Journal of Information Technology Impact
Vol. 7, No. 2, pp. 111-130, 2007

Using Educational Tools for Teaching Object Oriented Design and Programming

Stavroula Georgantaki [1] Symeon Retalis [2]
University of Piraeus
Greece

Abstract

The development of software systems is a complex process which requires a diverse set of skills and expertise. The Object Oriented programming paradigm has been proven to better organize the inherent complexity of software systems, than the traditional procedural paradigm. Hence, Object Oriented (OO) is becoming the dominant paradigm in the recent years. The software industry is placing increasing emphasis on newer, object-oriented programming languages and tools, such as Java. It is highly interested for software engineers capable to analyze and develop systems using the OO programming paradigm. The early exposure of students to object oriented programming has become increasingly important in both academia and industry. As an effect, after IEEE and ACM recommendations, the object oriented programming approach has established its place in the Computing Science (CS) Curricula of the Universities. Furthermore, a lot of research is being conducted with regards to the effective instructional methods for the Object Oriented design and programming. Such methods propose the utilization of educational tools for helping students overcome learning difficulties. The scope of this paper is to present some of the well known educational tools which can support the instructional process of Object Oriented design and programming. It also aims to show how one of the most widely known educational tool, called BlueJ, has been utilised and evaluated by students within an educational setting. Finally it discusses trends in the development of this tool (and other related ones) which could influence the teaching of Object Oriented design and programming .

Keywords: Educational tools, design, programming, learning difficulties.

Introduction

The software industry is placing increasing emphasis on newer, object-oriented programming (OOP) languages and tools, such as Java. It is highly interested for software engineers capable to analyze and develop systems using itthe OO programming paradigm (Bureau of Labor Statistics, 2006). As a consequence, the OOP paradigm was established fifteen years ago as a teaching subject, mainly, in tertiary education curriculum (Barr et al., 1999; Lewis, 2000; Kölling and Rosenberg, 2001; Berge at al., 2003; Bennedsen and Caspersen, 2004; Ragonis and Ben-Ari,

2005b). The OOP paradigm has proven to be most appropriate than the traditional procedural one, for decomposing the inherently complex synchronous software systems (Booch, 1991; ACM, 2002). Thus, its significance for the education of future software engineers is great. The early exposure to OOP has become increasingly important in both academia and industry (ACM, 2002).

Academics are searching for effective ways to teach OOP. Many related studies have shown that students face various learning difficulties with this paradigm (Holland et al., 1997; Carter and Fowler, 1998; Wiedenbeck and Ramalingam, 1999; Wiedenbeck et al., 1999; Fleury, 2000; Fleury, 2001; Ragonis and Ben-Ari, 2002; Ben-Ari et al., 2002; Milne and Rowe, 2002; Teif and Hazzan, 2004; Ragonis and Ben-Ari, 2005a; Ragonis and Ben-Ari, 2005b; Lahtinen et al., 2005). As a consequence students find it hard to conceptualize the OOP philosophy and its concepts.

Even from the procedural programming era, the utilization of educational tools in the instructional process has been highly recommended for helping students overcome difficulties and better achieve the learning objectives (Brusilovsky et al., 1994; Brusilovsky et al., 1997). These objectives are mainly the consolidation of programming techniques and the development of problem solving skills using programming, rather than learning a specific programming language with its syntactic and semantic details.

Moreover, it has been reported that the utilization of mere educational tools is not the best strategy. Students should also get acquainted with professional programming environments which will utilize when working as professionals (Xinogalos et al., 2006; Kölling et al., 2003). Thus, it is important that students learn to use professional tools during their academic carrier. An appropriate combination of educational and professional tools and a smooth transition from the first ones to the second ones would be an ideal situation.

The scope of this paper is to present the educational tools specifically developed for supporting the instructional process of the Object Oriented Programming paradigm. We will present their characteristics, and make a detailed description of the usage of the most popular educational tool, called BlueJ, in combination with a professional programming environment within authentic educational settings. Findings from an evaluation study about the learning effectiveness of these tools will be discussed.

In the next sections, we first describe a set of the educational tools created to better support the instructional process of OOP. Then, we will make reference to a case study concerning the combined utilization of the BlueJ tool with the professional programming environment, IBM SUNOne. We will present some results derived from the systematic evaluation of this case study. In the end, we will discuss the future development trends of educational tools and their use in educational settings which could influence the way OOP can be taught.

Literature Review of Educational Tools for OOP

Various tools have been developed for teaching OO programming and design. These tools can be identified and presented in seven main categories: programming microworlds, educational programming environments, tools for enhancing the capabilities of programming environments,

software tools supporting the "objects-first" teaching approach, gaming environments, and tools based on specific educational programming languages tools. Details about these tools can be found in white papers about them as well as on-line manuals.

