Modeling and multiple representation systems in the design of a computer environment for the learning of programming and C by beginners

Maria Kordaki

Dept of Computer Engineering and Informatics
Patras University, 26500, Rion Patras, Greece
e-mail: kordaki@cti.gr

Abstract: This study considers a modeling methodology for the design of a computer based problem-solving environment (named L.E.C.G.O: Learning Environment for programming and C using Geometrical Objects) for the learning of computer programming and the programming language C by beginners. For this design, constructivist and social learning theories were taken into account. The general design has taken into consideration models of the learning process and subject matter as well as potential learner behavior in dealing with fundamental tasks. The main emphasis was placed on the role of: a) multiple external representations in student learning, b) motivation through performing problem-solving activities taken from the familiar and meaningful context of drawing and using simple geometrical objects, c) the active participation of students in their learning by using hands-on experience and d) visual feedback on the actions taken by students for their self-correction.

Introduction

Programming is not merely a specific topic in the Computer Science and Engineering curriculum but a 'mental tool' of general interest (Satratzemi, Dagdilelis & Evaggelidis, 2002), a complex task including understanding, method finding and coding (Brooks, 1999). According to Winslow (1996), it can be divided into four steps: a) comprehension of the problem at hand, b) definition of a solution to that problem, initially in any form, such as text-based, math-based, pseudo-code and flow-chart, c) translation of that form into a selected programming language, and d) testing and debugging of the resulting program. Good performance in programming also implies the ability of learners to use various and new representation systems to express their problem solving strategies (Komis, 2001; Brooks, 1999). Students are rarely aware of the problems that can be solved by a computer and the benefits to be had from using programming (Lemone & Ching, 1996). It is worth noting that students encounter serious difficulties in performing all the steps mentioned above (Putnam, Sleeman, Baxter. & Kuspa, 1989; Kurland & Pea, 1989; Lemone & Ching, 1996; Christiaen, 1998; Perkins, Hancock, Hobbs, Martin & Simmons, 1989). Programming in schools is currently supported by professional integrated software environments. Despite their capabilities and aids for the experienced users, these tools are useless for novices with limited programming skills: they provide insufficient feedback, and they are also extremely complicated (Freund, & Roberts, 1996). Thus, the student's frustration with programming often depends less on the target programming language and more on the programming environment in use (Freund, & Roberts, 1996).

Existing studies indicate that there is a need for a novice-oriented programming environment. It is suggested that the success of such an environment is mainly dependent on its ability to: a) support the algorithmic solution rather than the syntactic rules of the specific programming language at hand, b) provide usable coding tools for selecting and combining the language structures, c) support problem solving settings, d) visualize the program and its output, and e) provide meaningful feedback (Freund, & Roberts, 1996; Sangwan, Korsh & LaFollete, 1998; DiGiano, Kahn, Cypher, & Smith, 2001; Brusilovski, Calabrese, Hvorecky, Kouchirenko, & Miller, 1997).

Well-known examples of such environments for the learning of programming in computer language C are BACCII (Calloni, B. & Bagert, 1994; 1997) THETIS (Freund, & Roberts, 1996) and 'Karel the Robot' (Pattis, Roberts & Stehlic, 1995). Despite the incorporation of one or more of the principles mentioned in the previous paragraph, together with fundamental principles of modern constructivist theories of learning, these environments either fail to emphasize learner ability and the need to express their knowledge in different representation systems or offer possibilities to solve a limited set of problems. In addition, the reported environments lack appropriate feedback to help novices to

meaningfully correct their mistakes.

In an attempt to exploit all the above, an open problem-solving computer learning environment was designed to support secondary level education students in their learning of programming and C. This environment allows them to: a) use hands-on experience while actively constructing their own problem-solving strategies b) express their solution strategies in Multiple Representation Systems (MRS), starting from intuitive representations and moving gradually to more sophisticated ones, b) solve a variety of familiar and meaningful problems, c) overcome the cognitive load of the syntactical rules of programming in C by using appropriately designed computer-based authoring tools and d) receive appropriate feedback to correct their mistakes. In the following section, we discuss the rationale behind the design of L.E.C.G.O. Next, the general architecture and the MRS provided are demonstrated and this design is discussed in reference to other reported environments for the learning of programming and C. Finally, we present proposals for future plans.

