Tools and Environments for Understanding Object-Oriented Concepts

Isabel Michiels[1], Alejandro Fernández[2], Jürgen Börstler[3], and Maximo Prieto[4]

[1] PROG, Vrije Universiteit Brussel, Belgium
[2] GMD-IPSI Darmstadt, Germany
[3] Umeå University, Sweden
[4] Universidad Nacional de La Plata, Argentina

Abstract. The objective of this workshop was to discuss current tools and environments for learning object-oriented concepts and to share ideas and experiences about the usage of computer support to teach the basic concepts of object technology. Workshop participants presented current and ongoing research. During the discussions the participants developed a general *"package"* of requirements for such tools and environments, which underlying pedagogical approaches could be applied, and how such tools and environments should look like.

1 Introduction

Successfully using object-oriented technology in a development project requires a thorough understanding of basic object-oriented concepts. However, learning these techniques has proven to be very difficult in the past. Misconceptions can occur during the learning cycle and the needed guidance can not always be directly provided.

The goal of this workshop was to share ideas and experiences about the usage of computer support in learning and teaching the basic concepts of object technology. This could be either tools used in environments, specific environments for learning object-oriented, as well as any kind of support for developing object-oriented learning applications themselves.

In order to develop useful results regarding the issue of understanding object-oriented concepts, the workshop wanted to focus on the following topics:

- Computer-based learning of object-oriented concepts;
- Intelligent environments for learning basic object-oriented concepts;
- Frameworks or class libraries to support learning;
- Microworlds for learning about the concepts of object-oriented technology;
- Any other kind of tool support for learning object-oriented technology;
- Frameworks or toolkits to support the development of teaching or learning applications.

This was the fourth in a series of workshops on issues in object-oriented teaching and learning. Previous workshops were held at OOPSLA'97 [1,2], ECOOP'98 [3] and OOPSLA'99 [4].

J. Malenfant, S. Moisan, A. Moreira (Eds.): ECOOP 2000 Workshops, LNCS 1964, pp. 65–77, 2000.

Table 1. Workshop program

Time	Topic
9.00 am	WELCOME NOTE
9.15 am	Close the window and put it on the desktop, presented by James O. Coplien
9.35 am	A library to support a graphics-based object-first approach to CS1, presented by Kim Bruce
9.55 am	Introducing Objects with Karel J. Robot, presented by Joseph Bergin
10.10 am	Experimentation as an aid for learning, presented by Dan Rozenfarb
10.30 am	COFFEE BREAK
11.00 am	Combatting the Paucity of Paradigms in Current OOP Teaching, presented by Theo D'Hondt
11.20 am	Minimalist Tools and Environments for Learning about Object-Oriented Design, presented by Mary Beth Rosson
11.35 am	A Programming Language for Teaching Concurrent Object Oriented Concepts, presented by Laszlo Kozma
12.00 am	LUNCH BREAK
1.00 pm	A Model-Oriented Programming Support Environment for Understanding Object-Oriented Concepts, presented by Stephen Eldridge
1.15 pm	Exploratorium: An Entry-Level Environment for Learning Object Technology, presented by Jan Schümmer and Alejandro Fernández
1.35 pm	A Tool for Co-operative Program Exploration, presented by Torsten Holmer and Jan Schümmer
2.00 pm	First Working group session
3.30 pm	COFFEE BREAK
4.00 pm	Second Working group session
5.00 pm	Wrap-up session

2 Workshop Organization

The workshop organizers wanted to gather people that were involved in the development or use of learning support for any kind of learning tool or environment. To get together a manageable group of people in an atmosphere that fosters lively discussions, the number of participants was limited. Participation at the workshop was by invitation only. Eighteen participants were selected on the basis of position papers submitted in advance of the workshop.

The workshop was organized into two presentation sessions (all morning), a demo session (after lunch), two working group sessions (afternoon) and a wrap-up session, where all working groups presented their results. Table 1 summarizes the details of the workshop program.

