Effect of algorithms' multiple representations in the context of programming education

Stefania Siozou, Nikolaos Tselios, Vassilis Komis

Department of Educational Sciences and Early Childhood Education,
University of Patras
(siozous@upnet.gr; nitse@ece.upatras.gr; komis@upatras.gr)
University of Patras, 26500 Rio, Patra, Greece

Abstract

The goal of the study presented in this paper is to compare the effect of different representations while teaching basic algorithmic concepts to novice programmers. Towards this goal, a learning activity was designed and mediated with two conceptually different learning environments, each one used by a different group. The first group used the learning environment "Visual Flowchart", which enables students to construct and examine an algorithm using visual representations of typical flowchart objects. The second group used the software "Language Interpreter", which allows students to express an algorithm using pseudocode. Analysis of results among the two groups showed no statistically significant differences in the students' performance with respect to the tool they used to solve the presented activity, the school stream they followed in high school and their gender. The aforementioned learning environments are also discussed in detail.

Keywords: didactics of informatics, teaching computer science, teaching algorithms, multiple representations, flowcharts, pseudocode, Visual Flowchart, Language Interpreter, programming education, novice programmers.

Introduction

An algorithm is a definite list of well-defined instructions for completing a task and solving a problem. Teaching of algorithms, in the frame of computer programming, poses specific challenges in comparison to other learning domains (Sajaniemi and Kuittinen, 2005, Kolikant and Ben-Ari, 2008). The student is exposed to novel, previously unforeseen, attributes of the computational technology. The teacher attempts to help her understand that she cannot only command the machine to execute specific orders, but she can also program the machine to carry out a set of predefined activities, according to specific rules. Mastering this process has a prerequisite of developing various skills, both abstract and concrete (Nardi, 1993). First, the student has to gain a general understanding of what the machine is capable of and conceptualize the basic logic that rules a programming language (Komis, 2001, 2005). In addition, she has to transform her approach for a solution in a given problem, in a concrete manner and finally express it with respect to the particularities of a given language.

Prior to the algorithm construction, the student has to develop skills in identifying the key aspects of the problem and planning a suitable solution, using tools to design abstract diagrammatic notations such as flowcharts. Following the pioneering work in cognitive theories of problem solving activities by Newell and Simon (1972), attempts to define the ill structured cognitive activity of programming have occurred (Shneiderman, 1980). This activity contains processing and conceptualisation of both semantic and syntactic information, the structuring of regularly-used information into schemas, and the solution of design problems in terms of previously acquired and frequently modified plans (Chang, 1990).

The need to communicate various but often complementary skills to the student, seems to shape a fertile area for introducing programming concepts using multiple representations. Algorithms can be expressed in various notations, including flowcharts and pseudocode. A flowchart is a schematic representation of an algorithm or a process. Pseudocode is a compact and informal high-level description of a computer algorithm that uses the structural conventions of programming languages, but omits detailed subroutines, variable declarations or too formal language-specific syntax which is one of the most significant difficulties that beginner students face (Brusilovsky et al., 1997).

The fundamental difference in the level of abstraction between pseudocode and diagrammatic representations, such as the flowcharts, seems to give an efficient alternative to present basic algorithmic concepts to students. As a result, it is reasonable to expect that different representations will stress different types of information at the expense of others. In addition, the possibility to deal with algorithm creation using multiple representations in an introductory level is complementary by nature, since a notation's understanding is often not complete and/or correct (Blackwell et al., 2001). According to Bohl (1971), flowcharting helps distinguish between the procedure a computer program is written to express it and the syntactical details of the language in which the program is written. Bohl, treats the flowcharting task as an essential tool in problem solving and argues that it is a strong indicator of one's ability to anticipate and analyze a problem, plan the solution and solve the problem.

Previous studies (Crews and Ziegler, 1998, Scanlan, 1987, 1989) have compared the use of these different tools by novice programmers, concluding that students perform better when using flowcharts. Crews and Ziegler (1998), conducted an experiment on 58 students, during the course "Introduction to Programming in BASIC". Students had been introduced to loops and conditional statements, and flowcharts had been used in both classes to illustrate the semantics of loops and conditional statements. The activity presented to the students involved three algorithms of different degree of difficulty: simple, medium and complex. Each algorithm had a flowcharted version and a QBasic program version involving simple conditions and outputs. Three metrics were given to the students and after they executed the algorithms they had to calculate the result of each implementation. Analysis of the results showed that students performed better when they used flowcharts since they needed less time, had higher self-reported confidence about the correctness of their answers and their solutions were more correct. Thus, Crews and Ziegler (1998) concluded that beginners familiarize with algorithms more effectively when they use flowcharts compared to using actual programming environments.

On the contrary, previous studies conducted by Shneiderman et al. (1977), investigated the use of flowcharts in understanding Programming. They concluded that flowcharts transport neither more, nor less information from Programming Languages. In addition, Atwood and Ramsay (1978) found that a program design language was actually better than a flowchart for software design by graduate computer science students. Finally, Brooke and Duncan (1980a, 1980b) found that flowcharts did help in tracing execution flow but were of little help in debugging. In specific, despite that flowcharts helped to localize the area where a bug was, the students were unable to identify it.