Programming microworlds

Most tools of this category are based on the Karel the Robot microworld (Pattis, 1981), developed for the introduction to Procedural programming using a Pascal-like language and also on the Karel++ which constitute the entrance of Karel to the OO era (Bergin et al., 1997). Some of the widely utilised tools are:

- ***KarelJ.Robot*** (http://csis.pace.edu/~bergin/KarelJava2ed/karelexperimental.html) is a translation of Karel++ microworld to purely Java language. It constitutes its Java-based sibling (Bergin et al., 2003). Recently a book has been published containing examples and exercises for this environment (Bergin et al., 2005).
- ***JKarelRobot*** (http://math.otterbein.edu/JKarelRobot/) (Buck and Stucki, 2001) is a platform and language/paradigm independent microworld (written in Java). It supports Pascal, Java and Lisp style languages, the capability for flowcharts in a reverse direction from the historical usage (conversion of code programs into flowcharts) and also the diagrammatic presentation of programs using control structure diagrams (CSD).
- The ***Jeroo*** (http://www.jeroo.org/) (Sanders and Dorn, 2003) programming microworld uses as metaphor a rare kind of an Australian kangaroo. Some of its design principles, that guided its development, are: the focus on "object", "method" and control structures in Java and C++. Its major advantage is that it uses a single window in which all components are visible at all times, i.e the source code, the metaphor's environment, the message area, the position of metaphor, the runtime behavior.
- ***Alice*** (http://www.alice.org) is a 3D animation environment, developed at Carnegie Mellon University (CMU) in which objects and their behaviors are visualized (Cooper et al., 2000; Cooper et al., 2003; Moskal et al., 2004). Essentially follows the tradition of "Karel microworlds" adding one more dimension. The 3D animation assists in providing stronger object visualization and a flexible, meaningful context for helping students to "see" OO concepts. Three-dimensionality provides a sense of reality for objects. In the 3D world, students may write methods from scratch to make objects perform animated tasks. The animation task provides a meaningful context for understanding classes, objects, methods, and events.
- ***objectKarel*** (http://csis.pace.edu/~bergin/temp/findkarel.html) (Xinogalos et al., 2006) is an environment developed at Department of Applied Informatics of the University of Macedonia, Greece. Important features of the environment are: the incorporated structure editor for developing programs, the possibility of exploratory visualization i.e. the explanatory messages about the semantics of the command being executed offered when students are tracing or executing step-by-step, the incorporated e-lessons with theory and activities provided in Greek and English language, and the possibility to record the students' activities during the development and debugging of a program, enabling the identification of difficulties and misconceptions in order for the teacher to adjust the lesson to students' needs.

Educational Programming Environments

The most well known and widely used representative of the Educational Programming environments is the ***BlueJ*** tool (www.bluej.org). It provides an integrated environment in which the OO software project structure is presented graphically in UML-like diagrams (Kölling et al., 2003). The student can create objects and invoke methods on those objects to illustrate their behavior. Its educational use is referenced also as the "BlueJ approach" (Fjuk et al., 2006, Chapter 6, p. 80). It is a complete Java integrated development environment (IDE), implemented completely in Java. Some of its important features are: its simplicity in use, its "visualization" capability of projects, objects and classes and its potential users to "interact" with classes creating objects and with objects invoking methods and "inspecting" their state, the implementation and interface view for the classes, the easy pass from visual to code environment and easy compilation with integration of error messages and source editor. The BlueJ environment comes with a textbook (Barnes and Kölling, 2005), with a "project driven" presentation of material.

Other educational programming environments which have been used in educational case studies are:

- ***DrJava*** (http://drjava.sourceforge.net/) is a pedagogical programming environment for Java that gently introduces students to writing programs (Allen et al., 2002). It enables the student to develop, test, and debug Java programs in an interactive, incremental way, providing a simple interface based on a "read-eval-print loop" (read, evaluate, print, REPL). Essentially, it is a Java interpreter that allows interactive evaluation of a series of Java statements (Kölling et al., 2003) and interactivity at object level using an interactive window typing expressions and Java statements and seeing immediately the results without the need of a main() method and an external compiler. It does not provide graphical representation for classes and objects.
- ***ProfessorJ*** (http://www.professorj.org/) (Gray and Flatt, 2003; Proulx and Gray, 2006) is a programming environment within DrScheme that provides support for the novice programmer through a Java-like programming language. It presents the student with levels of language's structures (Beginner, Intermediate, Advanced), providing a simplified interface to the Java compiler and virtual machine. Each level is a language's subset. By this way students are gradually introduced in syntactic and semantic details, having an environment that follow course's advance, which is a gradual exposition to more complex structures. All parts of the environment -the syntactical rules, the error messages, the testing environment, and all supporting libraries- respect the Java-like language constraints (Powers et al., 2006). Currently, this tool supports the instructional process at beginner and intermediate levels (http://download.plt-scheme.org/drscheme). The version of the tool for advanced level is currently undergoing formative testings.
- ***JPie*** *(Java Programmer's Interactive Environment)* (http://jpie.cse.wustl.edu/) is an integrated programming environment designed to make the power of OO software development accessible to a wider audience (Goldman, 2003). It represents programs graphically and not textually and enables live construction of Java applications through direct manipulation of graphical representations of programming abstractions and structures (classes, subclasses, type and scope of instance variables, methods, event handler methods etc.). JPie's rests on the notion of a *dynamic class*, whose signature and

implementation can be modified live, even while instances of that class exist in a running program, thereby eliminating the "edit-compile-test" cycle. The environment developed at the Department of Computer Science and Engineering of Washington University in St. Louis, and supports the "concepts-first" curriculum organized around the fundamental programming concepts ("big ideas") (Goldman, 2004).