The rationale behind the design of L.E.C.G.O

The design principles of L.E.C.G.O arose as a transformation of the theoretical considerations of three models (Mellar & Bliss, 1994; Kordaki, & Potari, 1998): a) the subject matter model, based on the literature on basic aspects and structures of programming and C, b) the learning model, based on modern social and constructivist theories of learning acknowledging the active, subjective and constructive character of knowledge as well as the crucial role of tools and of MRS in knowledge construction (Jonassen, 1991;1999; Ainsworth, 1999), and c) the learner model, based on the literature on how students learn essential aspects of programming. As regards the role of MRS, it is acknowledged that computer learning environments that provide a variety of RS of different cognitive transparency could encourage students to select the most appropriate to express their knowledge. These different RS can provide students with opportunities to express their inter- and intra-individual variety (Janvier, 1987). It is noteworthy that most learner difficulties are found in the gap between their intuitive knowledge and the knowledge they need to express themselves in the RS proposed for use. For example, prepositional, symbolic and abstract RS prevent some learners (usually beginners) from expressing their knowledge, the same systems being intended for use by advanced learners. Contrariwise, metaphors of everyday life and visual RS are more suitable for beginners.

L.E.C.G.O. was designed to be a possible learning environment for twelfth grade students (18 years old) and for first-year University students. The programming language C was selected as a learning subject as this is a modern language with great capabilities which could also become a solid background for the learning of object-oriented programming. The models constructed are described below.

The subject matter model

In contrast to traditional behaviorist theories (Skinner, 1968), where the content of the subject matter is strictly organized, starting from the easier and gradually moving to more advanced matters, constructivists put the emphasis on the fundamental concepts of the subject matter (Jonassen, 1991). Thus, the teaching of programming and C should stress the basic structures and not the syntactical rules of the language. The following hierarchical network (Fig.1) depicts the process of solving a problem using C. Each level of the process includes its sub-processes until it reaches the elementary ones, which can be performed using the specific commands of the target programming language. The division of the process has three objectives: a) to allow students to realize the structure of the programming process in C, b) to develop their own problem-solving strategies using elementary processes and the related commands, and c) to make it possible to transform these elements into aids or templates within a programming environment. These templates are elementary authoring tools designed to construct a program in C while avoiding the cognitive load of recalling the specific syntactical rules and the vocabulary of the language.

Taking into account the described architecture of the problem-solving process in C and the requirements of the constructivist learning model (see next section), we do not propose the use of 'drill and practice' activities, but fundamental, authentic, meaningful, real life problems (Jonassen, 1991; Nardi 1996) from the domain of drawing, using simple geometrical objects (Kordaki, 2006). Drawing was selected as a context for the learning activities as it would motivate learners to become actively and passionately involved in their own learning. In addition, drawing using simple geometrical objects was selected to give students the chance to learn about the graphic functions in C. The aforesaid activities can be solved in various ways, motivating students to express their inter- and intra-individual

differences (Calloni & Bagert, 1994). They also can be solved without the extra cognitive load that might stem from the demand to perform complex geometrical constructions.