Scanlan (1987) studied students' preference between the use of flowcharts and pseudocode. The sample size of the study was 39 students of various ages. All subjects had very similar background-knowledge in flowcharts and pseudocode. A questionnaire was given to each student. Subjects were exposed randomly to both pseudocode and fllowcharts during lectures prior to the administration of the questionnaire, for 6 weeks. The hypotheses of this research were derived from a neurocognitive model, which states that the left hemisphere of the brain, processes information sequentially, verbally and logically, it processes verbal algorithmic techniques, such as pseudocode. Consequently, pseudocode stimulates the left part of our brain. However, flowcharts also contain information that can stimulate the left department. The right hemisphere of the brain processes information simultaneously and spatially. Flowcharts stimulate this department of the brain too, while the pseudocode does not (Bayman and Mayer, 1988). Thus, the entire brain is stimulated by flowchart representation, while only the left part is stimulated by pseudocode. Therefore, Scanlan (1987) hypothesized that students prefer flowcharts. The results of the research, confirm his hypothesis and indeed, students preferred flowcharts and comprehend them better. The only circumstance under which they prefer to be taught in pseudocode is when they are asked to write a Program immediately after learning the algorithm.

Cunnif and Taylor (1987) devised a graphical language FPL, which is 'a structured flowchart informationally equivalent to Pascal'. FPL was superior compared to Pascal, in their experiment involving speed and accuracy of novices on recognition of simple structures, flow of control, input/output and evaluation of simple program fragments. A subsequent study (Cunnif et al., 1989) investigated novices' program construction. They concluded that although certain typical semantic errors are similarly common in both FPL and Pascal, certain other bugs appear to be rarer in FPL. According to the authors the FPL's spatial arrangement allows the user to think more clearly about loops and missing initializations. According to Green (1989) the critical factor determining comprehensibility of notations is accessibility of information. He argues that certain types of information are likely to be accessed more easily from diagrams.

In addition, other studies report that almost 75% to 83% of students are "visual" students (Fowler et al., 2000, Thomas et al., 2002). A "visual student" is the student that remembers something better when she is exposed to representations such as tables, diagrams, pictures, videos, concept maps, graphs and presentations. Consequently, teaching seems to be more effective when the ideas, the concepts, the data and all the relevant information are presented using rich visual representations. Scanlan (1989) conducted a study involving 193 students and 16 learning sessions, in which he investigated their preference when learning introductory and simple algorithms using graphical representations (flowchart) and lexical representations (pseudocode). The results indicated that students prefer flowcharts. The preference for flowcharts ranged from 75.1% to 89.1%. In specific, the results indicate that students preferred flowchart presentations in their books. Moreover, they favored the usage of flowcharts while comprehending algorithms, and preferred to view a structured flowchart representation prior to a structured pseudocode version, when they were given the opportunity to use both techniques.

According to the aforementioned research efforts, one could conclude that the use of flowcharts is preferred, as opposed to the use of Programming Languages or pseudocode, by novice programmers (Larkin and Simon, 1987). In some cases, research has indicated that flowcharts are easier to use, comprehend and help novice programmers. Moreover, the visualization that they offer is an important help for the users. On the contrary, programming languages require strict syntax and structure, facts that make their use difficult, while at the same time they do not allow users to focus on the concepts and content of programming (Scanlan, 1989). However, alternative representations, such as pseudocode, seem easier to handle while in the process of debugging a program, or when a programming tasks is too big and/or complex.

The goal of the research presented in this paper is to compare the effect of different representations while teaching basic algorithmic concepts to novice programmers. To achieve this goal, a suitable experiment was designed to study the performance of two groups, in an identical activity of creating an algorithm. In specific, the research question is the existence or not of statistically significant difference in the students' performance in relation with the tool they used to solve the activity, the school stream (*scientific*, focused on 'hard sciences' such as Physics and Mathematics, *technological*, focused on applied Informatics and *social sciences and humanities*, which gives special emphasis to lessons related to social sciences) they followed in high school and their gender.

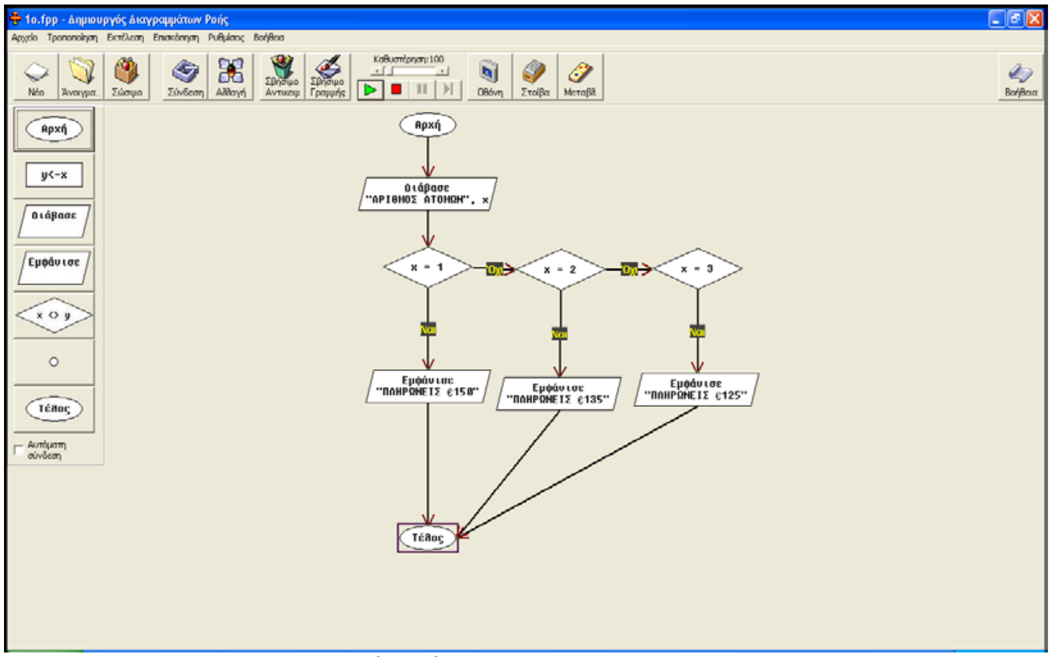
In this paper, the methodology and the context of our study is presented first, followed by a presentation of the learning environments created and used to support the experiment. Subsequently, results of the students' performance while solving the activity are presented and discussed. Finally, we attempt to reason on the findings and propose some ideas for future work.