- ***Javiva*** is a compiler for Java programs for novice students. Using this, students are able to include pre-conditions and post-conditions for functions, and abstraction functions for classes, as stylized comments in Java source code. Javiva processes the comments to incorporate instrumentation into the class files that it produces. When these classes are used at runtime, violation of pre- and post-conditions are automatically detected and reported. In addition, abstract visualizations of the objects that are created and manipulated by classes are automatically generated (Turner and Zachary, 2001). These possibilities, enforce students ability to think abstractly about the components of a program (e.g. what a method does, what a class represents) and thus prevent the trend to conceptualize a program as an unstructured collection of statement and expressions.
- The ***GiniPad*** (http://www.mokabyte.it/ginipad/english.htm) environment offers a window that presents classes, methods, attributes and interfaces of source code in a tree structure. It marks code items with colors and allows the transition from a compilation error to the point of code with the error. It does not provide a debugger and the possibility of step-by-step execution.
- ***JGrasp*** (http://www.eng.auburn.edu/grasp/) (Hendrix et al., 2004) supports the notation of code elements with colors and also the diagrammatic presentation of programs using control structure diagrams (CSD). These diagrams aim at improving the comprehensibility of source code and display the control structures, the program flow and generally each program's section structure, in the space derived from code indentation. It provides code templates for classes, methods and control structures, marks the first compilation error by changing the color, the line code where probably the error occurs and also provides a visual debugger that allows the introduction of break points and the step-by-step program execution. The environment creates UML diagrams and allows students to create objects of any class and interact by invoking methods on them in the workbench area.
- The ***jCreator LE*** (http://www.jcreator.com/index.htm) environment displays in a tree structure the OO software project's files, allows classes to be added using templates, provide a summary for the structure of each class and also the possibility classes to be extended using templates. It does not provide a debugger and the possibility of step-by-step execution.
- The ***D-ChK*** system is an IDE for Java language that allows tighter integration between the theory and the practical aspects by incorporating several user interface and functionality enhancements (Depradine and Gay, 2004). It acts as a front end to the Java SDK. Some important of its features are: (i) it checks the code as it is typed and displays the errors in real-time, (ii) it provides templates that represent necessary sections of code and therefore save students considerable time and effort by reducing errors due to incorrect program structure, and (iii) it hides the complex user interfaces of the Java SDK tools by providing direct access to them and therefore avoiding transfer the focus from the course material to the tools.

- ***Greenfoot*** (http://www.greenfoot.org) is a tool for teaching OOP aimed at students at college level or below. It is designed by combining the most useful characteristics of some existing tools. Its design was mainly inspired by the simplicity and the excellent visualization aspect of objects, their state and behavior in Karel the Robot environment and also by the strong support for direct object interaction in BlueJ environment (Henriksen and Kölling, 2004). This environment is a combination of a framework for producing Java programs that can be visualized in two-dimensional grid and of an IDE (with class browser, editor, compiler, execution control, debugger etc.) suitable for novice programmers.
- The ***P-coder*** tool (Roy and Armarego, 2004) provides a complete environment to demonstrate and implement many OO concepts for novice users. It focuses on developing and understanding the main principles using graphical and textual pseudocode notation representing the important programming elements in a tree structure. The environment supports multiple views of a program as: Design View, Class View, Code View and Object View. Using this system, users can design, implement, compile and run complete programs as usual and also it supports interactive objects' creation. The P-coder is implemented to produce Java code without supporting the full language, but this does not affect the novice users.

Tools for enhancing the capabilities of programming environments

The main scope of these tools is to support novices in compiling Java programs and to offer the on-demand possibility for the examination of code for its style, layout, comment coverage and correct indentation. The most well known tools of this category are:

- The ***Java Tool set*** which is a suite consisted of three related applications *(Java Pre-Scanner, Instant Feedback, Pretty Printer)*, written in pure Java (Lang, 2002).
- The ***Expresso*** is an educational tool for Java programming which enhances the functions of a compiler (Hristova et al., 2003). It is a pre-compiler that generates better error messages than existing compilers and also provide suggestions on how to fix the code.

Software tools supporting "objects-first" teaching approach

The "objects-first" teaching approach advocates that students should be exposed to the OOP concepts from the early beginning of their studies in programming. (Kölling and Rosenberg, 2001). It also advocates the gradual explanation of concepts from simple to higher level ones hiding the details of the OOP language. Various aforementioned environments such as BlueJ, greenfoot, JPie, jGrasp, KarelJ.Robot, Jeroo, Alice and objectKarel could also be included in this category. Other well known tools are:

- ***Java Power Tools (JPT)*** (http://www.ccs.neu.edu/jpt/jpt_2_3/index.htm) (Rasala et al., 2001; Proulx et al., 2002; Cooper et al., 2003) provides a comprehensive, interactive GUI, consisting of several classes with witch the student will work. Students interact with the GUI, and learn about the behaviors of the GUI classes through this interaction. It is a tool based on the event-driven programming, which depends on starting with ready-made code programs modified and extended by students.

