A Learning Research Informed Design and Evaluation of a Web-enhanced Object Oriented Programming Seminar

Stavroula C. Georgantaki
Symeon D. Retalis
Department of Technology Education and Digital Systems
University of Piraeus
80 Karaoli & Dimitriou, 185 34 Piraeus, Greece
rgeo@unipi.gr retal@unipi.gr

ABSTRACT

"Object-Oriented Programming" subject is included in the ACM Curriculum Guidelines for Undergraduate and Graduate Degree Programs in Computer Science as well as in Curriculum for K-12 Computer Science. In a few research studies learning problems and difficulties have been recorded, and therefore, specific pedagogical guidelines and educational tools have been proposed which aim at better supporting the instructional process of Object-Oriented Programming. This paper presents an empirical pilot study of a seminar related to the basic principles-concepts of Object-Oriented Programming. The seminar was at undergraduate educational level using the Java language and web technologies. Its instructional approach was based on selected best instructional practices (either in the form of guidelines or design patterns) already published in the literature. The fundamental aim of the present study was to investigate the factors that might affect the learning effectiveness of a web-enhanced instructional process of the Object-Oriented Programming subject.

Keywords: Object-Oriented Programming, Didactical guidelines, Design patterns, Didactical problems, Instructional approach, Web technologies, Programming environments

1. MOTIVATION

The subject of Object-Oriented Programming (OOP) has been introduced in the computing curricula of the universities during the last years. Several studies have shown that students face various learning difficulties with OOP (Fjuk, Karahasanovic, and Kaasbøll, 2006). These difficulties relate to the comprehension of object-oriented (OO) concepts and the relation between these concepts (Teif and Hazzan, 2004; Ragonis and Ben-Ari, 2002), the misconceptions constructed by students (Holland, Griffiths, and Woodman, 1997; Ragonis and Ben-Ari, 2005b; Fleury, 2000), the perception about OO principles such as encapsulation and reuse (Fleury, 2001) and about dynamics aspects of OO programs (Ragonis and Ben-Ari, 2005a) and so on. Students undoubtedly confront a lot of barriers that they need to overcome simultaneously such as realize new concepts (object, class, attributes, encapsulation, inheritance, polymorphism), apply these concepts in practice by writing software applications using an OO programming language, etc. (Schulte and Niere, 2002). Students who have already been taught the procedural paradigm of programming face additional learning problems which are related to the "paradigm shift", i.e. the transition to the new OO programming paradigm (Luker, 1994).

Various approaches have been proposed for augmenting the learning effectiveness of OOP instructional practices which try to cover the topics recommended by the IEEE/ACM Computing Curricula 2001 (CC2001, 2001). It is also highly recommended to utilize in the instructional process web technologies and educational programming environments (Brusilovsky et al., 1994; Brusilovsky et al., 1997; Gill, 2004). These environments are either programming microworlds such as Karel J. Robot (Bergin et al., 2004), Jeroo (Sanders and Dorn, 2003), JkarelRobot (Buck and Stucki, 2000b), objectKarel (Xinogalos, Satratzemi, and Dagdilelis, 2006; Xinogalos and Satratzemi, 2002), Alice (Cooper, Dann, and Pausch, 2003), or integrated programming environments with educational features like BlueJ (Kölling et al., 2003).

These environments focus on introductory OO concepts, but they hide the details of the programming language as well as the notion of developing programs "from scratch". It has been reported that students upon completion of this introduction still face difficulties in using an actual programming environment, in smoothly transiting to the use of such environments (Kölling et al., 2003; Xinogalos, Satratzemi, and Dagdilelis, 2006) and of course in the acquisition of skills of writing bigger and more complete programs using an OO programming language, such as C++, VB.NET, Java, etc. (Ragonis and Ben-Ari, 2005b; Benander, Benander, and Sang, 2004).

In general, a large body of work has been published on topics such as the didactics of OOP and the instructional

methods in OOP (Fjuk, Karahasanovic, and Kaasbøll, 2006). This paper presents an empirical pilot study of a web-enhanced 9-week seminar on OOP for undergraduate students. The seminar was designed based on some best practices of didactics and pedagogy in OOP (e.g. the adoption of "objects-first" strategy (Kölling and Rosenberg, 2001), the gradual explanation of concepts from simple to higher level ones (Bennedsen and Caspersen, 2004), etc.). Our goal was to investigate which factors might affect the learning effectiveness of learning OOP via web-enhanced technologies. Learning effectiveness is conceptualized as being related to a multiple measurement index consisting of cognitive and attitudinal outcomes (e.g. students' comprehension of basic OO concepts-principles, acquisition of OOP programming skills, self-estimation about their knowledge level in OOP and their feelings and attitudes towards OOP) (Psaromiligkos and Retalis, 2003).

The 9-week seminar followed a blended learning approach which contained face to face lectures as well as web-enhanced learning activities offered via the Moodle Learning Management System (LMS) (http://www.moodle.org) that incorporated learning material chunks structured according to the sections recommended by the IEEE/ACM Computing Curricula, students' assignments and a complete well documented case study. Special emphasis was given on encouraging the asynchronous discussions between students and teacher in the sense of a community of practitioners (Lave and Wenger, 1991).

In this paper, we present the overall design philosophy of this seminar along with the results of a systematic evaluation study of its learning effectiveness that we performed. The evaluation study was based on "pre" and "post-test" questionnaires along with in depth focus group interviews of the students (since they were just 13).

The structure of this paper is as follows: We first describe the instructional approach, i.e. philosophy, material, tools, delivery mode of the 9-week seminar. Later, we will present the conceptual framework of the evaluation study and its results. Finally some concluding remarks about the didactics and pedagogy of OOP teaching using Java as programming language and future plans will be given.