Methodology

The study took place in the computer lab of the Department of Educational Sciences and Early Childhood Education (DESECE) of the University of Patras. 38 students participated voluntarily in the study (7 male, 31 female, aged 21- 24 years). Among them, 24 were students of DESECE while the rest 14 were students of other departments of the University and the Higher Technological Education Institution of Patras. None of the students of the sample studied in a department related to computer science or other relevant topic. The sample was randomly divided into two groups. The first group consisted of 20 students and the second group consisted of 18 students, since two more students were not able to participate in the study. The same learning activity was assigned to both groups. The first group had to use flowcharts and the second pseudocode to express their solutions.

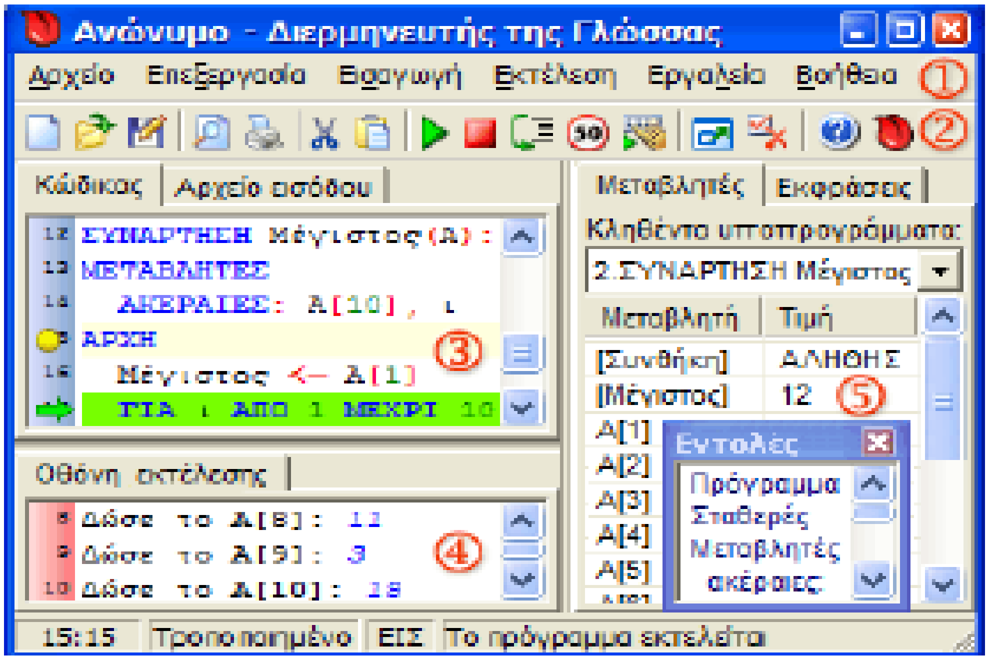
The members of the first group used the 'Visual Flowchart' (VFC), (Tselios et al., 2007, freely available at http://www.ecedu.upatras.gr/flowchart). In this learning environment, the algorithms are developed in the form of a flowchart, and consequently the program implementation process occurs in a visual way. Relative research has shown the learning effectiveness of visual representations of an algorithm (Cunniff and Taylor, 1987, Scanlan, 1989), specifically if the learners are in the introductory stages of a programming course. The program follows the typical design conventions of a direct manipulation application such as menus, toolbars and buttons. The student can select flowchart objects, presented on the left side of the screen, drag them to the workspace and connect them (Figure 1). A representative screenshot of the learning environment is presented in Figure 1 depicting a solution of the activity presented to the students. In the upper part of the screen, a toolbar containing the commands the user needs in order to run the algorithm, is found.

The VFC provides the facility of saving an algorithm in a form that can be "called" from another algorithm as well as from itself, thus supporting the function and recursiveness concept respectively. A function to create an executable file that allows the autonomous implementation of an algorithm without the requirement of the environment is also present. Moreover, the ability to change the delay of each step of the implementation is given. In case that the longest possible delay is selected the algorithm is practically executed step to step, and the program's execution is paused until the students presses the corresponding key (play). In addition, various representative algorithms are available as examples, giving the opportunity to the user to experiment with them. Finally, the user can observe the current values of selected variables and the memory stack while the algorithm is executed.


1o.fpp - Δημιουργός Διαγραμμάτων Ροής
Αρχείο Τροποποίηση Εκτέλεση Επισκόπηση Ρυθμίσεις Βοήθεια
Νέο
Άνοιγμα
Σώσιμο
Σύνδεση
Αλλαγή
Σβήσιμο Αντικειμ
Σβήσιμο Γραμμής
Καθυστέρηση:100
Οθόνη
Στοίβα
Μεταβλ
Βοήθεια
Αρχή
y<-x
Διάβασε
Εμφάνισε
x <> y
Τέλος
Αυτόματη σύνδεση
Αρχή
Διάβασε "ΑΡΙΘΜΟΣ ΑΤΟΜΩΝ", x
x = 1
Όχι
x = 2
Όχι
x = 3
Ναι
Ναι
Ναι
Εμφάνισε "ΠΛΗΡΩΝΕΙΣ €150"
Εμφάνισε "ΠΛΗΡΩΝΕΙΣ €135"
Εμφάνισε "ΠΛΗΡΩΝΕΙΣ €125"
Τέλος
Figure 1. User interface of the 'Visual Flowchart' learning environment

The "Language Interpreter" (LI) was used by the members of the second group to solve the activity. The LI is an educational software used for the creation and implementation of algorithms using pseudocode. The form of the pseudocode, follows the lexical and syntactic conventions as derived by the Greek national curriculum of secondary schools (Greek Pedagogical Institute). It has got similar structural and syntactical characteristics with many classic, widespread programming languages such as PASCAL and C. The LI follows the classical design approach of a window application while at the same time the configuration of the user interface is considerably similar to the majority of the environments that are used for the creation of the typical programming languages. The desktop of the program consists of: a menu bar (1), a tool bar (2) and the 3 parts of the Interpreter, the top part, which is the code writing area (3), the down part, which is the execution simulation screen area (4) and the right part, which presents the used variables, their current values and the available syntactic expression (5), as is it shown in Figure 2.