- Various ***Graphics Libraries*** (Bruce et al., 2001; Roberts and Picard, 1998) are used in an "objects-first" approach. There is some sort of canvas onto which objects are drawn. These objects may have methods invoked on them and they react accordingly. Such libraries are: ii) an elegant and small graphics package called ***Nice Graphics Package (NGP)*** (Alphonce and Ventura, 2003) aiming at keeping the attention of students and simultaneously supporting teaching of fundamental OO principles. NGP allows the creation of event-driven graphical programs using only inheritance and method overriding. It provides students with classes which hide the complexity of AWT/Swing and thus allows focusing on using graphical event-driven programs to deepen understanding of OO concepts (ii) the ***objectdraw*** library (http://cortland.cs.williams.edu/~cs134/eof/library.html) (Bruce et al., 2001) supports OO graphics and enables the incorporation of event-driven programming techniques.

Gaming environments

The most well known repsresentative of these environments is the ***Robocode*** (http://robocode.sourceforge.net/). It is a shooting game between tanks (small automated robots with six wheels), in which a small sequence of Java code can be made in order to create a complex game (Borge and Kaasbøll, 2003). Base concept is the active objects, i.e. entities with their own behavior that can interact with other objects in different ways based on the situation. When one tank sees another tank or gets hit by a shot (or whatever else happens) an event is created and students can program the tank to react in a certain way. The better the tank handles these events, the better the tank will do in a battle. Tanks are battling until only one of them remains running. With Robocode, students learn OO concepts such as: subclasses, method calls, method definitions, event handling, inheritance, encapsulation, and polymorphism.

Educational programming languages

Various educational OO languages have been proposed to help students better acquire knowledge and skills of OO. For example the ***PIIPOO*** is a language for use when the programming paradigm is changed from Procedural to Object Oriented (Fernández Muñoz and Peña, 2002). Another such language is the ***Initial Object-Oriented Programming Language (IOPL)*** (Harrison et al., 2005). It is designed for an introductory level course which is accompanied by a structure editor. Classes are constructed via this editor that ensures syntactic correctness and provides "templates" in order that students can "fill in" incomplete code parts during software development. IOPL is a combined language of Pascal-like and Smalltalk-like syntax. It provides the possibility for interactive objects' creation, methods' invocation and objects' state inspection.

In Tables 1 and 2 (Appendix A), we attempt a brief comparative analysis for the above mentioned tools and environments taking into account the comparative analysis approach proposed by McIver (2002). The first Table contains the first two categories of tools and the second the rest ones. As can be easily seen, the Bluej's educational added value is obvious. It offers many features and it is a mature educational tool (Hagan and Markham 2000; Van Haaster and Hagan, 2004). The greenfoot environment incorporates the extra feature of dynamic visualization, but usage results have not recorded in literature, according to our knowledge.

A usage case study of tools for teaching OO design and programming

In an attempt to test and validate some of the findings of the comparison between the tools, we designed a technology enhanced learning script for teaching OOP based on the "modelling first" approach. This script was tested during a two and a half months seminar for postgraduate students. In brief, the proposed learning script is comprised of learning activities that occur during three main phases of the instructional process: (i) *observation* of the development process of an exemplar OO software application (ii) *problem solving tasks with guided instructions* and (iii) *autonomous problem solving task.*

During the *first phase* the expert (teacher) is called to analytically present the way of designing and implementing an OO software application using well-suited examples, i.e. authentic (real life) examples close to the interests and social context of the students, utilizing modeling Computer Aided Software Engineering (CASE) tools, such as UML (Unified Modelling Language) CASE tools. In our seminar we chose the *ArgoUML CASE tool* (http://argouml.tigris.org/) for supporting the main design process of the exemplar case and the *BlueJ* educational environment for further development of the initial exemplar OO software application. The teaching of the implementation process of the exemplar application was also supported by the *SUN One Studio IDE* (http://java.sun.com/). It was used as the proper instructional medium to demonstrate the creation, compilation, and execution of the "completed" exemplar OO software application (with a "main" method). Thus students could learn how to use a "professional" development tool and acquire an overall picture about program flow (Ragonis & Ben-Ari, 2005a). Hence, the overall test attempted the combination of educational tools with professional environments.

In the *second phase* students try to internalise the tacit knowledge they have just listened to. So they are asked to develop and deliver a small OO software application following the exact same steps of the expert. During this process detailed instructions about OO design and programming, i.e. scaffolds, are given to them. These scaffolds support active and exploratory learning thus offering students the opportunity to explore concepts from multiple perspectives and also to see and consolidate experts' solutions to problems. The scaffolds are mainly on-line learning resources integrated into an on-line Learning Management System such as the Moodle system (http://www.moodle.org). The resources offered are: (i) the video-taped step-wise teacher's explanation of the OO software application development process of an exemplar OO application, (ii) brief pieces of theory accompanied by representative examples from everyday life, (iii) short exercises for further practice about OO concepts (not obligatory) with their solutions, (iv) well-written code examples, (v) a well documented case study of an OO software application, containing the requirements specification, the OO analysis, the design decisions, the class diagrams, the source code, and testing cases, and (vi) study guides included in each didactic unit and other informative material such as tools' installation and usage manuals, links to other resources on Java language, etc. The learning resources had been structured into didactic units that relate to the fundamental OO concepts and principles that students encountered during the observation phase. Moreover, as scaffolds for knowledge construction we offered (i) an asynchronous web discussion forum where students could post questions that the teacher/expert or other peers could answer thus forming a community of practitioners, and (ii) some face to face meetings for addressing learning difficulties in person.