2. THE EMPIRICAL STUDY

2.1 Instructional philosophy

As in most introductory courses/seminars which deal with OOP, the designed seminar had as learning objectives that upon its completion students should be able to:

- describe the principles of object-oriented programming and design,
- analyze problems from an object-oriented perspective and demonstrate object-oriented problem solving techniques,
- create OO designs which will be ready for coding,
- implement, test and debug a pilot OO project utilising programming environments.

Our ultimate goal was to make students effectively acquire certain programming behaviors (i.e. according to the OO paradigm). To achieve this goal, we designed learning activities according to the social cognitive model of sequential skill acquisition of Zimmerman and Kitsantas (Zimmerman and Kitsantas, 1999). This model suggests that learners' acquisition of new skills occurs via four sequential levels: observation, emulation, self-control, and self-regulation.

The *observation level* of skill is mainly characterized by modeling. The student is forming through observation, a mental model of the activity from the other person's actions, hearing descriptions, observing consequences. At the *emulation level*, the student mimics the experiences occurred in the first level and uses feedback from teacher and peers to refine his performance and to develop standards of correct performance that are essential for higher levels of learning. In the stage of *self-control*, the learner compares his practice efforts with personal standards acquired previously from a model's performance. At the final level of *self-regulation*, students learn to adapt their performance to changes in internal and external conditions. Students are mainly interested in performance outcomes.

The various learning activities either face to face or web-enhanced incorporated into our seminar aimed to:

- introduce students gradually to the OO concepts and how they can be applied in practice within OO software applications,
- motivate students to correspond to OO programming challenges (simple ones at the beginning and more difficult at a later point),
- build learning communities in such a way so that all members could help each other in answering questions, etc.,
- scaffold learning up to the autonomous learning,
- ensure the premises for formative performance evaluation,
- support the curriculum-embedded assessment.

2.2 Adoption of best practices in teaching OOP

As already mentioned, when designing this seminar we made an extensive literature review in order to collect effective principles (best practices) in teaching OOP. These practices have been recorded either as short guidelines or as design patterns. Most of these practices are OO language independent, while few of them focus on teaching OOP using Java. The practices adopted in our seminar are the following:

- Deal with the important OO concepts from the beginning. This constitutes the "objects-first" strategy (Kölling and Rosenberg, 2001; Bergin, 2006b, pattern "Early Bird").
- Use a concept-driven approach that emphasizes the role of OO concepts in program development (Hadjerrouit, 1998; Hadjerrouit, 1999) and do not focus on the programming language's details. Introduce concepts gradually from simple to higher level ones (Bennedsen and Caspersen, 2004).
- Pay special attention to the quality of the designed learning material (examples, exercises) in order for students to be benefited while using and studying it (Kölling and Rosenberg, 2001; Buck and Stucki, 2000a; Fleury, 2000; Bergin, 2006a, pattern "Quality is Job One").
- Design appropriate learning activities for active construction of knowledge by the students

(Hadjerrouit, 1999; Fleury, 2000; Fleury, 2001; Bergin, 2006a, pattern "Active Student").
- Use representative examples to start a new topic (Fricke and Voelter, 2000, pattern "Relevant Examples").
- The course notes should not attempt to explain every single detail of an OO language, instead the focus should be on concepts, terminology and usage (Ben-Ari, 2004).
- Follow the guidelines like "Don't start with a blank screen", and "Read code" (Kölling and Rosenberg, 2001; Bergin, 2006a, pattern "Read before Write"), in order students to become gradually able to develop programs. Thus, focus on reading and debugging activities (Fleury, 2000) and on an apprenticeship model of learning (Astrachan and Reed, 1995).
- Use "case studies" for the application of OO principles (Linn and Clancy, 1992; Fleury, 2001).
- Do not defer for a long the presentation of a "class-program" with main method (Ragonis and Ben-Ari, 2005a).
- Integrate appropriately the utilization of an actual programming environment for handling the difficulty in using such environments (Kölling et al., 2003; Xinogalos, Satratzemi, and Dagdilelis, 2006).
- Use technology to keep in touch with students (Bergin, 2006a, pattern "24 by 7").
- Provide individual help to students (Fleury, 2000; Bergin, 2006a, pattern "Differentiated Feedback").
- Notice students' mistakes and difficulties and consider them in course's improvement (Anthony, 1996, pattern "Pitfall Diagnosis and Prevention").
- Review and improve the course after each application (Anthony, 1996, pattern "Iterative Course Development").

2.3 Didactic decisions concerning the seminar's syllabus

The seminar's syllabus comprised of the following seven (7) didactic units so as to be aligned with the Computing Curricula Recommendations of IEEE/ACM (CC2001, 2001):

(i) Introduction to the Object-Oriented Programming philosophy.
(ii) Object, Class, Attributes, Methods, diagrammatic presentation.
(iii) Abstraction, Encapsulation – information-hiding – Separation of behaviour and implementation.
(iv) Objects' creation – Memory allocation – Constructor – References to objects.
(v) Static variables and methods.
(vi) Methods overloading.
(vii) Class hierarchies – Inheritance – Polymorphism – Methods overriding – Dynamic Binding.

We also took into account the already known learning difficulties and we designed carefully the learning material so as to help students avoid misconceptions (Holland, Griffiths, and Woodman, 1997; Teif and Hazzan, 2004; Anthony, 1996, pattern "Pitfall Diagnosis and Prevention").