To gather some input for the working group sessions participants were asked to finish their presentations by raising a few central questions or problems that were (not) addressed by their position statement. Related questions and problems were then grouped into working group topics.

3 Summary of Presentations

This section summarizes the main points of all workshop presentations. More information on the positions presented at the workshop as well as further position papers accepted for the workshop can be obtained from the workshop's home page at `http://prog.vub.ac.be/ecoop2000/`.

Jim Coplien gave an introduction and critical review on the *Minimalist Approach* as a learning paradigm as proposed by John M. Carrol's book "The Nürnberg Funnel" [6]. The basic idea of this approach is to support active, explorative learning. Learners possess relevant knowledge and use it to make sense of new information. Real tasks should be embraced early. Only a minimum of guidance/instructions is given. This will lead to real mistakes, which can be used as a learning opportunity.

The talk's main point was that we have created an artificial dichotomy between pedagogy and application, i.e. the usage of the knowledge acquired through education. Although we recognize that good pedagogy must include application, we ignore that good application must attend to pedagogy. Using a program is a learning experience, and a program should be designed with that in mind.

"The Nürnberg Funnel" provides an extensive case study demonstrating that people are best at learning through experience and experimentation. Jim pointed out two important points overlooked by the book:

- Much of the learning in the case studies involved mastering gross mismatches between the learner's expectation and what the program provided; one can easily postulate that much of the learning would have been unnecessary had the program designers been more attentive to usability.
- Learning and experimentation are an essential component of any human activity, including our daily use of both familiar and new computing tools. These behaviors are not limited to institutionalized pedagogy. That in turn suggests that the same mechanisms, tools, and design rules that apply to pedagogical environments should become the stock and trade of application developers as well.

The talk concluded with a vision of going forward with interactive program design: We should either get rid of all the programmers and have all programs written by developers of electronic learning environments, or we should get rid of all training software developers and let training emerge as a natural outgrowth of the use of tools that support the learning process. This approach could build on a wealth of longstanding complementary practices, such as the direct manipulation metaphor, which software developers employ in the designs both of their systems and the interfaces by which users become acquainted with them.

Kim Bruce presented experiences using the *MagicDraw* library developed at Williams College. MagicDraw was developed to support an "objects-first, graphics-first" approach for the teaching of Java in a CS1 course. The library enables them to:

- Use an object-first approach, requiring students to think from the start about the programming process with a focus on methods and objects.
- Use graphics and animation extensively. Using graphics was considered an important aspect of a tool both because students were able to create more interesting programs, and because graphic displays allowed students to receive visual feedback when they made programming errors.
- Introduce event-driven programming early in the course. Most of the programs students use today are highly interactive. Writing programs that are similar to those they use is both more interesting and more "real" to the students.
- Introduce concurrency as a natural tool for program construction, especially when used in conjunction with animation.

The interest in introducing the last two items early was motivated, in part, by Lynn Andrea Stein's [10] work on highlighting the importance of interaction in an introductory course.

Using the library instructors can focus on getting the concepts across unobscured by the complexities of language constructs, which are designed to be more flexible and powerful than needed in an introductory course. At the same time, the library was carefully designed to allow a smooth transition to the raw language facilities once the students are comfortable with the concepts.

Joseph Bergin talked about *Karel J. Robot*, to support the introduction of object-oriented concepts in Java courses. Karel J. Robot is a successor of *Karel the Robot*, which was initially developed to introduce structured programming in Pascal to novices [9].

The idea of using a robot is that students are familiar with the concept of a robot. In the Karel world, robots live in a space that represents the first quadrant of the Cartesian plane, with horizontal streets at the integer points and avenues at the vertical points. There can be walls between the streets and avenues, and the world can contain beepers that the robots can pick up and carry. Furthermore, students can "act as a robot." Behaving consistently with the associated metaphor makes it easy for students to understand what is going on in a program.

Different versions of the robot exist, which are all built around pedagogical patterns [8,11]. The most important pattern for the development of the approach taken by Karel J. Robot is *Early Bird* pattern. This pattern helps to order class topics in order of importance and find ways to teach the most important ideas early.