In the *third phase* the students act independently, developing an OO software application and submitting it in one piece at a specific deadline without necessarily following the steps that had been taught during the previous phases. In brief, the three phases' are shown in Figure 1.

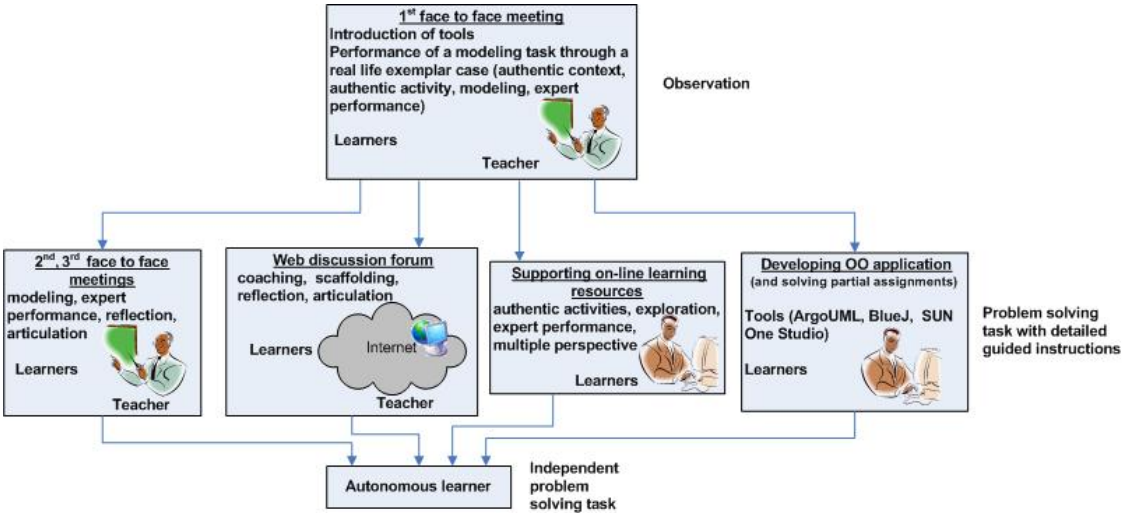

Figure 1. Overview of the learning script

Evaluation study

A systematic and thorough evaluation study has been performed for the above described case study aiming at: (i) investigating the learning effectiveness of the technology enhanced learning script, and (ii) examining which factors of the script affected most the seminar's learning effectiveness. One of these factor is the *BlueJ* usage and its combination with a professional programming environment such as SUN One Studio.

Eighteen (18) students registered for the seminar (17 men, 1 woman) participated in the study.

Instruments for Data Collection

We collected students' opinion using two types' questionnaires and focus group interviews from three (3) students (randomly chosen). The first type of questionnaire ("pre-test") was given to the students at the end of first face to face meeting after they have listened to the seminar's instructional philosophy. The second questionnaire ("post-test") was returned by the students after the end of the seminar. Additionally we analyzed students' assignments delivered at the second phase.

Some questions of the "pre-test" were replicated in the "post-test" in order to measure the seminar's effect. However, the second 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. 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 concerned the students' likes and dislikes towards the learning resources, the deficiencies concerning the resources and the approach and suggestions for improving either of them.

Data Analysis and Results

Overall, the analysis of the students' assignments revealed that students conceptualized well the OO concepts and acquired abstract knowledge that was applied successfully to the new problem. Our seminar seems to have been successful. The great majority of participants was able to model an OO software application, to design class diagrams in ArgoUML tool, and successfully implement their assignment using the BlueJ tool, first, and the SUN One Studio, at the end. Students' performance in a scale from 0 to 100 is shown in Figure 2.

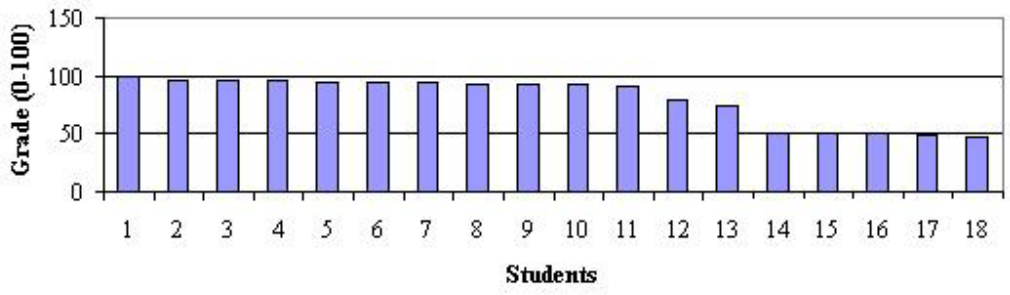

Figure 2. Students' grades for the final assignment

The great majority of students achieved a very good grade. Concerning the students who achieved a grade around the "pass" level, they had time constraints and hence, they could not to deliver the final part of their projects in a complete form.Only one student failed to successfully complete the seminar.