- More specifically, the following decisions have been made:
  - Attention was paid for the existence of specially designed examples and exercises that aimed at avoiding misconceptions and misconstructions of specific topics. This means that: (i) concrete examples of classes with their attributes and methods and objects derived from these classes have been designed for distinguishing class and object concepts and comprehending the "abstract" concept of class and the "concrete" concept of object, (ii) in all the examples and exercises, classes have more than one attribute (instance variable) each of different type (int, String, double, etc.) in order to avoid the confusion between object/variable (Holland, Griffiths, and Woodman, 1997), (iii) special care has been taken to present classes with an appropriate design model, i.e. mainly with private attributes and public methods and also to work with teaching examples and exercises having more than one instances from a class in order to eliminate the confusion between object/class (Holland, Griffiths, and Woodman, 1997).
  - To help students overcome the difficulty in understanding the constructor method and the instantiation (Ragonis and Ben-Ari, 2002), we designed and presented examples with all possible cases of the constructor method definition (by default and explicitly declared) and for the initialization of attributes (by default constructor without or with initialization of attributes in their declaration, constructor explicitly declared in class giving constant values to attributes or giving values using parameters).
  - To help students better understand what happens in memory during program execution (Milne and Rowe, 2002), we took measures in the unit that contains the internal – invisible operations in OOP (memory allocation during objects' creation, during reference types' definition, during assignment and de-assignment of reference types with and from objects, during objects' destruction and also during the implicit calling of constructor method by operator "new"). We integrated an attempt of dynamic visualization (Ben-Ari, Ragonis, and Ben-Bassat Levy, 2002) of all these subjects in question, using video of simulated program execution and a few carefully selected program statements. Besides this measure, we used the static graphical presentation of memory situation as well.

2.4 Web-enhanced learning environment

The seminar consisted of face to face lecturing as well as web-enhanced learning activities. Lectures occurred twice per week. Each lecture lasted two hours. Lecturing followed a live coding approach (Hyland and Clynch, 2002). For supporting the ex-cathedra teaching method, an on-line learning environment was set up. Students could access the Moodle LMS to get the study guide, the online learning material, the lecture slides as video presentations and other informative material such as tools' installation and usage manuals links to other resources on Java language, etc. Moodle was also used as a medium for asynchronous

discussions, as well as for collecting and grading students' assignments.

Figure 1 portrays the delivery mode, the utilized tools and the characteristics of the learning material for the proposed approach.

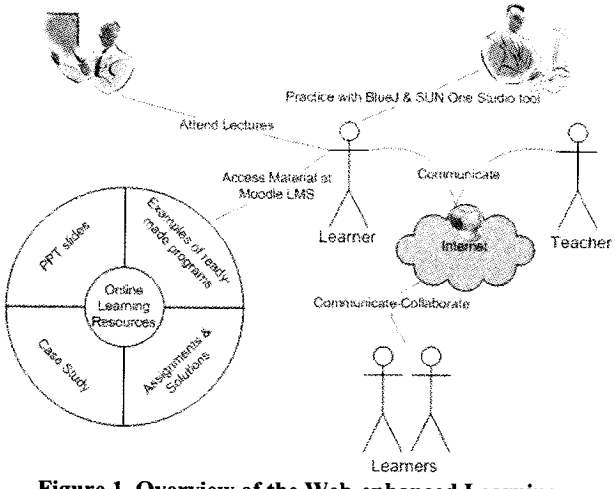

Figure 1. Overview of the Web-enhanced Learning Environment

The online learning material was structured as course notes about the fundamental OO concepts-principles and not as a Java language's textbook, assuring the "concepts driven" philosophy of our approach. Each unit consisted of learning objectives to inform students about the concepts-principles covered and what was expected from them to be achieved in terms of concepts' consolidation and their application in programs, using the BlueJ educational environment and the SUN One Studio professional programming development tool. Source code files, as well as the implemented BlueJ projects' source code files of examples and exercises, were also included in the material. Figure 2 shows a screenshot of the learning material that concerns the memory allocation. The online learning material was delivered as an IMS content package (IMS 2003b).

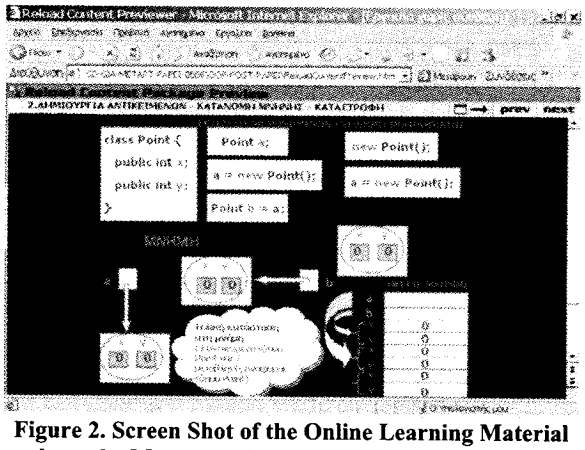

Figure 2. Screen Shot of the Online Learning Material about the Memory Allocation, Packaged Using IMS Content Package Specification

Each unit was also accompanied by a carefully designed *set of questions and exercises* that needed to be solved and handed in by students within pre-determined deadlines. These exercises were used for assessing students' knowledge level and skills according to the learning objectives of each unit.

An important learning asset was the *case study*. The requirements specification, the design decisions, the documentation, the testing and the source code of a simple application were given to the students. With this case study, the basic Object-Oriented Programming principles were applied and demonstrated (details about the case study can be found in the appendix of this paper). The importance of the educational use of case studies has been well documented in the past (Linn and Clancy, 1992) and also teaching recommendations include the existence of case studies in order to present a total picture about program organization (Fleury, 2001).

Care had also been taken in order to ensure that the source code of given examples was well written and well commented. In this way students could be benefited from reading this code (Kölling and Rosenberg, 2001) and could access well-designed object-oriented programs from the early start of the course (Buck and Stucki, 2000a).

For better consolidating the OOP concepts, students had to perform learning activities using two (2) programming tools:

(i) *The BlueJ Educational Programming Environment* (Kölling et al., 2003; Barnes and Kölling, 2005). BlueJ was chosen because of its full support of the "objects-first" approach, its "simplicity" in use, its "visualization" capability of objects and classes and its potential to "interact" with classes creating objects and with objects invoking methods and "inspecting" their state.

(ii) *The Integrated Development Environment in Java, SUN One Studio.* This environment was chosen so as to help students become competent in using a professional software development environment. We also aimed at creating a "full picture" about Object-Oriented programs and their dynamic flow by the students, using programs' code, which make use of classes, create objects and invoke methods. Since a professional software engineer or computer scientist should be able to utilize professional tools, it is necessary for the students to get used to such tools (Kölling et al., 2003).