Karel J. Robot courses start by teaching methods, objects, and classes, because they are the prerequisites for teaching polymorphism (the key object-oriented concept, according to Joseph Bergin). Students develop programs by inheriting from existing Robot classes, refining their behavior.

The Karel philosophy supports a *Spiral Teaching Approach*, i.e. students can successively deepen their knowledge while having the tools to build exploratory

programs from the start. The pedagogical foundings are well designed and the tools' GUI's are appealing.

Experiences at Pace University are promising. Karel J. Robot has proven to be very successful in introducing object-oriented programming to novices.

Dan Rozenfarb gave a talk on experimentation as an aid for learning. His main research area is the construction of domain-specific frameworks and the importance of experimentation to acquire knowledge.

Learning is achieved through many different means. A single tool is therefore not enough to support the learning process. An appropriate learning environment needs a set of tools that trigger off experimentation and support various learning processes.

The talks main points can be summarized as follows:

- Experimentation is a way to learn and to acquire knowledge;
- There are many factors that deter experimentation and learning, like
  - lack of immediate feedback,
  - lack of support for decision making,
  - limited undo facilities,
  - testing/debugging support (for partly finished classes),
  - indirect or no object manipulation,
  - too many classes and objects;
- The need of an integrated environment with tools that facilitate experimentation;
- A proposed set of tools that alleviate the problems.

Reviewing the literature the following topics where consistently mentioned as a necessity for teaching object-orientation:

- Complexity hiding and gradually revealing details;
- Integrated and live environments that help the students to concentrate on the main issues;
- The need for us (the teachers) to define what is essential in the object oriented paradigm and what is not;
- Emphasis on design issues from the very beginning;
- The importance of showing the students that there is something different to Java.

Theo D'Hondt was very concerned about the current trend in OOP teaching to use Java only as a (first-year) programming language. Because of the overwhelming success of Java as "the" language for the internet, the continuous demand for skilled Java programmers has led to tremendous pressure on educational systems to promote Java as a first-year language. As a consequence, computer science graduates are no longer familiar with a wide range of programming paradigms.

Going against this trend, Theo reported on his experience with Pico, a *tweakable* language implementation for teaching, as a medium for learning *all* concepts

of object technology. Pico was used in an experiment within the context of a European Master on Object-Oriented Software Engineering [14]. The students had very different backgrounds, so there was a strong need to establish a common ground among the group.

Pico is a self-contained technological framework for uniting all relevant object-oriented concepts. It is a simple, dynamically typed language with automatic memory management similar to the Scheme language. Pico is *tweakable* in the sense that moving from the original language to a derived language is very simple. Since every aspect of the language is first class, new features can be prototyped and moved out into the implementation of the actual virtual machine.

Once the students managed the approach of the prototype-based version of Pico, they could easily describe a class-based tweak of the Pico language.

Using a tweakable virtual machine as technological support proved to be extremely efficient. Using well-known concepts such as functions, closures, scoping, etc., it was possible to describe the semantic notions of the object paradigm. Theo believes that a similar but simplified approach can be successfully applied at the undergraduate level.

Mary Beth Rosson talked about the problems of teaching design as a component of the standard object-oriented programming course offered by the Department of Computer Science at Virginia Tech. All of her work is example-based and grounded in the principles of minimalist instruction [6].

The essence of minimalism is to assume from the start that learners possess a considerable amount of knowledge, and that they will use it in making sense of new information. Example-based learning is a powerful technique for minimalist instruction; examples can be complex and therefore more realistic, as long as appropriate guided exploration is provided.

The intent of one project was to provide an overview of Smalltalk. The teaching materials were minimal, i.e. just enough to make experienced programmers curious and therefore learn the rest on their own. One of the objectives was to convey the MVC framework of Smalltalk. The learning materials to achieve this objective was a card game. At first, all Smalltalk code was hidden and the learners just interacted with a bare model. Afterwards, the code behind message sends was revealed. The exploration of this tool was supported by the View Matcher, a kind of specialized debugger.