Additionally in the questionnaires, students estimated that their competence in OOP and their ability to write programs in Java was at a good level at the end of the instructional process, and stated that their knowledge level was better after the seminar as shown in Table 3. Of course, these statistics show some trend since the number of students who participated in the evaluation study was relatively small.

Table 3. Students' self-estimation about their level in OOP

Question	Mean (pre-test)	Mean (post-test)
How do you rate your level about OOP?	2.72	2.83
How would you assess your ability to write programs in Java?	-	3.22*
Based on what you have been taught till now, how well do you think that you have learned OOP?	3.06	3.33**

Answers' coding: 5= Professional, …, 1= Novice
* Answers' coding: 5= Very much capable, …, 1= Not at all capable
**Answers' coding: 5= Very much, …, 1= Not at all

Especially about the choice of utilizing the *BlueJ* environment as a modeling and implementation tool, students were asked in the "post-test" questionnaire to state the most valuable features and the reasons for using it. The collected results are depicted in the Table 4. Students particularly appreciated its visualization, simplicity and interactivity attributes (Kölling & Rosenberg, 2001; Kölling et al., 2003).

Table 4. Students' opinion about the most valuable features/reasons for using the BlueJ environment in OOP

Question: Which features of the BlueJ environment made you use it for performing learning tasks? Select those that you think.	**Number of students who chose a feature (max 18)**
Classes and objects are represented graphically.	12/18
Easy instantiation and visual representation of objects.	12/18
Ease-of-use of the environment.	11/18
Users can interact directly with classes and objects.	9/18
Users can easily inspect attributes' values of the objects.	9/18
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.	8/18
It's easy to compile programs. Compile-time errors are displayed directly in the source editor by highlighting the error line.	8/18
You can get immediate feedback after having experimented with it	8/18
The environment incorporates an easy-to-use debugger.	5/18
The environment shows the 'implementation' aspect and 'interface' aspect of classes.	4/18

More analytically, as can be seen in Table 5, when students responded regarding the contribution of various learning resources to the enhancement of their knowledge and skills, the following mean values resulted, concerning activities in the *BlueJ* environment, ready-made projects in BlueJ, activities in the ArgoUML and SUN One Studio environments.

Table 5. Contribution of learning resources to the enhancement of knowledge and skills

Question: Evaluate in what degree the following learning resources contributed the enhancement of knowledge and skills.	**Mean (From 'post-test')**
The activities in the BlueJ environment.	4.17
The ready made BlueJ projects.	3.94
The activities in the ArgoUML environment	3.28
The activities in the SUN One Studio environment	3.80

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

As it can be seen, the use of *BlueJ* offered great help to students. The utilization of SUN One Studio was also highly appreciated. Moreover, the analysis of the students' assignments showed that they achieved to use successfully this IDE, and produced correct programs. Students did not highly appreciate the use of the ArgoUML environment (as they did for other factors). This finding in combination with some students' answers in related questions showed that although they appreciated the task to design class diagrams and create code skeletons for their applications using a CASE tool, they faced some usability problems when using this tool. Nevertheless, all students managed to submit their designs which most of them were correct.

In the open-ended questions of the "post-test" questionnaire, some students stated that more examples in using ArgoUML should be needed, for its better utilization.

Concluding remarks and future trends

The systematic utilization of educational tools for teaching programming is well documented in the literature, since the Procedural's programming paradigm days. In this paper we conducted a literature review about tools designed for supporting the instructional process of OO programming. We conducted a comparative analysis which showed that the BlueJ environment holds a lot of advantages. These advantages had been also been supported by 18 students who participated into a OOP seminar. This seminar had been designed based on a technology enhanced learning script.

The evaluation study of this seminar showed that students highly appreciated the BlueJ tool for its simplicity and usability. They also commented that the activities performed with this tool enhanced their knowledge and skills.

BlueJ does not enable students to design software programs using the pure UML notation, and then to generate classes' code with code skeletons for further editing. The development of educational tools which will provide dynamic visualization of the software programs under development and also a full "view" of objects' life-cycle and program flow is a necessity (Ala-Mutka, p.9-10; Ben-Ari et al., 2002). The next generation of tools that support the instructional process of the OOP will include these features.

Acknowledgement

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

References

ACM (2002). Obituary. Retrieved April 30, 2007, from http://www.acm.org/announcements/turing_obit.html

Ala-Mutka, K. *Problems in Learning and Teaching Programming. A literature study for developing visualizations in the Godewitz-Minerva project*. Retrieved December 29, 2006, from www.cs.tut.fi/~codewitz/literature_study.pdf

Allen, E., Cartwright R., & Stoler, B. (2002). DrJava: A lightweight pedagogic environment for Java. *ACM SIGCSE Bulletin, 34*(1), 137-141.

Alphonce, C., & Ventura, P. (2003). Using graphics to support the teaching of fundamental object-orientation principles in CS1. In *Proceedings companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '03)*, Anaheim, California, October 26-30, 156-161.