Students had also access to a great number of well documented Java programs (running examples in BlueJ or SUN One Studio), with which they could experiment so as to better understand concepts and to familiarize themselves with coding themes, such as compilation and execution errors. In parallel students were asked to solve few exercises of gradually increasing difficulty.

3. THE EVALUATION STUDY OF THE SEMINAR

The seminar was organized for undergraduate students of all semesters during the spring semester of 2005, in the Department of Technology Education and Digital Systems of

the University of Piraeus. The primary objective of this department is to provide students with the knowledge and practical experience in various fields of net-centric systems and their development methodologies and environments. In total, the seminar lasted 9 weeks comprised by 17 two-hour lectures, twice per week. Attendance was not obligatory. Actually, 20 students registered for the seminar. Their age ranged from 18 to 23. During the first 12 lectures (out of 17) almost all the students attended, but during the rest 5 lectures, about twelve (12) students attended. The main reasons for the decrease of attendance was the heavy obligations of other compulsory courses and/or scheduling conflicts and the fact that students had eventually to focus on studying for the forthcoming examination period.

All participants had already been taught the procedural programming paradigm with the C programming language and were highly interested in learning OOP, since that knowledge would be helpful in accomplishing the learning goals of other courses in the curriculum. The students' motivation was to enhance their knowledge and skills in OOP. Only three students attended the seminar in order to get better prepared for passing the course's exams.

3.1 Instruments for Data Collection – Participants

The evaluation study was performed by analyzing students' assignments and students' performance in the final OOP course examination, studying the journal kept by the lecturer and analyzing the students' opinion about the learning effectiveness of the seminar. The latter was collected with the aid of a *"pre-test"* and a *"post-test"* questionnaire electronically submitted by the students and with focus group interviews.

The "pre-test" questionnaire was filled out during the first days of the seminar and not later than the first week. It consisted of 46 questions. Students filled out the questionnaire after listening about the seminar's instructional philosophy. The questions concerned with "demographic data", background in programming, self-estimation about programming, learning styles, attitudes towards OOP. Additionally some questions pertained to the identification of their expectations from the seminar, its instructional philosophy and its delivery mode.

The "post-test" questionnaire was filled out immediately after the end of the seminar and consisted of 41 questions. Some questions of the "pre-test" replicated in the "post-test" in order to measure the seminar's effect. However, this questionnaire mainly consisted of a wide number of closed-end questions that were used to evaluate the contribution of various factors to the seminar's effectiveness, such as the type and quality of the various learning resources, the usability of the utilized programming tools, the type and relevance of seminar's learning activities, etc. The answers in closed-end questions were measured in a five-point Likert-type.

It also included a section with a number of open-ended questions to supplement the quantitative data. The open-ended section related to students' likes and dislikes towards the learning material, the deficiencies concerning the material and the approach and suggestions for improving either of them.

The total number of undergraduate students who responded to the evaluation study were 13 (out of 20), 8 women and 5 men, originated from all semesters (2nd, 4th, 6th, 8th semester of undergraduate studies).

The majority of the students (61.5%) utilized only the seminar's resources and this "adds more value" to their evaluation and comments, as their learning was not influenced by other sources.

3.2 Data analysis and Findings

This 9-week seminar seemed to be successful. Three students participated in a final examination of the course about OOP. One student succeeded full-marks (10/10), another one a "very well" grade (8/10) and a third student got a pass (6/10).

Students stated that their knowledge level about OOP was much better after the seminar. More specifically, the mean values of the students answers to the question "Based on what you have been taught till now, how well do you think that you have learned OOP?" increased from 2.1 (at the "pre-test") to 3.8 at the "post-test" (Answer's coding scheme: 4 = I learned well, ..., 1 = I did not learn at all). Of course, these statistics are just descriptive and show some trend since the number of students who participated in the evaluation study was relatively small.

Those statements were accurate since we made a cross-reference of students' opinion to the quality of their assignments. The analysis of the students' assignments also revealed that students acquired solid knowledge about the OO concepts and became able to solve problems applying the OO philosophy. Several didactical problems had been eliminated such as: understanding the difference between the concepts of class and object, understanding the meaning of the constructor method and its place within the class (Ragonis and Ben-Ari, 2002), perception of the encapsulation principle (Fleury, 2001), etc. However, some difficulties remained such as the confusion between attributes of an object and its parts (Teif and Hazzan, 2004), the correct identification of attributes for the classes, the comprehension of the composed classes.

Furthermore, as shown in Table 1, the students' feelings towards OOP were more positive after the seminar and the degree of students' satisfaction was high. The seminar's instructional approach offered great help to the students in overcoming their difficulties and fears concerning the OOP. While interviewing students, we found out that their fears about OOP remained almost the same. They stated that, although OOP is a much simpler subject than previously believed (i.e. before attending the seminar); it demands a lot of effort to become a competent OO programmer. A possible explanation for these beliefs is that students performed a lot of learning activities in OOP during the seminar and thus they formulated a clearer opinion about this unquestionably demanding subject.

As already discussed, with this evaluation study we wanted to check which factors contribute to the acquisition of knowledge and skills in OOP. Thus, we asked students before and after the seminar to estimate the importance of various factors to the enhancement of knowledge and skills in OOP. The mean values of their answers are shown in Table 2.