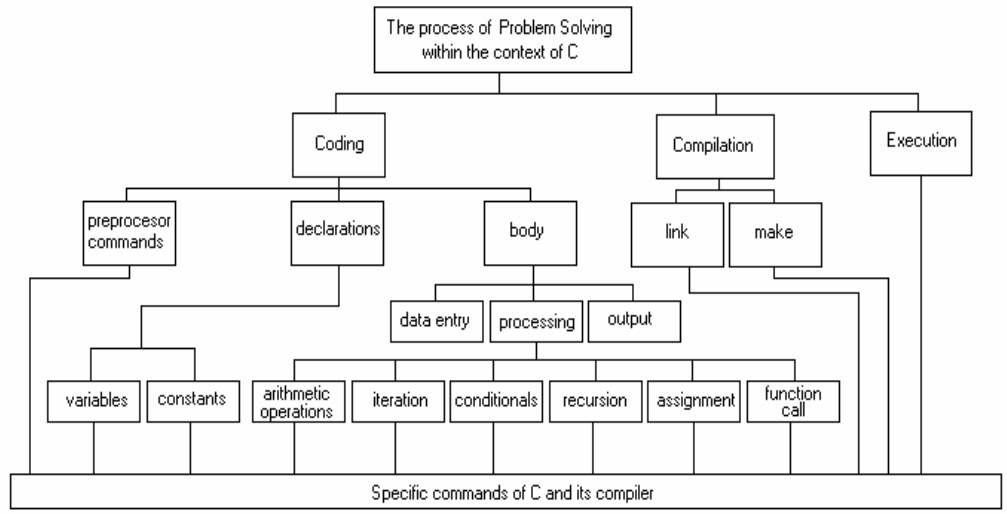

Figure 1. The problem-solving model within the context of programming and C

The learning model

Constructivism requires learning to be reconceptualized as an active, subjective and constructive activity (Jonassen, 1991). This challenge, in combination with the social theories of learning that emphasize the role of computer tools as cognitive tools in the learning process, has inspired the design principles of various open problem-solving environments (Jonassen, 1999; Maureen, 2000). For the design of L.E.C.G.O., basic aspects of constructivism have been interpreted as design principles (Kordaki & Potari, 1998) (Table 1).

Theoretical considerations	Design principles
The active role of the learner	• Interactivity • Providing various appropriate tools
The subjective character of knowledge	Providing tools for: • Constructing MR of learner solution strategies to various problems • Inter-operation • Helping students to construct and study their own computational objects related to the tasks or the concepts to be learned • Supporting students in overcoming their difficulties related to the subject matter
The constructive character of knowledge	Providing tools for: • Supporting a variety of problem-solving activities • Providing intrinsic visual feedback on learner actions, for self assessment • Supporting authentic learning activities in a familiar and meaningful context • Supporting hands-on experience in manipulating primary sources of data related to the problems to be solved

Table 1. The learning model

The learner model

Students often begin programming with a “big handicap”: they do not know how a computer works. This lack of knowledge results in many other “deep misconceptions” as well as “surface errors” that appear as programming bugs (Du Boulay, 1986; Soloway & Spohrer, 1989a; Kurland & Pea, 1989; Mayer, 1989). Students try to explain the functionality of the computer by using anthropomorphic expressions (Putnam, Sleeman, Baxter. & Kuspa, 1989). Thus, lacking a schema that can incorporate new concepts and explain others, students begin programming with an important innate disability.

After students find a solution, they usually tend to omit the “algorithmic solution” and immediately confront the problem (Allwood, 1986). Students cannot easily identify primitive constructs, such as conditionals, recursions and loops, and entities, such as variables, counters and conditions, in their

initial solution; nor can they express them with computer-based structures. There is a big gap between the informal solutions they give and the more formal computer-oriented solution that is required (Winslow, 1996) and the bigger the gap, the more difficult it is for the student to surpass it (Christiaen, 1998). Spohrer and Soloway (1989b) argue that the majority of "surface errors" are caused by these "plan-composition problems" rather than by semantic misconceptions of the language. Students also have semantic and syntactic misconceptions with almost every basic structure and concept of a programming language (Lemone & Ching, 1996; Kurland & Pea, 1989; Putnam, Sleeman, Baxter. & Kuspa, 1989; Samurcay, 1989; Christiaen, 1998; Komis, 2001).

Specifically, the variable is thought to be static and is confused with its algebraic notion. Consequently, the assignment command is interpreted as equality (Christiaen, 1998; Putnam, Sleeman, Baxter. & Kuspa, 1989; Samurcay, 1989; Lemone & Ching, 1996; Komis, 2001). The initialization, updating and testing of variables are also difficult for students, especially when they are encountered inside loops and conditions (Samurcay, 1989). Regarding conditionals, programmers fail to realize the interaction between the serial execution of the program and the condition (Komis, 2001). The condition cannot easily be defined, especially when it uses logical variables or variables that change inside the body of the conditional (Kessler, & Anderson, 1989). Students fail to understand that the conditions are static yet the entangled variables change and, as a result, the same condition can have a different result at each repetition of the loop. Moreover, students seem to interpret "While.. do" as a kind of exception within the program (Lemone & Ching, 1996). Regarding recursion, studies have shown that repetition and loops are more understandable than recursion and that there is no transfer of knowledge from the latter to the former (Kessler, & Anderson, 1989). There are also important difficulties in understanding the way input and output commands function (Putnam, Sleeman, Baxter. & Kuspa, 1989) and, according to Du Boulay (1986), students find it difficult to manipulate arrays. Finally, not only do inexperienced users have difficulties in using a programming language, but, in contrast to those with experience, they also find it difficult to comprehend and use the online help provided by the environment (Allwood, 1986).
In an attempt to help learners overcome these difficulties, L.E.C.G.O. provides them with a number of RS to express their problem-solving strategies and to move gradually from informal solutions to more computer-oriented ones. The RS provided are discussed in the next section of this paper.