Barnes, D. J., & Kölling, M. (2005). *Objects First with Java, A Practical Introduction using BlueJ*. Prentice Hall, Second edition.

Barr, M., Holden, S., Phillips, D., & Greening, T. (1999). An exploration of novice programming errors in an object-oriented environment. *ACM SIGCSE Bulletin, 31*(4), 42-46.

Bennedsen, J., & Caspersen, M. (2004). Programming in Context - A Model-First Approach to CS1. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Educatio*n, Norfolk, Virginia, March 3-7, 477-481.

Ben-Ari, M., Ragonis, N., & Ben-Bassat Levy, R. (2002). A Vision of Visualization in Teaching Object-Oriented Programming. In *Proceeding of 2nd Program Visualization Workshop*, HornstrupCentret, Denmark, 83-89.

Berge, O., Borge, R. E., Fjuk, A., Kaasbøll, J., & Samuelsen, T. (2003). Learning Object Oriented Programming. Norsk Informatikkonferanse NIK'2003, Tapir Akademisk Forlag, 37-47.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (2005). *Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. USA: Dream Songs Press.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (2003). *Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java.* Published manuscript. Retrieved December 31, 2006, from http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (1997). *Karel++ the Robot: A Gentle Introduction to the Art of Object-Oriented Programming* (2nd ed.). New York: John Wiley & Sons.

Booch, G. (1991). *Object Oriented Design with Applications.* Bedwood City, California: The Benjamin/Cummings Publishing Company.

Borge, R. E., & Kaasbøll, J. (2003). What is "OO first"? *The* 7th *Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts at 17th European Conference on Object-Oriented Programming*, July, Darmstadt, Germany.

Bruce, K., Danyluk, A., & Murtagh, T. (2001). A library to support a graphics-based object-first approach to CS 1. *ACM SIGCSE Bulletin 33*(1), 6-10.

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

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

Buck, D., & Stucki, D.J. (2001). JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. *ACM SIGCSE Bulletin, 33*(1), 16-20.

Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook*, 2006-07 Edition, Computer Programmers. Retrieved on June 15, 2007 from http://www.bls.gov/oco/ocos110.htm (visited).

Carter, J., & Fowler, A. (1998). Object Oriented Students? *ACM SIGCSE Bulletin, 28*(3), 271.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching Objects-first in Introductory Computer Science. *ACM SIGCSE Bulletin 35*(1), 191-195.

Cooper, S., Dann, W., and Paush, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing in Small Colleges, 15*(5), 108-117.

Depradine, C., and Gay, G. (2004). Active participation of integrated development environments in the teaching of object-oriented programming. *Computers & Education, 43*(3), 291-298.

Fernández Muñoz, L., Peña, R. (2002). PIIPOO: An Adaptive Language to Learn OO Programming. Position paper presented at *16th European Conference on Object-Oriented Programming*, 6th Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts (or workshop "Tools and environments for Learning Object Oriented Concepts"), Malaga, Spain.

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

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

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

Goldman, K. J. (2004). A Concepts-First Curriculum for Introductory Computer Science. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Educatio*n, Norfolk, Virginia, March 3-7, 432-436.

Goldman, K. J. (2003). A Demonstration of JPie: An Environment for Live Software Construction in Java. In *Proceedings companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '03),* Anaheim, California, October 26-30, 74-75.

Gray, K. E. and Flatt, M. (2003). Professorj: a gradual introduction to java through language levels. In *Proceedings companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '03),* Anaheim, California, October 26-30, 170-177.

Hagan, D., & Markham, S. (2000). Teaching Java with the BlueJ Environment. Paper presented at the *Australian Society for Computers in Learning in Tertiary Education* (ASCILITE), Coffs Harbour, Australia.

Harrison, C.J., Sallabi, O.M., & Eldridge, S.E. (2005). An initial object-oriented programming language (IOPL) and its implementation. *IEEE, Transactions on Education, 48*(1), 3-10.

Hendrix, T.D., Cross, J.H., & Barowski, L.A. (2004). An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. *ACM SIGCSE Bulletin, 36*(1), 387-391.

Henriksen, P., & Kölling, M. (2004). Greenfoot: Combining Object Visualization with Interaction. In *Proceedings companion of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '04)*, Vancouver, BC, Canada, 73-82.

Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. In *Proceedings of the 28th SIGCSE,* 131-134.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education,* Reno, Nevada, USA, 153-156.

Kölling, M., Quig, B., Patterson, A., & 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.

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

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers**.** In *Proceedings of the 10th Annul SIGCSE conference on Innovation and Technology in Computer Science Education,* 14 – 18, June 27-29, Monte de Caparica, Portugal, also published in *ACM SIGCSE Bulletin, 37*(3), 14-18.

Lang, B. (2002). Teaching new programmers: a Java tool set as a student teaching aid. In *Proceedings of the inaugural conference on the Principles and Practice of programming,* and in *Proceedings of the 2nd workshop on Intermediate representation engineering for virtual machines*, Dublin, Ireland, 95-100.

Lewis, J. (2000). Myths about Object-Orientation and Its Pedagogy. In *Proceedings of the 31st SIGCSE technical symposium on Computer Science Education,* Austin, Texas, March, 245-249.