Question	Mean ("pre-test")	Mean ("post-test")
Have the reasons that brought you to attend the seminar been satisfied?	-	4.7
Do you appreciate the cognitive subject of OOP?	2.3	4.1
Do you have difficulties with OOP?	3.5	3.3
The seminar aided me to get over the difficulties that I was facing concerning OOP.	-	4.2
Do you fear OOP?	2.5	2.5
Do you agree that OOP is a much simpler subject than previously believed?	-	4.0
The seminar aided me to get over fears that I was feeling concerning OOP.	-	4.0

Answers' coding: 5= Very much, ..., 1= Not at all

Table 1. Students' Feelings and Attitudes Towards OOP

Question: Estimate the importance of the following factors to the enhancement of knowledge and skills	Mean ("pre-test")	Mean ("post-test")
The assignments.	4.7	4.9
The lectures' slides uploaded on Moodle.	4.6	4.8
The online course notes.	4.7	4.7
The case study.	4.2	4.6
The hands-on practice with programming environments/tools.	4.5	4.6
The discussions on the subject matter with teachers via Moodle.	4.4	4.5
The discussions on the subject matter with other students via Moodle environment.	3.8	3.8
The comments/corrections of teachers to submitted assignments.	-	4.9
Examples with Java source code files included in the material.	-	4.5

Answers' coding: 5= Absolutely important, ..., 1= Totally unimportant

Table 2. Contribution of Various Factors to the Enhancement of Knowledge and Skills in OOP

As can be seen from Table 2, the comments/corrections made by the teachers on the students' assignments had been rated as the most important factor. This factor in combination with the one about "teacher-student communication via Moodle" indicates how beneficial the interaction with the teachers was for the students. On the contrary, the discussions among students had not been highly appreciated. This may be explained by the fact that the students had not been used to such collaborative working schemes. Perhaps, the fact that students came from different semesters with different expectations prevented the creation of a "learning community".

Students estimated that the assignments contributed significantly in increasing their knowledge and skills, and therefore their appropriateness and their successful design was emerged.

The case study also contributed greatly to the enhancement of knowledge and skills, although students initially estimated the contribution of this factor lower compared to the others.

As predicted, the students highly appreciated the easy and flexible access to the presentation slides, the examples in Java and the on-line reference material via the Moodle LMS. Moreover, they considered pedagogically effective their involvement in learning activities utilizing the educational tools. The visualization of classes and objects that BlueJ supports had been highly appreciated by the students. The environment considerably helped in the clarification of various concepts such as class, object, method, inheritance and operations such as instantiation and method invocation.

Characteristically, two students said that: "BlueJ helped me because I could see what happened when I was writing the code and how classes were interrelated within the program" and "BlueJ aided me considerably to realize what happens inside a program, how classes are related to each other and to objects".

Students were asked in the "post-test" questionnaire to state the most valuable features/reasons for using the BlueJ environment such as ease-of-use, interactivity, etc. The collected results are depicted in the Table 3. Students particularly appreciated the simplicity of the environment. As expected, the strengths of the BlueJ tool had been its interactivity, visualization and simplicity attributes (Kölling and Rosenberg, 2001; Kölling et al., 2003).

Question: Which features of the BlueJ environment made you use it for performing learning tasks? Select those that you think.	Number of students (max 13)
Ease-of-use of the BlueJ environment.	11/13
Classes and objects are represented graphically.	10/13
Users can interact directly with classes and objects.	8/13
Easy instantiation and visual representation of objects.	9/13
Users can easily invoke methods and check their function (you can pass parameters and get the returning result in an easy way) without the need to write a single line of code.	9/13
Users can easily inspect attributes' values of the objects.	7/13
It's easy to compile programs. Compile-time errors are displayed directly in the source editor by highlighting the error line.	9/13
The environment shows the "implementation" aspect and "interface" aspect of classes.	6/13
The environment incorporates an easy-to-use debugger.	4/13
You can get immediate feedback after having experimented with it.	5/13

Table 3. Students' Opinion about the Most Valuable Features/Reasons for Using the BlueJ Environment

Of course, students had some dislikes towards the seminar's instructional approach. Few students stated the need for practical lab sessions with the physical presence of a tutor. Also, some students said that the sections (a) Encapsulation – information-hiding – Separation of behavior and implementation and (b) Polymorphism, need improvement.

The observations during the lectures and some students' answers to interviews, made us decide some revisions of the instructional philosophy. More specifically, the order of presentation of some didactic units should change such as swap between the third (abstraction-encapsulation) and the fourth topic (instantiation-constructor), because constructors should come before any other concept and right after the introduction of main OOP concepts, as well as a change for the assignments that students have to submit after the end of each didactic unit should happen. Each assignment should be a part or a piece of a medium-size software development project that could be gradually developed step by step. Thus, the students will more easily understand how the various OO concepts are interrelated.

Having made the aforementioned changes based on the evaluation findings and students' feedback, we performed in spring semester of 2006 a new evaluation study with a considerably larger group (66 students).

More specifically, 66 students voluntarily attended a 9-week course on OOP which followed the blended learning approach already described. The students' profile was the same to the one of the students who participated to the first experiment. Students originated from all semesters of undergraduate studies in the Department of Technology Education and Digital Systems at the University of Piraeus. Apart from having assessed the student's assignments students had also to take exams on OOP at the end of the course. Six students did not appear due to unspecified personal reasons. 45 out of 60 students passed the exams (i.e. 75%). The average score of students' grades was 6.28 with standard deviation of 3.50. The success rate is impressive when compared to the success rate of the same course being taught using traditional methods in the same Department. The success rate at the same academic year was 27.38% (23 out of 84 students). The course's examination included questions and exercises that aimed to:

- examine students' comprehension about OO concepts and their application in Java programs,
- ask students analyze problems in object-oriented terms and create short programs in Java language in order to solve these problems,
- test if students were able to write Java programs from a given class diagram design.

From the assessment of students' assignments and final examination papers as well as the analysis and interpretation of the students' answers to "pre" and "post-test" questionnaires, we can mention that:

- Students stated that their knowledge level about OOP was much better after the new course and appreciated the importance of OOP for their studies and their professional future.
- The same instructional design factors contributed to the enhancement of students' knowledge and skills in OOP at almost the same high level. For example, Table 4 shows students answers before and after the seminar. Students' opinions about the importance of various factors to the enhancement of their knowledge and skills in OOP are quite close to ones given by the first small group of students.
- The discussions on the subject matter among students via the Moodle environment had been considered an important factor to their enhancement of knowledge and skills. This new finding can be justified by the fact that emphasis was given on creating an active online learning community during the new seminar.