LI gives the possibility of calling a procedure using the command "Call". In addition, tools to execute the algorithm step by step and examine the current values of selected variables (Figure 2 part 5) are provided. Such functions enable deep exploration of the algorithm's behavior by providing rich feedback on the students' expressed programming approaches. In addition, the system aids the student while structuring and expressing her approach, providing meaningful and detailed error messages supported by clear explanations (Figure 2, part 4), automatic coloring of the words depending on their syntactic importance and automated code alignment (Figure 2, part 3). It also includes embedded help which contains the formal definitions of the commands coupled with representative examples. Finally, the possibility to export the code list in various common formats such as HTML format is also provided.



```
ΣΥΝΑΡΤΗΣΗ Μέγιστος(Α):
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ: Α[10], ι
ΑΡΧΗ
  Μέγιστος <- Α[1]
  ΓΙΑ ι ΑΠΟ 1 ΜΕΧΡΙ 10
```

```
Δώσε το Α[8]: 11
Δώσε το Α[9]: 3
Δώσε το Α[10]: 18
```

Μεταβλητή	Τιμή
[Συνθήκη]	ΑΛΗΘΗΣ
[Μέγιστος]	12
Α[1]	
Α[2]	
Α[3]	
Α[4]	
Α[5]	

15:15 | Τροποποιημένο | ΕΙΣ | Το πρόγραμμα εκτελείται

Figure 2. User Interface of the 'Language Interpreter' learning environment

Description of the activity

The goal of the activity given to the students was to calculate the cost per person of a vacancy trip to a Greek Island. The cost was calculated with the creation of a suitable algorithm. The activity was comprised of five sub-goals characterized by gradually increased difficulty. Concerning the first question, the algorithm had to calculate the cost per person, which varied, depending on the number of individuals that had signed up together (1-3 individuals). To tackle this problem, the students had to put the logical components of the algorithm, which were given to them, in the right order. In order to solve the second question, they had to embed an appropriate select case structure. As a result, when the input reflecting the number of individuals was different than 1, 2 or 3, the message "Invalid number of students" was appeared.

In the third question, the calculation of two new variables was added: a) the variable *s*, which is the amount that will be subtracted from the highest payable amount and is equal to the constant difference 15€ multiplied by the number of students that signed up together and b) the variable *j*, which is the result of the subtraction. In the fourth question, the variable *a* (the possibility of choosing an airplane as a means of transportation), is introduced, as well as the need to check the variable and calculate accordingly the new price of j. Therefore, if the variable *a* is equal to 1, which means that an airplane is used as a means of transportation, the price of the variable *j* (depicting the payable amount) should be increased by 70€ per individual. In the last question, variables *c* and *e*, are introduced and represent the entrance of the students in a bar. The variable *c* is examined, and if it is equal to 1 (which means the entrance in a "club" is selected), the variable *e* takes price depending on the number of individuals that have signed up together. Otherwise, the variable e is 0. In the end of the algorithm, the price of the variable *j* changes and is equal to *e*.

Procedure

An introductory presentation of the two tools was given to all students participating in the study. Information was also provided at that time to inform them about what they would be doing. At the end of each presentation, the students were given copies of the presentations as well as the electronic address from which they could download the two tools (www.ecedu.upatras.gr/algorithmics). Moreover, a supplementary brochure for each tool was given to the students, in case they needed help.

The students were divided into two groups. The first group used the software “Visual Flowchart”, while the second group used the “Language Interpeter” for solving the exercise. However, all students had both tools at their disposal, and if they had time they could try solving the activity with both tools. 20 identical computers of the departments’ computer lab were used, on which the students solved the exercise that was given to them.

All the data from the activities that the students had solved were collected and analyzed. In addition, a questionnaire to report their subjective opinions concerning the usefulness of the learning environment which they used was also given to them. The questionnaire consisted of two parts: the first part referred to the name of the student, the possession of a computer and the school stream followed in high school, while the second part consisted of 12 questions, 2 of which had 3 sub questions, and all of them could be answered on a Likert scale 1-5. The questions referred to flowcharts, pseudocode and the two learning environments.

Concerning the activity, a specific system for evaluating the answers of the students for each question of the exercise was used. The values were given depending on the performance of each student, giving up to ten points for a correct solution for each question. Each student involved in the study gathered a total score with a maximum of 5*10=50 points. All of the students’ files expressing their approaches were saved on their computers. Subsequently, the data were collected and analyzed. After each student finished the exercise, she completed the questionnaire, wrote her name on the forms that she had at her disposal and withdrew. Analysis of the data was conducted using SPSS 15.0.

Results

As derived from the questionnaires, 90% of the first group own a computer and 10% do not. For the second group, the percentage of computer ownership (78%) is smaller than the equivalent of the first group, but is still substantially higher than the percentage of those that do not own a computer (22.2%).

With regard to the school stream followed by the participants, 50% of the first group and 41.2% of the second followed the ‘’Social Sciences and Humanities’ Stream, 15% and 11.8% of the first and second group followed the ‘Scientific’ Stream and 35% of the first group and 47.1% of the second group followed the ‘Technological’ Stream.