The general architecture and features of L.E.C.G.O.

The focus when designing L.E.C.G.O. was on the learning of basic algorithmic structures in C. This was chosen because a considerable amount of research has reported serious student difficulties in understanding the algorithmic logic in problem solving (Komis, 2001). L.E.C.G.O. provides students with opportunities to: a) express their solution strategies in MRS starting from 'anthropocentric', non-necessary programming solutions and gradually moving to more computer-oriented programming solutions, b) acquire hands-on experience in providing graphic solutions to the problems at hand, using tools that support the direct manipulation of computational objects on a computer screen, c) deal with graded difficulty learning activities, within the context of drawing and using basic geometrical objects, d) offer assistance with their problem-solving strategies. This help is provided in three modes: i) as ready specific expressions that could be used to describe, in natural language, a specific algorithmic solution to the task at hand, ii) as ready structures and functions in pseudo-code, provided in the form of buttons and iii) as ready structures and functions in C, also provided in the form of buttons and iv) as visual feedback from the output of students' programs to be compared with their graphic solutions of the problem at hand. It is worth noting that the design of L.E.C.G.O. could be modified for the learning of any programming language. Such an environment for the learning of programming and C has not yet been reported.
The general architecture of L.E.C.G.O. is presented diagrammatically in Figure 2 and also demonstrated on the L.E.C.G.O. home page. The aforementioned architecture is divided into two main parts: a) that presenting an appropriate content for learning fundamentals in programming and C (Kordaki, 2006). The design of this learning content was also based on modern social and constructivist theories of learning. This content is structured in a hierarchical order of five hyperlinked layers, including: i) complex examples (1st layer), ii) simple examples (2nd layer), iii) broad information about programming in C (3rd layer), iv) fundamentals in programming (4th layer) and e) broad information at various locations on the WWW (5th layer) and b) that dedicated to the learning activities that have to be actively performed by the students in order to learn fundamentals in programming and C. This part includes tools for algorithmic solutions to problems in MRS, such as: i) graphic RS providing opportunities to solve problems graphically using Cabri-Geometry II tools (Laborde, 1990). Cabri-

Geometry II is a well-known educational software designed to perform geometrical constructions within the context of Euclidean Geometry; ii) text-based RS providing possibilities for translating the graphic solution given by a student in Cabri into natural language; iii) imperative RS, providing specific expressions in the imperative which could be used for translation of the solution at hand into the imperative; iv) pseudo-code RS, to translate the solutions expressed in the previously mentioned RS into pseudocode; v) C language-based RS and vi) the graphic output of the written programs. These MRS are described in the next section of this paper.

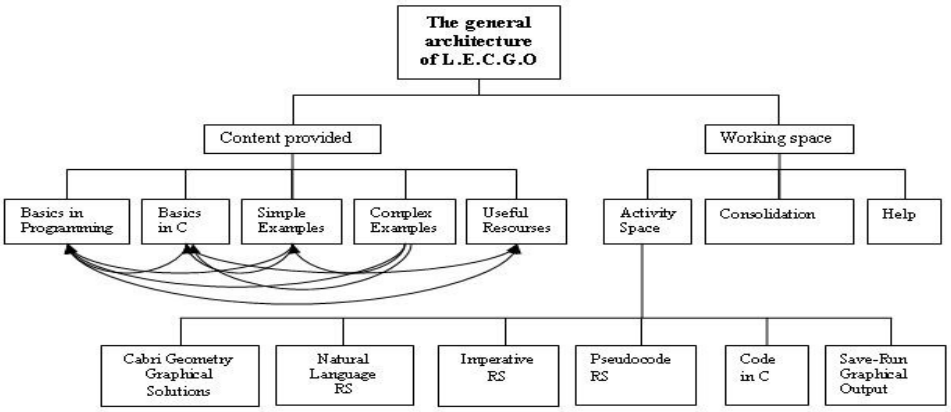

Figure 2. The general architecture of L.E.C.G.O.