McIver, L. (2002). Evaluating Languages and Environments for Novice Programmers. *14th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002),* Middlesex, UK, 100-110.

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.

Moskal, B., Lurie, D., and Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education,* Norfolk, Virginia, March 3-7, 75-79.

Pattis, R. E. (1981). *Karel the Robot: A gentle Introduction to the art of Programming.* John Wiley & Sons.

Powers, K., Cooper, S., Goldman, K., Carlisle, M., McNally, M., & Proulx, V. (2006). Tools for teaching introductory programming: What works? In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education.* Houston, Texas, March 1-5, 560-561.

Proulx, V., & Gray, K. E. (2006). Design of Class Hierarchies: An Introduction to OO Program Design. *ACM SIGCSE Bulletin 38*(1), 288-292.

Proulx, V., Raab, J., & Rasala, R. (2002). Objects from the beginning – with GUIs. *ACM SIGCSE Bulletin 34*(3), 65-69.

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

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

Ragonis, N., & 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.

Rasala, R., Raab, J., & Proulx, V.K. (2001). Java Power Tools: Model Software for Teaching Object-Oriented Design. *ACM SIGCSE Bulletin 33*(1), 297-301.

Roberts, E. (2006). An interactive tutorial system for Java. *ACM SIGCSE Bulletin 38*(1), 334-338.

Roberts, E., & Picard, A. (1998). Designing a Java graphics library for CS1. In *Proceedings of the 3rd Annul SIGCSE conference on Innovation and Technology in Computer Science Education,* Dublin, Ireland, July, 213-218.

Roy, G.G., & Armarego, J. (2004). Teaching programming with objects. Australasian Society for Computers in Learning in Tertiary Education Conference, Perth, Western Australia. Retrieved Jan. 4, 2007, from http://www.ascilite.org.au/conferences/perth04/procs/ roy.html

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

Teif, M., & 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.

Turner, T. A., & Zachary, L. (2001, February). Javiva: ATool for Visualizing and Validating Student-Written Java Programs. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, Charlotte, 45-49.

Van Haaster, K., & Hagan, D. (2004). Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool. *Information Science & Information Technology Education Joint Conference*, Rockhampton, QLD, Australia, 456-470.

Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles, *International Journal of Human-Computer Studies, 51*(1), 71-87.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore C. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers, 11*(3), 255-282.

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

---

[1] Ms. Stavroula Georgantaki is a doctoral candidate at the University of Piraeus in Greece. She can be reached at Department of Technology Education and Digital Systems, University of Piraeus, 80 Karaoli & Dimitriou str., GR-18534 Piraeus, Greece. Phone: +(30) 210-4142765; Fax: +(30) 210-4142753; E-mail: rgeo@unipi.gr.

[2] Dr. Symeon Retalis is an Assistant Professor at the Department of Technology Education and Digital Systems, University of Piraeus, Greece. He can be reached at 80 Karaoli & Dimitriou str., GR-18534 Piraeus, Greece. Phone: +(30) 210-4142765; Fax: +(30) 210-4142753; E-mail: retal@unipi.grtechnology.

Appendix A

Table1. Comparative presentation for programming microworlds and educational programming environments

Features	Programming microworlds					Educational Programming Environments									
	JKarelRobot	Jeroo	Alice	objectKarel	BlueJ	DrJava	Professor J	JPie	Javiva	GinuPad	jGrasp	jCreator	D-Chk	greenfoot	P-Coder
Simplicity	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	
Interactivity					✓	✓		✓			✓			✓	✓
Visualization		oo[1]	oo[1]	oo[1]	✓			gpc[2]	avo[3]	tscma[4]	otf[5]	tscma[4]		✓	
Modeling					✓									✓	
Program's structure visualization					✓						✓			✓	
Dynamic Visualization	✓	✓	✓	✓										✓	
Code skeletons					✓									✓	
Graphical interface	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	
Full Java support					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Easy compilation		✓		✓	✓	✓	✓			✓	✓	✓	✓	✓	
Notation of wrong code line		✓		✓	✓	✓				✓	✓	✓	✓	✓	
Debugger	✓				✓	✓					✓			✓	
Usage maturity	enough	small	small	small	enough	small	small	small	small	small	small	small	small	small	small

1 oo = only objects
2 gpc = graphical presentations of concepts
3 avo = abstract visualization of objects
4 tscma = tree structure fior classes-methods-attributes
5 otf = objects with textual form

Table2. Comparative presentation for the rest tools

Important features	**For enhancing capabilities of programming environments**		**Supporting "objects first" approach**		**Games' environments**	**Educational programming languages**	
	Java Tool Set	Expresso	Java Power Tools	Graphics Libraries	Robocode	PIIPOO	IOPL
Simplicity							
Interactivity							✓
Visualization				go [6]	oo [7]		
Modeling							
Program's structure visualization							
Dynamic Visualization					✓		
Code skeletons							
Graphical interface							
Full Java support	✓	✓		✓			
Easy compilation	✓	✓	✓				
Notation of wrong code line		✓					✓
Debugger							
Usage maturity	small	small	Small	small	small	small	small

[6] go = graphical objects
[7] oo = only objects

Page left blank