In Table 1, the average of the two groups, for all the remaining questions is presented. A Likert scale (1-5) was used. Independent t-tests for each question revealed no statistically significant difference for the answers of each group. However, in several questions the difference in the averages is notable and reaches up to 0.6.

Table 1. Average score of each group for the questions of the questionnaire

Question	**Group 1 (VFC)**	**Group 2 (LI)**	**p- value**
Experience in Programming	2.1	1.8	0.499
Experience in “flowcharts”	1.9	1.6	0.447
Experience in “pseudocode”	1.8	1.8	0.957
Suitability of VFC (LI) for learning basic programming concepts	3.6	3.4	0.581
VFC (LI) Usability	3.9	3.3	0.131
Self efficacy	2.7	2.6	0.827
Was the activity interesting and engaging?	3.9	4.1	0.705
Did you understand flowcharts (pseudocode) in the frame of the activity?	3.7	3.4	0.378
Was this activity suitable for learning basic programming skills?	4.1	3.7	0.290

As far as the presented activity is concerned, the goal of the presented research was to investigate whether certain learner’s attributes influence their performance while solving the activities using the two aforementioned computational learning environments. In specific, the existence of significant statistical importance is examined, in connection with: a) the tool they used to solve the activity, b) the school stream they followed in high school, and c) their gender.

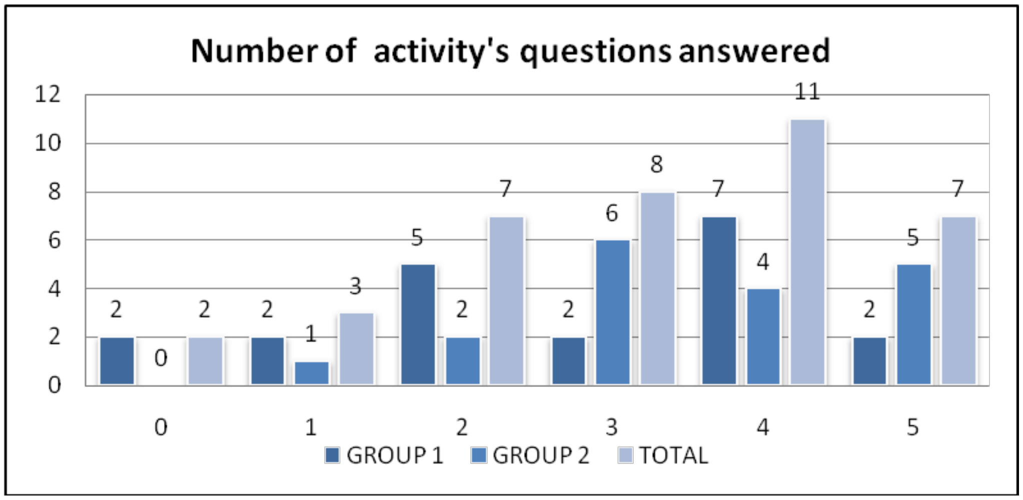


Figure 3. The number of questions answered by the sample

In Figure 3, the total number of answers is presented, but not the success rate on each specific question. In certain cases some students had answered questions, but not in a sequence. Figure 4 presents the number of students of each group that answered each one of the five questions of the activity. As it is apparent in the graph, 34 out of 38 dealt with the first question, 32 with the second, 27 with the third, 16 with the fourth and 9 with the last one. Consequently, as the degree of difficulty increases in the questions, the performance decreases. However, our sample consisted of novice programmers. Thus a weakness in comprehending questions of gradually increasing complexity was expected.

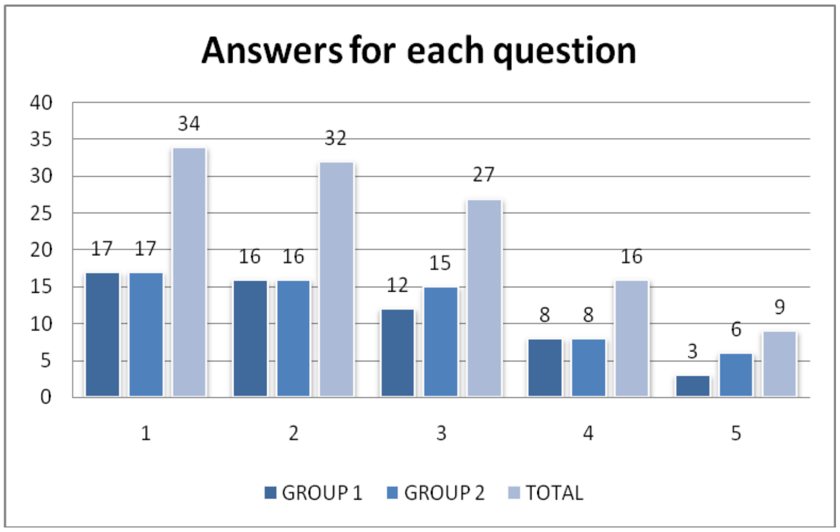


Figure 4. Number of answers from students for each question of the activity

As discussed previously, each member of the sample was given a total score, depicting his performance on the activity. The first group's average performance was 24.3 out of 50 while the second group's was 28.5 out of 50. For the whole sample the average performance was 26.4 out of 50. Therefore, the group that used the program VFC had a lower average than the group that used the program LI, by 4.2 units. For each question, the grades that each team achieved are presented in Figure 5.