Multiple representation systems in L.E.C.G.O

- *Drawing-visual representations using hands-on experience.*

Students can express their intuitive, drawing representations of the solutions to the given problems by using any of the tools included in Cabri-Geometry II. Within Cabri-Geometry II, students can be helped to build various geometrical constructions by acquiring hands-on experience. Students can also see the coordinates of these constructions and therefore use them as values of the related parameters in the corresponding commands in C.

- *Free-text based representations.* Here, students have the opportunity to express their solution strategies using the familiar symbolic system of natural language. This step calls for reflection on their previous hands-on experience (Vlahogiannis, Kekatos, Miatidis, Kordaki & Houstis, 2001) in the context of Cabri-Geometry II. Here, it is worth noting that reflection is an essential process in the development of a student's critical thinking. The text produced is a 'transitional' representation that, as with the previously mentioned graphic representation, acts as a scaffolding element to help students to gradually develop the ability to express their problem-solving strategies in the C language.
- *Text-based representations using imperative and specific expressions*. Here, students have to express their free-text-based solution strategies in the imperative. This has been decided upon to help students to move from the 'I' or 'we' situation to one in which the student gives directions to the computer. The system also provides students with a number of specific expressions in the imperative, to select those most appropriate for the description of their solutions. These expressions are related to the specific available actions that can be performed in Cabri. These are not immediately visible but appear to help students if they are unable to take this step unaided.
- *Pseudo-code based representations*. This is the step where students have to express their imperative-based representations in pseudo-code, using a set of buttons. Each of these buttons shows a basic algorithmic structure or geometrical figure (those that can be represented using a graphic function in C) in the corresponding text-box.
- *Representations in C.* Providing a number of authoring tools was seen as useful to the students. A number of buttons showing the basic component skeletons of a program in C e.g. '*basic structure of a C program*', '*for*', '*while*' and '*if*'' statements, as well as showing basic graphic commands e.g. '*line*', '*circle*', '*rectangle', 'ellipse'* and *'triangle*', are offered in this editor. By clicking on these buttons students have the possibility to view the corresponding code. All these tools can help the students to focus on the essential elements of the language, reducing the cognitive load arising from the need to recall the specific syntactic rules and commands of the language.
- *Graphic output of the written programs.* After students have finished writing the code, they can compile the program and see the results of their programming attempts in another window on the

computer screen. Students can verify their programming attempts by comparing this output with their drawing solutions performed using Cabri-Geometry II. An early description of these representation systems was presented in Zikouli, Kordaki & Houstis, (2003).

On the whole, the construction of the solution to a problem starts from its graphic representation and moves gradually to its representation in the programming language C, through the previously mentioned RS. This was chosen to help students move from intuitive graphic solutions to solutions using pseudo-code and, ultimately, to solutions using the commands of a programming language such as C. Figure 3 shows the solution in the previously mentioned RS of the following simple drawing problem: 'develop a program in C to draw three circles with the same center and three different radii'.

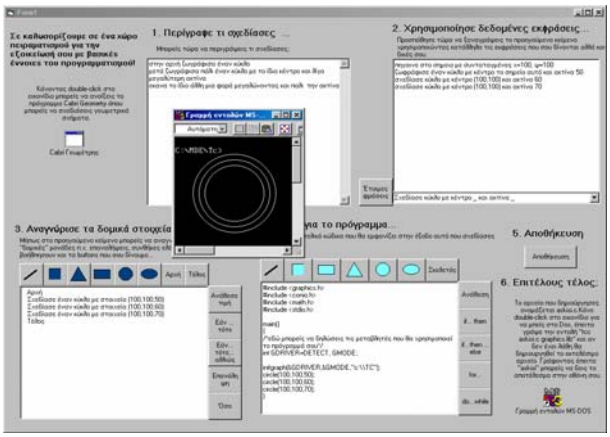

Figure 3. The general interface of L.E.C.G.O

To realize **L.E.C.G.O**, Microsoft Visual Basic 6.0. is used. The final program generated in **L.E.C.G.O** is compiled by the "Turbo C 2.01" compiler . Every action of the student is also recorded in a log file providing the researcher with a valuable bank of raw data.

