	confused the index of the array holding the number with the number itself	10,1%

12	correct termination condition, but wrong display condition	1,4%
13	used the "for" statement but computed anew its index values hoping to reduce the size of the array	1,4%
14	difficulties in formulating the termination condition (used AND) that resulted in repeated modifications of the code	2,9%

Table 1: Summary of student errors

Table 1 directed us to a more concise analysis of the paths to the solution followed by students. This analysis provided us with qualitative data for each student. Table 2 shows a typical path followed by a student; the timeline can provide a better understanding of the path followed by a student.

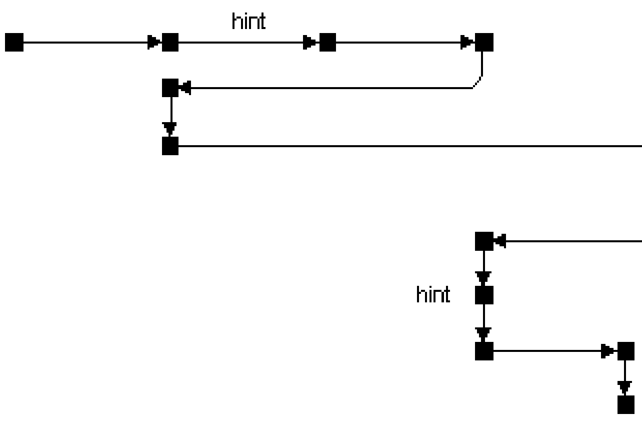
SYNTACTIC ERRORS			LOGICAL ERRORS		SEMANTIC ERRORS		
used parenthesis instead of bracket	used div	error in if...then...else statement	while and AND	value assignment	Initialization of LowLimit, HihgLimit	confused I, A[I]	condition in while statement
							

Table 2: Typical case of the path to a solution followed by a student and the corresponding timeline.

The typical timeline presented in Table 2 helps us establish the following:

- The continuous movement of the timeline between the syntactic and logical errors clearly demonstrates the difficulties posed to the novice student programmer by the idiosyncratic nature of "real" programming languages [3, 4]. In particular, the repeated attempts of students to correct syntactic errors shows that students should be introduced to programming through languages with simple syntactic rules. Although the time spent in correcting syntactic errors is not shown in Table 2, it is clear that the syntax of the programming language can be a negative factor in problem solving.

- Similarly, it is clear that the error messages generated by the system can play an important role in the process of problem solving. In our example, the hint for using operator DIV is an effective help to the user, whereas the message about the array boundaries is incomprehensible. The student repeatedly attempts to correct his errors without understanding the meaning of the error message produced by the compiler. The above observation suggests that a systematic study of student errors and an investigation of the reasons students cannot take advantage of the compiler error messages could improve the produced error messages. Our empirical studies show that this could be achieved by using more detailed and/or translated in to Greek error messages.

- A study of the timelines brings to light the parts of the algorithm students find hard to implement. For example, regardless of their final solution, almost all students had problems in determining the boundaries of the sub-array to search next. The timelines indirectly indicate the way students think. It is clear that most of the students use simulation to build their programs: they "run" their code in a virtual machine in their minds.

6. Conclusions

The development of these environments does not only have a practical value but it is also paradigmatic. The described tools offer us data of didactic interest not only in the area of teaching programming, but also in the broader field. The results obtained from both environments support what we have claimed. Thus, both these environments point out ways of using ICT with a didactic rationale. We maintain that what is usually missing from the educational environments is not ICT but the didactic knowledge of the subject at hand, as well as a formulated theory which could direct the use of ICT in teaching. This framework could consist of a general guide not only for recognizing student conceptions but also for developing didactic situations of the various subjects to be taught. We believe that intensive research is required in order to develop such kinds of tools and better exploit the full potential of contemporary ICT.

6. References

1. Baulac Y., Bellemain F., Laborde J.M. (1988), *Cabri-géomètre, un logiciel d'aide a l'apprentissage de la géomètrie*, Cedic-Nathan, Paris.

2. Bentley J., (1986), *Programming Pearls*, Addison-Wesley,.

3. Brusilovski, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. & Miller P., (1997), Mini-languages: a way to learn programming principles, *Education and Information Technologies 2*, pp. 65-83.

4. du Boulay, B., (1989), Some Difficulties Of Learning To Program, *In Studying The*

Novice Programmer, Soloway, E., Sprohrer, J. (Eds.) Lawrence Erlbaum Associates, pp. 283-300.

5. Dagdilelis, V., (1993), Cabri and HyperCabri: An Intelligent Environment for Teaching and Learning in Euclidean Geometry, (in Greek), *Hellenic Conference on Informatics,* Athens, 1993.

6. Dagdilelis, V., (1996), Computer Science Didactics. Teaching Programming: student conceptions on program development and verification and didactic situations that shape them, *Ph.D. Thesis*, Department of Applied Informatics, University of Macedonia.

7. Lesuisse R., Some Lessons Drawn from the History of the Binary Search Algorithm, *The Computer Journal*, Vol. 26, n. 2, 1983, pp. 154-163.

8. Maurer, H., "Necessary Ingredients of Integrated Network Based Learning Environments", *Proceedings of ED-MEDIA and ED-TELECOM 97 Conference, AACE*, Calgary, Canada, 1997.

9. Satratzemi M., Dagdilelis V., "Telemachus an Effective Electronic Marker of the Students' Programming Assignments", poster presentation, *Proceedings of the 5th Annual Conference on Innovation and Technology into Computer Science Education (ITiCSE'2000-ACM),* Helsinki, Finland, July 11-13, 2000.

10. Satratzemi M., K. Chatziathanassiou, V. Dagdilelis, "AnimaPascal: An educational environment to support teaching of introductory programming courses" (in Greek), *Proceedings of the Second Conference on Information and Communications Technologies in Education*, Patras, Greece,October 2000.

11. The non Significant Difference Phenomenon: http://nova.teleeducation.nb.ca/nosignificantdifference/

12. http://odysseia.cti.gr