Question: Estimate the importance of the following factors to the enhancement of knowledge and skills.	**Mean ("pre-test")**	**Mean ("post-test")**
The case study.	4.50	4.39
The discussions on the subject matter with other students via Moodle environment.	4.22	4.10
The discussions on the subject matter with teachers via Moodle	4.56	4.28
The assignments	4.39	4.28
The lectures' slides uploaded on Moodle.	4.30	4.32
The hands-on practice with programming environments/tools.	4.50	-
The online course notes.	4.56	4.44
Attending the face to face meetings of the seminar and the lectures during these meetings.	-	3.89
The comments/corrections of teachers to submitted assignments.	-	4.10
Examples with Java source code files included in the material.	-	4.28

Answers' coding: 5= Absolutely important, ..., 1= Totally unimportant

Table 4. Contribution of Various Factors to the Enhancement of Knowledge and Skills in OOP

4. CONCLUDING REMARKS

In this paper, we presented our efforts for building up an undergraduate course on OOP whose instructional approach was based on didactical guidelines and design patterns that have been proposed in previous studies. We designed the learning material, i) emphasizing on the dissimilarity of OO programming philosophy, ii) focusing on OO concepts-principles with the use of representative examples, iii) providing a great number of well documented exemplar programs in Java and complete "class-programs" as case examples, iv) asking students to perform learning activities and submit assignments using the BlueJ educational tool as well as the SUN One Studio professional programming environment. We also offered to students an on-line learning environment for the submission and grading of assignments, the communication with teachers and their peers, as well as the easy access to the various learning resources.

We performed two evaluation case studies which showed that the proposed approach is effective. Students'

achievements in submitted assignments as well as their performance in final exams, confirmed that several didactical problems have been solved successfully. Via an analysis of teacher's journal as well as students' answers to questionnaires and interviews, we found out that most of the design decisions made when setting up the course had been appreciated by the students and contributed to its learning effectiveness.

Concluding, we plan to formalize the instructional approach of this seminar using the IMS Learning Design specification (IMS 2003a), thus augmenting its re-usability. It's also under our consideration to enrich our instructional approach by giving more emphasis on the modeling perspective of OOP, since a model is itself an abstraction of something for the purpose of understanding it (Groven, Hegna, and Smørdal, 2003; Berge et al., 2003).

5. ACKNOWLEDGMENT

We would like to thank all students who participated to the evaluation study for giving us valuable feedback.

6. REFERENCES

Anthony, D. L. G. (1996), Patterns for Classroom Education, Pattern Languages of Programming 2, Vlissides, Coplien, Kerth (editors), Addison Wesley, 1996, pp 391, retrieved October 1, 2006, from http://ianchaiwriting.50megs.com/classroom-ed.html

Astrachan, O., and Reed, D. (1995), "AAA and CS1: The Applied Apprenticeship Approach to CS1." ACM SIGCSE Bulletin, 27(1), 1-5.

Barnes, D. J., and Kölling, M. (2005), Objects First with Java, A Practical Introduction using BlueJ. 2nd edition, Prentice Hall.

Benander, A., Benander, B., and Sang, Janche (2004), "Factors related to the difficulty of learning to program in Java—an empirical study of non-novice programmers." Information and Software Technology, 46(2), 99-107.

Bennedsen, J., and Caspersen, M. (2004), "Teaching Object-Oriented Programming – Towards Teaching a Systematic Programming Process." Paper presented at 18th European Conference on Object-Oriented Programming, 8th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, Oslo, Norway.

Ben-Ari, M., Ragonis, N., and Ben-Bassat Levy, R. (2002), "A Vision of Visualization in Teaching Object-Oriented Programming." Proceedings of the 2nd Program Visualization Workshop, 83-89, HornstrupCentret, Denmark.

Ben-Ari, M. (2004), "Situated Learning in Computer Science Education." Computer Science Education, 14(2), 85-100.

Berge, O., Borge, R. E., Fjuk, A., Kaasbøll, J. and Samuelsen, T. (2003), "Learning Object-Oriented Programming." Norsk Informatikkonferanse.

Bergin, J., Stehlik, M., Roberts, J., and Pattis, R. (2004), Karel J. Robot a gentle introduction to the art of object oriented programming in Java. Published manuscript, retrieved July 13, 2006, from http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html

Bergin, J. (2006a), Some Pedagogical Patterns, retrieved October 1, 2006, from http://csis.pace.edu/~bergin/patterns/fewpedpats.html

Bergin, J. (2006b), Fourteen Pedagogical Patterns, retrieved October 1, 2006, from http://csis.pace.edu/~bergin/PedPat1.3.html

Brusilovsky, P., Kouchnirenko, A., Miller, P., and Tomek, I. (1994), "Teaching programming to novices: a review of approaches and tools." Proceedings of ED-MEDIA 94-World Conference on Educational Multimedia and Hypermedia, Vancouver, Canada, 25–30 June, 103-110.

Brusilovsky, P., Calabrese, E., Hvorecky, Y., Kouchnirenko, A. and Miller, P. (1997), "Mini-languages: a way to learn programming principles." Journal of Education and Information Technologies, 2(1), 65-83.

Buck, D., and Stucki, D. (2000a), "Design early considered harmful: Graduated exposure to complexity and structure based on levels of cognitive development." ACM SIGCSE Bulletin, 32(1), 75–79.

Buck, D., and Stucki, D. (2000b), "JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum." ACM SIGCSE Bulletin, 33(1), 16-20.

CC2001 Computing Curricula 2001 (final report) (2001), The Joint Task Force on Computing Curricula (IEEE Computer Society and ACM), December 15, retrieved January 27, 2007, from http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/cc2001.pdf

Cooper, S., Dann, W., and Pausch, R. (2003), "Teaching Objects-first in Introductory Computer Science." ACM SIGCSE'03, 191-195.