Discussion and future plans

In this paper, the design and the basic features of a computer environment for the learning of computer programming and language C has been presented. A number of decisions were made to interpret in the design principles both basic aspects of constructivism and social theories of learning, as well as student difficulties in the learning of programming that have emerged from the literature. As a result, a problem-solving environment was designed to facilitate students to express their problem-solving strategies in MRS, based on: a) drawing and visual constructions using hands-on experience and motivating students to be actively involved in their learning as well as encouraging them to express their intuitive knowledge, b) free-text, using natural language encouraging students to reflect on their hands-on experience while constructing their drawing solution, c) text using the imperative, with each sentence describing only one action, assisting students to move from the description of a solution performed by themselves to a solution performed by a computer, d) specific expressions in the imperative, helping students describe their solution strategies appropriately as a set of commands for the computer, d) pseudo-code using basic algorithmic structures, e) programming in C, and g) the visual output of the written programs. All the representation systems above, with the exception of the last, are designed to act as 'transitional' representation systems to fill the gap between students' concrete graphic solutions and the symbolic ones written in C. These 'transitional' systems are designed to act as scaffolding elements for those students unable directly to express their solving strategies in C. The RS provided can also be separated into two categories; those that support the expression of a student's own thinking and those that act as supporting elements for this expression. In fact, students are helped to express themselves by: a) using hands-on experience, b) forming their text-based solutions and c) transforming their solutions into the imperative. On the other hand, students can be helped to express their knowledge by: a) selecting among given specific text-expressions in the imperative, b) using given descriptions of basic algorithmic structures and of graphic functions in both pseudo-code and the programming language C. Students can be helped to understand how a computer works by being asked to explain their solutions both by giving a number of elementary directions to the

computer as well as by using the basic algorithmic structures provided. The intrinsic visual feedback can also encourage students to take control for their learning.
Comparing the design of the proposed learning environment to others reported in the literature, we stress:

- The existence of multiple representations that facilitate the student to smoothly surpass the gap between the intuitive and the formal solution to the problem at hand
- The context of drawing, which minimizes the cognitive load of the student, providing him with both a rich set of interesting, motivating problems and essential intrinsic visual feedback.

Finally, it is worth noting that L.E.C.G.O has been evaluated in the field using real students with positive results (Kordaki, in preparation). However, this evaluation study provided some evidence for the enhancement of this environment by providing appropriate feedback on student solutions to the given problems in the form of correct interpretation of these solutions in all the MRS provided. The design and implementation of such a feedback system is in our future plans.

References

Ainsworth, S.E. (1999). The functions of multiple representations. *Computers and Education*, 33(2-3), 131-152.

Allwood, C. (1986). Novices on the computer : a review of the literature. International Journal of Man-Machine Studies, Vol. 25, 633-658.

Brooks, R. (1999). Towards a Theory of the Cognitive processes in Computer Programming. Int. J. Human-Computer Studies ,51, 197-211.

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

Calloni, B. & Bagert, D. (1994). Iconic programming in BACCII vs. Textual programming: which is a better learning environment?. ACM, SIGCSE '94 3/94, Phoenix AZ,188-192.

Calloni, B. & Bagert, D. (1997). Iconic Programming proves effective for teaching the first Year programming sequence. ACM, SIGCSE '97 , CA USA, 262-266.

Christiaen, H. (1998). Novice Programming Errors: Misconceptions or Mispresentations? ACM, SIGCSE Bulletin, Vol. 20, No. 3, 5-7.

DiGiano, C., Kahn, K., Cypher, A. & Smith, D.C. (2001). Integrating Learning Supports into the Design of Visual Programming Systems. Journal of Visual Languages and Computing, 12, 501-524.

Du Boulay, B. (1986). Some difficulties in Learning to Program. Journal of Educational Computing Research, 2 (1), 57-73.

Freund, S. N. & Roberts, E.S. (1996). THETIS: An ANSI C programming environment designed for introductory use. ACM, SIGCSE '96 2/96, Philadelphia, PA USA,300-304.

Janvier, C. (1987). Representation and understanding: The notion of function as an example. In C. Janvier (Eds), *Problems of representation in teaching and learning of mathematics* (pp. 67-72). London: Lawrence erlbaum associates.

Jonassen, D. H. (1991). Objectivism versus constructivism: do we need a new philosophical paradigm? Journal of Educational Research, 39 (3), 5-14.

Jonassen, D. H. (1999). Designing constructivist learning environments. *Instructional design theories and models*, 2, 215-239.

Kessler, C. M. & Anderson, J. R. (1989). Learning Flow of Control: Recursive and Iterative Procedures. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 229-260, Hillside, N.J. Erlbaum.