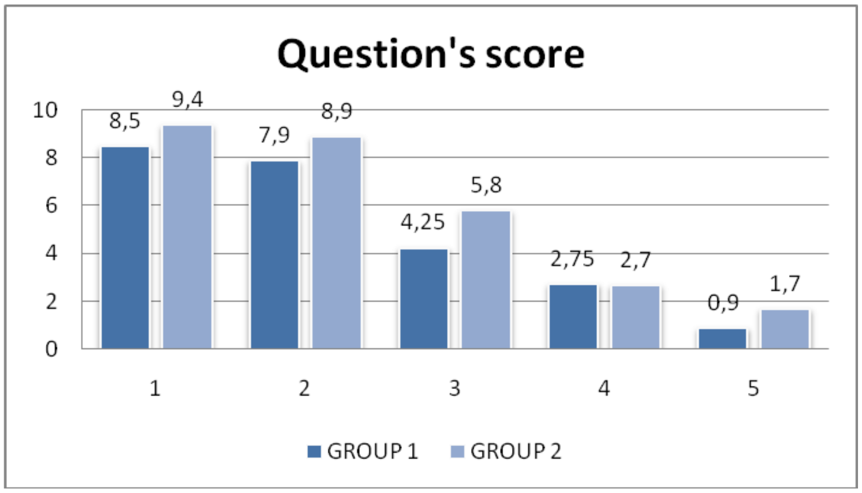


Figure 5. Each group's average for each question

In the first question, the first group had an average of 8.5 (out of 10) and the second group had an average of 9.4. In the remaining questions, the first group had an average of 7.9, 4.25, 2.75 and 0.9, respectively for each question, while the second group had an average of 8.9, 5.8, 2.7 and 1.7. The question number is a strong determinant of each group performance as expected due to its increased difficulty. Specifically, in the two last questions, the performance of both teams is very low. This ascertainment depicts that novice programmers can achieve a higher performance if simple questions, fewer variables, actions and clean logical structures, are preferred.

For the data analysis of both groups' performance on the activity, an ANOVA was performed (Table 2). As aforementioned, the analysis investigated the effects of the tool used by the students to solve the exercise, the school stream they followed in high school and their gender on their performance.

Table 2. Results of ANOVA analysis

		Sum of Squares	df	Mean Square	F	p-value
Student performance according to the tool they used	Between Groups	7.474	24	.311	2.024	.093
	Within Groups	2.000	13	.154		
	Total	9.474	37			
Students' performance per followed school stream	Between Groups	17.892	23	.778	.722	.760
	Within Groups	14.000	13	1.077		
	Total	31.892	36			
Students' performance according to their gender	Between Groups	3.711	24	.155	1.005	.515
	Within Groups	2.000	13	.154		
	Total	5.711	37			

Students' performance according to the school stream followed in high school (Scientific, Technological, Social Sciences and Humanities), is presented in Figure 6 (leftmost part). From the graph it appears that the students that had followed the "Technological Stream" gathered the highest score (28.46 out of 50). This could be attributed to the fact that the particular school stream includes a course in Computer Programming named "Design of Applications in a Programming Environment". However, their difference was small. In specific, 3.1 points higher than students that followed the "Theoretical Stream" (25.35) and 4.2 points higher than the individuals of Positive Stream (24.20). An interesting finding is that even though the "Positive Stream" deals with Mathematics, and Computer Programming is connected directly with them, students' from the "Theoretical Stream" achieved a higher score. As shown in Table 3, the school stream that was followed, does not appear to influence their performance in the activity (p=0.760, ns).

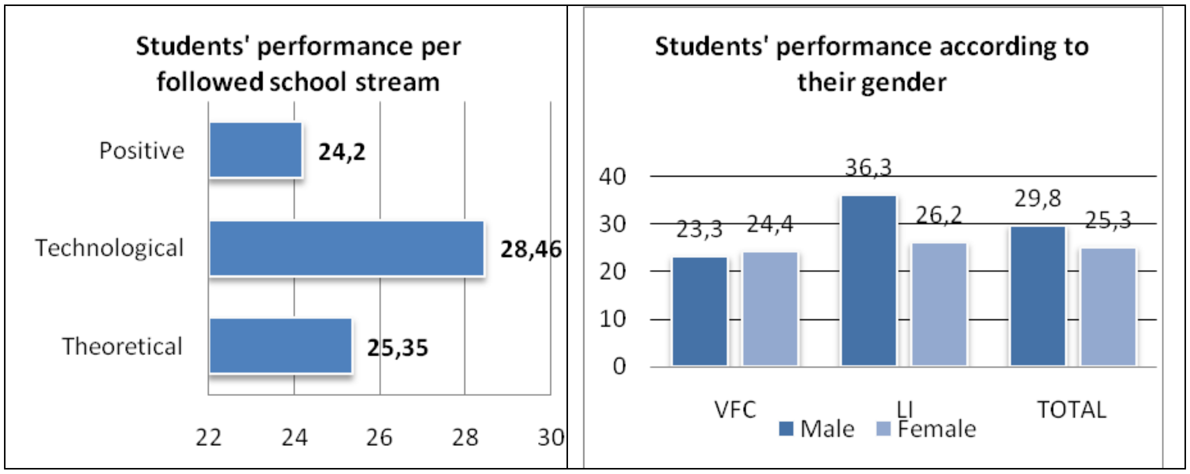


Figure 6. Students' performance per followed school stream (left) and according to their gender (right)

Rightmost part of Figure 6 shows the scores of students' performance according to their gender. From the analysis it resulted that the boys scored 4.5 units higher than the girls. However, the sample was quite unbalanced and female students greatly surpassed the male students (31 girls, 7 boys). Therefore, the difference was not statistically significant (p=0.515, ns). However, it should be reported that male students seem to handle better the lexical and syntactic need of the pseudocode representation, while on the other hand the difference while using the visual representation presented by the flowchart is very small in favour of the female students.

Finally, certain repeated errors and misconceptions were observed during the data analysis, both at an interaction and at a conceptual level, as well as repeated resolution strategies of the exercise. Some students using the VFC experienced difficulties while in the process of saving their solution. In addition, the process of connecting two objects posed some difficulties to the students, since they had to select the two objects and subsequently select the command 'connect'. On the other hand, when they decided to remove a connection, they claimed that there was not clear feedback explaining whether the connected object would be deleted as well, or not. In addition, in some cases the select case object was correctly used but reversely connected, thus leading to inverted algorithm behavior.