Fjuk, A., Karahasanovic, A., and Kaasbøll, J. (2006), Comprehensive Object-Oriented Learning: The Learners Perspective. Informing Science Press, California.

Fleury, A. E. (2000), "Programming in Java: Student-constructed rules." ACM SIGCSE Bulletin, 32(1), 197–201.

Fleury, A. E. (2001), "Encapsulation and reuse as viewed by java students." ACM SIGCSE Bulletin, 33(1), 189–193.

Fricke, A., and Voelter, M. (2000), "Seminars: A Pedagogical Pattern Language About Teaching Seminars Effectively." Proceedings of EuroPLoP 2000, retrieved October 1, 2006, from http://www.voelter.de/publications/seminars.html

Gill, T. Grandon (2004), "Teaching Flowcharting with FlowC." Journal of Information Systems Education, 15(1), 65-78.

Groven, A., Hegna, H., and Smørdal, O. (2003), "OO learning, a modeling approach." Paper presented at 17th European Conference on Object-Oriented Programming, 7th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, July, Darmstadt, Germany.

Hadjerrouit, S. A. (1998), "Constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum." ACM SIGCSE Bulletin, 30(3), 105-107.

Hadjerrouit, S. (1999), "A Constructivist Approach to Object-Oriented Design and Programming." ACM SIGCSE Bulletin, 31(3), 171-174.

Holland, S., Griffiths, R., and Woodman, M. (1997), "Avoiding object misconceptions." ACM SIGCSE Bulletin, 29(1), 131-134.

Hyland, E., and Clynch, G. (2002), "Initial experiences gained and initiatives employed in the teaching of Java programming in the Institute of Technology Tallaght." ACM International Conference Proceeding Series (AICPS), Vol. 25, 101-106.

IMS Learning Design Specification (2003a), retrieved October 11, 2006, from http://www.imsglobal.org/learningdesign/index.cfm

IMS Content Packaging Specification (2003b), retrieved October 11, 2006, from http://www.imsglobal.org/content/packaging/index.cfm

Kölling, M., and Rosenberg, J. (2001), "Guidelines for Teaching Object Orientation with Java." ACM SIGCSE Bulletin, 33(3), 33-36.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003), "The BlueJ system and its pedagogy." Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, 13(4), 249–268.

Lave, J., and Wenger, E. (1991), Situated learning: Legitimate peripheral participation. Cambridge University Press, Cambridge.

Linn, M., and Clancy, J. (1992), "The case for case studies of programming problems." Communications of the ACM, 35(3), 121–132.

Luker, P.A. (1994), "There's more to oop than syntax!" ACM SIGCSE Bulletin, 26(1), 56-60.

Milne, J., and Rowe, G. (2002), "Difficulties in Learning and Teaching Programming – Views of Students and Tutors." Education and Information Technologies, 7(1), 55-66.

Psaromiligkos, Y., and Retalis, S. (2003), "Re-Evaluating the Effectiveness of a Web-based Learning System: A Comparative Case Study." Journal of Educational Multimedia and Hypermedia, AACE, 12(1), 5-20.

Ragonis, N., and Ben-Ari, M. (2002), "Teaching Constructors: A Difficult Multiple Choice." Paper presented at 16th European Conference on Object-Oriented Programming, 6th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, Malaga, Spain.

Ragonis, N., and Ben-Ari, M. (2005a), "On Understanding the Statics and Dynamics of Object-Oriented Programs." ACM SIGCSE'05, 226-230.

Ragonis, N., and Ben-Ari, M. (2005b), "A Long-Term Investigation of the Comprehension of OOP Concepts by Novices." Computer Science Education, 5(3), 203-221.

Sanders, D., and Dorn, B. (2003), "Jeroo: a tool for introducing object-oriented programming." ACM SIGCSE Bulletin, 35(1), 201–204.

Schulte, C., and Niere, J. (2002), "Thinking in Object Structures: Teaching Modeling in Secondary Schools." Paper presented at 16th European Conference on Object-Oriented Programming, 6th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, Malaga, Spain.

Teif, M., and Hazzan, O. (2004), "Junior High School Students' Perception of Object Oriented Concepts." Paper presented at 18th European Conference on Object-Oriented Programming, 8th Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, Oslo, Norway.

Xinogalos, S., and Satratzemi, M. (2002), "An Integrated Programming Environment for Teaching the Object-Oriented Programming Paradigm." Lecture Notes In Computer Science (LNCS), Vol. 2510, 544-551.

Xinogalos, S. Satratzemi, M., and Vassilios Dagdilelis, V. (2006), "An introduction to object-oriented programming with a didactic microworld: objectKarel." Computers & Education, 47(2), 148-171.

Zimmerman, B. J., and Kitsantas, A. (1999), "Acquiring writing revision skill: Shifting from process to outcome self-regulatory goals." Journal of Educational Psychology, 91(2), 241-250.

AUTHOR BIOGRAPHIES

Stavroula C. Georgantaki graduated from Department of Physics, School of Sciences of the National and Kapodistrian University of Athens, Greece, and received the MSc degree in Electronic Automation, School of Sciences, National and Kapodistrian University of Athens, Greece. She had been employed as software engineer in private organizations. Her current job is teacher of Informatics in Secondary Education. She has taught for over 10 years. She is currently a PhD Candidate at the Department of Technology Education and Digital Systems of the University of Piraeus, Greece. Her main research interests are in the Didactics of Object-Oriented Programming.