Komis, V. (2001). A study of basic programming concepts within a constructivist teaching approach. Themes in Education, 2 (2-3), 243-270.

Kordaki, M. & Potari, D. (1998). A learning environment for the conservation of area and its measurement: a computer microworld. Computers and Education, 31, 405-422.

Kordaki, M. (2006).΄Learning activity΄ as the basic structural element for the design of web-based content: A case study. In T. Reeves & S. Yamashita (Eds), Proceedings of *World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education (E-Learn 2006),* October, 13-17, Honolulu, Hawaii, USA, pp.88-96. Chesapeake, VA: AACE

Kurland, D. M. & Pea, R. D. (1989). Children's Mental Models of Recursive Logo Programs. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 315-324, Hillside, N.J. Erlbaum.

Laborde, J-M. (1990). Cabri-Geometry [Software]. France: Universite de Grenoble.

Lemone, K. A. & Ching, W. (1996). Easing into C: Experiences with RoBOTL. ACM, SIGCSE Bulletin*,* Vol. 28, No. 4,45-49.

Maureen, T. (2000). Constructivism, Instructional Design, and Technology : Implications for Transforming Distance Learning. Educational Technology & Society, 3(2), 50-60.

Mayer, R. E. (1989). The Psychology of How Novices Learn Computer Programming. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 129-160, Hillside, N.J. Erlbaum.

Mellar, H., & Bliss, J. (1994). Introduction: Modeling and Education. In H. Mellar, J. Bliss, R. Booham, J. Ogborn, C. Tompsett (Eds), Learning with Artificial Words: Computer Based Modeling in the Curriculum, 1-8, London: The Falmer Press.

Nardi, B.A. (1996). Studying context: A comparison of activity theory, situated action models, and distributed cognition. In B.A. Nardi (Ed.), *Context and consciousness: Activity theory and human-computer interaction*, Cambridge, MA: MIT Press.

Pattis, R.E. , Roberts, J. & Stehlic, M. (1995). Karel-The Robot , A Gentle Introduction to the Art of Programming. 2nd ed., New York, Wiley.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. (1989). Conditions of Learning in Novice Programmers. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 261-280, Hillside, N.J. Erlbaum.

Putnam, R. T., Sleeman, D., Baxter, J.A. & Kuspa L.K. (1989). A Summary of Misconceptions of High-School BASIC Programmers. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 301-314, Hillside, N.J. Erlbaum.

Samurcay, R. (1989). The Concept of Variable in Programming: Its Meaning and Use in Problem Solving by Novice Programmers. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 161-178, Hillside, N.J. Erlbaum.

Sangwan, R. S., Korsh, J. F. & LaFollete, P. S. (1998). A system for programming visualization in the classroom. ACM, SIGCSE '98 , Atlanta GA USA, 272-276.

Satratzemi, M., Dagdilelis, V. & Evaggelidis, G. (2002). An Alternating Approach of Teaching Programming in The Secondary School. In Proceedings of 3rd Panhellenic Conference with International Participation, 'Information & Communication Technologies in Education', 289-298, Phodes, Greece.

Skinner, B. F. (1968). The Technology of Teaching, New York : Appleton, 1968.

Soloway, E. & Spohrer, J. C. (1989a). Studying the novice programmer. Hillside, N.J. Erlbaum.

Spohrer, J. C. & Soloway, E. (1989b). Novice Mistakes: Are the Folk Wisdoms Correct?. In Soloway & Spohrer (Eds), 'Studying the Novice Programmer', 401-416, Hillside, N.J. Erlbaum.

Vlahogiannis, Kekatos, Miatidis, Kordaki, M. & Houstis, E. (2001). A multi-representational environment for the Learning of the Bubble Sorting Algorithm. In Proceedings of Panhellenic Conference with International Participation, 'New Technologies in Education and in Distance Education, 481-495, Rethymnon, Greece.

Winslow, L.E. (1996). Programming Pedagogy. SIGCSE Bulletin*,* Vol. 28, No. 3, 17-22.

Zikouli, K., Kordaki, M. & Houstis, E. (2003). A Multi-representational Environment for Learning Programming and C. *3rd IEEE International Conference on Advanced Learning Technologies*, (pp. 459), July, 9-11, Athens, Greece, 2003.