As far as Language Interpreter is concerned, some students used mixed English and Greek Language to solve the activity, despite that it was pointed out to them that they had to use English only. For instance, a variable initially described using English characters in the beginning of the algorithm, was afterwards expressed by using the corresponding letter of the Greek alphabet. As a consequence, the algorithm was not able to recognize the variable and thus, it was not executed. In addition, the message error occurred in the 'Language Interpreter' environment for letters common in Greek and English Language (Invalid identifier: 'K') confused them and sometimes distracted them from the solution's definition.

Concerning the pseudocode's syntactic issues, problems with the correct usage of quotation marks and the command "Write" appeared, which are in line with other similar studies (Putnam et al., 1986). In many cases, a student wanted to present the value of a variable, but instead the name of the variable was presented. That is, because the variable was put inside quotation marks, as it is presented in the pseudocode extract presented in Table 3.

Table 3. Extract of an algorithm with wrong use of quotation marks

```
PROGRAM     Travel cost calculation
VARIABLES     s , j
  INTEGER: x
START
  s=15
  j=165
 READ x
 IF x = 1 THEN
  WRITE "You pay j-s"
 ELSE_IF x = 2   THEN
  WRITE "You pay j-2s"
 ELSE_IF  x = 3  THEN
  WRITE "You pay j-3s"
 ELSE_IF  x > 3  THEN
  WRITE  "INVALID STUDENT NUMBER"
 END_IF
END
```

Another erroneous usage of quotation marks leads to double appearance of a required value. For instance, the students wrote: Write «150», j, where 150, is the actual value of the variable j. Moreover, there was some confusion regarding the command "Read". Frequently, students used the command not only for variables, but also for constant values. With regard to the constant values, even though they had assigned a description as a constant value, in the algorithm they requested an input, thus depicting erroneous concept comprehension. Putnam et al., (1986) as well as Dagdilelis et al. (2004) report similar difficulties in previous studies, for the "Read" command. Problems also occurred with the syntax of Selection case, since some times the 'end_if' statement was omitted, leading to algorithm execution denial (Hoc, 1983).

Conclusions

In the present study two alternative learning environments were presented, 'Visual Flowchart' which uses flowcharts and 'Language Interpreter' which uses pseudocode, aimed to support learning of basic programming skills. Subsequently, a case study involving 38 students separated into two groups using one of the aforementioned educational tools towards solving a suitable learning activity is presented. No significant difference in the students' performance was found. In addition, the presented research showed that neither the school stream followed in high school nor the gender of the participants influenced the students' performance on the presented activity. The conclusions of the present study are in contrast to the research that has taken place in the past (Scanlan, 1987, Crews and Ziegler, 1998), which compared usage of flowcharts and pseudocode to educate novice programmers, and wider adoption of flowcharts was depicted.

The lack of statistically significant difference among the two groups could be attributed to the non complicated nature of the given activity. Flowcharting is better suited for visualizing complex algorithms, but in the study presented in this paper, this was not the case. In addition, once a program is expressed in the form of pseudocode, it is then easier to carry out slight modifications while experimenting with its behavior. On the other hand, flowcharts are considered ideal for visualizing some of the fundamental control structures employed in computer programming, thus compensating the advantage of easier modification of algorithms expressed in pseudocode.

Concerning the participants' reflection upon their experience, there was no significant difference in the students' subjective preference in one of the two tools. Both of the tools were considered to be usable and suitable for novice programmers to develop an understanding of flowcharts and pseudocode in creating algorithms. The activity was considered to be interesting, suitable and comprehensible for the introduction of Computer Programming, "flowcharts" and "pseudocode". Finally, the majority of the participants, considered both programs suitable for the introduction in Computer Programming. The aforementioned ascertainments, point out that the presented tools could present a useful learning experience to novice programmers. In addition, since the tools are complementary by nature they could be introduced together in suitably designed activities of increasing difficulty. Introduction of both tools confronts with Kolb's (1984) view of experiential learning, which states that four different students' learning approaches should be supported: active experimentation (learning through doing), concrete experience (learning through feeling), reflective observation (learning through watching and listening), and abstract conceptualization (learning through thinking).

However, it should be stressed out that the findings of the research are dependable from the type of the activity. The procedure of designing and writing a program includes five discrete stages, namely 1) Analyze the Problem, 2) Design a Solution Plan, 3) Construct an Algorithm, 4) Implement the Algorithm and 5) Test and Debug Algorithm (Crews and Ziegler, 1998). The presented study mainly focuses on the third and on the fourth stage. Therefore, it is difficult to generalize the findings of the study without caution. A different activity, involving serious debugging could differentiate the effectiveness of the two tools, from a students' perspective.

As for future work, longitudinal studies of the effect of the different representations in the frame of an introductory, first semester academic course in Computer Science is planned. Such a study could also provide insight for the extent to which the acquired programming skills are retained in the long term. In addition, subsequent targeted studies aimed on teaching different programming structures are considered important to investigate possible variation on the obtained results.

References

Atwood, M.E. and Ramsay, H.R. (1978), *Cognitive structures in the comprehension and memory of computer programs: an investigation of computer debugging*. Alexandria, VA: U.S. Army Research Institute, TR78A21.

Brooke, J.B. and Duncan, K.D. (1980a), “An experimental study of flowcharts as an aid to identification of procedural faults”, *Ergonomics*, 23(4), pp. 387-399.

Brooke, J.B. and Duncan, K.D. (1980b), “An experimental study of flowcharts as an aid to identification of procedural faults”, *Ergonomics*, 23(4), pp. 1057-1091.