Symeon D. Retalis is Associate professor at the Department of Technology Education & Digital Systems, University of Piraeus. He holds a diploma of Electrical and Computer Engineer from the Department of Electrical and Computer Engineering studies, National Technical University of Athens, Greece, an MSc degree in Information Technology-Knowledge Based Systems from the Department of Artificial Intelligence, University of Edinburgh, Scotland, and a PhD diploma from the Department of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research interests lie on the development of web-based learning systems, design of adaptive hypermedia systems, web engineering, and human computer interaction. He has coordinates and participated in various European R & D projects such as WEENET, MAUSE, TELL, ELEN, UNIVERSAL, etc. He serves in the editorial board of Computers in Human Behavior, IEEE Journal of Educational Technology and Society, ACM Computing Reviews, Journal of Information Technology Education. He participates to the ACM Web Engineering special interest group, to the CEN/ISSS learning technologies workshop. He is also director of the CoSy LLab (Computer Supported Learning Engineering Laboratory)[http://cosy.ted.unipi.gr]. His publication list contains more than 70 items.

APPENDIX
SOME ASPECTS OF THE CASE STUDY

This appendix contains highlight some aspects of the case study included in the instructional material. The case study was used for explaining students the analysis, design and implementation process of a small OO application, step by step. The OO application concerned a "street market".

Description of the Problem to Be Solved

In a "Street Market" consumers, walk around the stands that sellers use to exhibit their products and choose products. The consumers have a basket for placing inside it the products and a wallet with money for paying the products. Correspondingly, the sellers have their "cash" with money receipted from sales.
The sellers weigh the chosen products and calculate their total price that the costumers have to pay before placing them in the shopping baskets and the sellers add the paid amount to their "cash". The products of the Market are of two kinds. The first-quality products, which have the marked price and the second-quality ones having a percentage discount.

Design of the Situation Using the Bluej Educational Environment

The first task concerns the creation of a class diagram for the "Street market" application. Students first see the hierarchy of classes and the specification of their characteristics. Class diagrams are shown in the BlueJ environment, as shown in Figure 3.

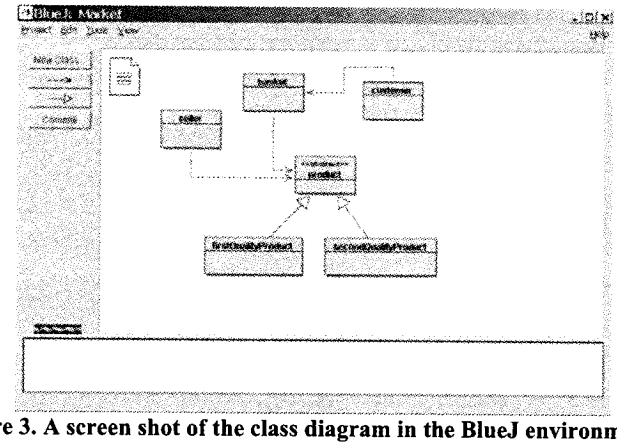

Figure 3. A screen shot of the class diagram in the BlueJ environment

Adding Functionality to Classes Using the Bluej

Students are gradually learning how to specify and add methods to the classes already specified during the previous step. Bodies of the methods added to the class specification are shown below (Figure 4).

```
public void addProduct(product product1) {
  myproduct.add(product1);
 }
 public double weigh(){
  return Math.random()*10;
 }
 public void pay(double amount) {
  amountInWallet -= amount;
 }
 public void receivePayment(double amount) {
  cashAmount += amount;
 }
 public double calculateValue(product product1, double quantity){
  return (product1.price * quantity);
 }
 public double salePrice() {
 return (price * (1 - (discountRate/100)));
 }
```

Figure 4. Some methods' bodies for the "Street market" application

Testing and Debugging Step-By-Step an OO Application Using the Bluej Environment

Students are also been taught how to test and debug an OO application using the BlueJ environment. Figure 5, shows how the application runs where objects are being created and methods have been invoked using the BlueJ interactivity feature.

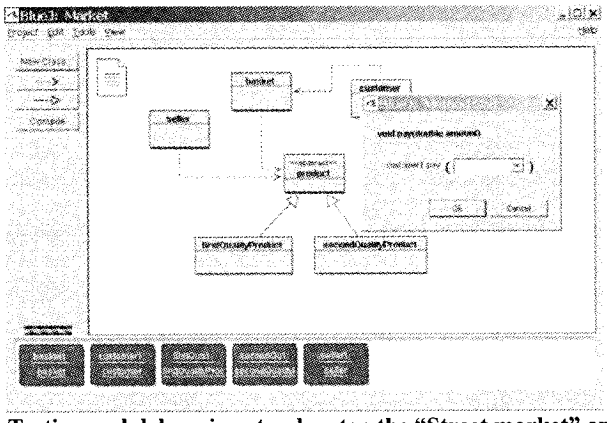

Figure 5. Testing and debugging step-by-step the "Street market" application

Transferring Classes' Code from Bluej to SUN One Studio, Creation of A "Class-Program" With Main Method, and Execution

The last step of the case study deals with showing students how to transfer the OO source code from the BlueJ environment to the SUN One Studio for further enrichment of the application's functionality and documentation. Figure 6 shows a screenshot of the ""Street Market" application source code in the SUN One Studio.

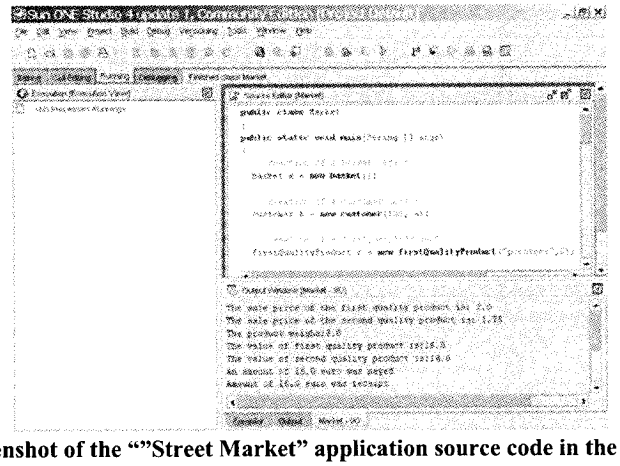

Figure 6. A screenshot of the ""Street Market" application source code in the SUN One Studio

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096