Bayman, P. and Mayer, R. E. (1988), “Using conceptual models to teach BASIC computer programming”, *Journal of Educational Psychology,* 80(33), pp. 291-298.

Blackwell, A., Whitley, K.N., Good, J. and Petre, M. (2001), “Cognitive Factors in Programming with diagrams”, *Artificial Intelligence Review*, 15(1), pp. 95–114.

Bohl, M. (1971), *Flowcharting Techniques*, Science Research Associates, Chicago.

Brusilovsky, P., Calabrese,E., Hvorecky,J., Kouchnirenko, A. and Miller, P. (1997), “Mini-languages: a way to learn programming principles”, *International Journal of Education and Information Technologies*, 2 (1), pp. 65–83.

Carlisle, M. C., Wilson, T. A., Humphries, J. W. and Hadfield, S. M. (2004), “RAPTOR: Introducing programming to non-majors with flowcharts”, *Journal of Computing Sciences in Colleges*, 19(4), pp. 52-60.

Chang, S. K. (1990), *Principles of Visual Programming Systems*. Englewood Cliffs, Prentice-Hall, New Jersey.

Crews, T. and Ziegler, U. (1998), “The flowchart interpreter for introductory programming courses”, *Proceedings of FIE '98*, pp. 307-312.

Cunniff, N. and Taylor, R.P. (1987), “Graphical vs. textual representation: An empirical study of novices’ program comprehension”, *in* Olson, G. M., Sheppard, S. B., Soloway, E. (eds.), *Empirical Studies of Programmers: Second Workshop*, pp. 114–131.

Cunnif N., Taylor, R. P., Black J.B. (1989), “Does programming language affect the type of conceptual bugs in begginers’ programs? A comparison of FPL and Pascal”. *In* Soloway, E. and Spohrer, J.C. (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Erlbaum.

Dagdilelis, V., Satratzemi, M. and Evangelidis, G. (2004), “Introducing Secondary Education Students to Algorithms and Programming”, *Education and Information Technologies*, 9(2), pp. 159–173.

Fowler, L., Allen, M., Armarego, J. and Mackenzie, J. (2000), “Learning styles and case tools in software engineering”, In Herrmann A. and Kulski, M.M. (eds), Flexible Futures in Tertiary Teaching. *Proceedings of the 9th Annual Teaching Learning Forum,* pp. 169-180.

Green, T.R.G. (1989), “Cognitive dimensions of notations”, in Sutcliffe, A. and Macaulay, L. (eds.) *People and Computer,* Cambridge University Press, Cambridge, pp. 443 – 460.

Kolb, D. (1984), *Experiential Learning: Experience as the Source of Learning and development,* Englewood Cliffs, Prentice-Hall, New Jersey.

Kolikant, Y. B.-D. and Ben-Ari, M. (2008), “Fertile Zones of Cultural Encounter in Computer Science Education”, *Journal of the Learning Sciences*, 17(1), pp. 1 — 32.

Komis, V. (2001), “Didactics of Informatics: from the Formation of the Scientific Field to the Conjunction among Research and School Practice”, in Manolopoulos, Y. and Evripidou, S. (editors), *Proceedings of 8th Panhellenic Conference on Informatics with international participation*, Greek Computer Society, University of Cyprus, November 2001, pp. 463-471.

Komis, V. (2005), *Introduction to Didactics of Informatics,* Kleidarithmos Press, Athens (In Greek).

Larkin J.H. and Simon, H.A. (1987), “Why a diagram is (sometimes) worth 10,000 words”, *Cognitive Science,* 11(1), pp. 65 – 100.

Hoc, J.M. (1983), “Analysis of beginners’ problem-solving strategies in programming”, In Payne S.J. and van der Veer, G .C. (eds): *The Psychology of Computer Use*, Academic Press, London, pp. 143 – 158.

Nardi, B. (1993), *A Small Matter of Programming : Perspectives on End - User Computing*. MIT Press, Cambridge, MA.

Newell, A. and Simon, H.A. (1972), *Human Problem Solving*, Englewood Cliffs, Prentice Hall, New Jersey.

Putnam, R. T., Sleeman, D., Baxter, J. and Kuspa, L. (1986), “A summary of misconceptions of high school BASIC programmers”, *Journal of Educational Computing Research*, 2(4), pp. 459-472.

Sajaniemi J. and Kuittinen, M. (2005), “An experiment on using roles of variables in teaching introductory programming”, *Computer Science Education*, 15(1), pp. 59-82.

Scanlan, D.A. (1987), “Data-structures students may prefer to learn algorithms using graphical methods”, *Proceedings of the eighteenth SIGCSE technical symposium on Computer science education*, St Louis, pp. 302-307.

Scanlan, D.A. (1987b), A niche for structured flowcharts. *15th ACM Annual Conference on Computer Science*, St. Louis, pp. 408.

Scanlan, D. A. (1989), “Structured flowcharts outperform pseudocode: An experimental comparison”, *IEEE Software*, 6(5), pp. 28–36.

Shneiderman, B. (1980), *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop, Cambridge, MA.

Shneiderman, B., Mayer, R., McKay, D. and Heller, P. (1977), “Experimental investigations of the utility of detailed flowcharts in programming”. *Communications of the ACM*, 20(6), pp. 373-381.

Thomas, L., Ratcliffe, M., Woodburry, J. and Jarman, E. (2002), “Learning styles and performance in the introductory programming sequence”. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education,* Cincinnati, Kentucky, pp. 33-42.

Tselios, N., Georgopoulos, A., Politis, P., Pyrza, F., Fanikos, A., Koubouri, D. and Komis, V., (2007), “Studying the use of multiple representations in the frame of educational programming environment”, *Themes in Education,* 8(1), pp. 3-24.