Because this tool proved to be quite successful, the same kind of tool was used to tackle the problem of reuse. Classes designed for reuse were documented implicitly by examples, and then the tool was used to explore these examples. This idea was based on the fact that expert programmers have a need to learn when they attempt to reuse a class, with the difference that they know much more to start out with.

The minimalist approach proved to be quite successful for "getting people to do something." An open problem is however, how this approach scales up to more complex maintenance or software engineering issues.

Lazlo Kozma presented a language to teach and learn concurrency. Concurrency is a natural concept, but not its realization inside a programming language. Different programming languages take different approaches, making it necessary to study a variety of programming languages to fully understand the concept. However, expecting students to learn a lot of languages to gain a deeper understanding of concurrency is clearly undesirable. Concurrent object-oriented concepts could be understood more easily if we had a programming language providing kinds of tools to express concurrency and object-orientation at the same time.

To achieve this aim an object-oriented extension to Pascal-FC (Functionally Concurrent Pascal) was proposed. Pascal-FC was originally developed as a teaching language for concurrency that supports all major concurrency primitives present in current programming languages [5].

In the object-oriented extension active objects are represented by processes and passive objects can be represented by monitors, resources, or processes. The central concept of concurrent object-oriented programming, besides concurrency, is knowledge-sharing, i.e. the re-use of object descriptions. The advantage of knowledge-sharing lies in increased modularity and hierarchical structuring. The tools for knowledge-sharing are subtyping and inheritance.

As a model of computation a reflective model of objects was implemented. Different kinds of synchronization mechanisms were included into the language to avoid inheritance anomalies. The general object level synchronization schemes (guarded methods, synchronization with enabled sets, synchronizers, transition specifications and synchronization sets) were implemented as well. Users can develop a concurrent program using concurrent objects with different kinds of synchronization mechanisms. The abstraction level of these mechanisms is somewhat different. Some mechanisms can be interpreted as either specification or implementation of synchronization of methods. To avoid inheritance anomalies the so-called sequential code of a method was separated from the synchronization code, so both codes could be inherited separately.

Stephen Eldridge introduced us to MPSE, a **M**odel-oriented **P**rogramming **S**upport **E**nvironment (MPSE) designed specifically to support the teaching of fundamental and general concepts that underpin the object-oriented paradigm. The main aim of the environment is to teach the underlying concepts of object-orientation via the specification of the desired properties of objects and to abstract over programming language and implementation specific details. The facilities provided by the MPSE support users whose level of skill changes significantly over time and which abstract over the more sophisticated facilities provided by conventional software development environments. The desired properties of a program are modeled directly in terms of object types and their attributes. A simple notation called MODEL is used to capture the desired properties of object types independently of implementation considerations.

Users interact with the system by directly manipulating objects whose attributes represent different kinds of description. The system has been used suc-

cessfully to support a wide variety of learning activities within the Computation Department at UMIST. The MODEL language is now suitable as a design language not just for use within a teaching system but also for more complex tasks and together with different concrete programming languages. A working prototype of the MPSE has been developed. Currently the system runs on Macintoshs only, but it should be easily portable to different architectures.

Jan Schümmer and **Alejandro Fernández** presented the *Exploratorium*, a cooperative learning environment especially designed to learn about building object-oriented systems. The Exploratorium combines common features of object-oriented CASE tools with visualizations and animations of object-oriented applications. Its purpose is to improve students' mental models of object-oriented systems by exploring the inner workings of real systems. Drawing on the experiences of both groups (GMD-IPSI [13], and LIFIA [12]) in the areas of object technology and cooperative learning environments, the Exploratorium is enriched with cooperation capabilities.

Users of the system can explore the definition and behavior of a working application that serves as an example. All aspects of the example application can be explored and modified cooperatively, from its user interface to its definition in Smalltalk. Even the example applications UML object and class diagrams can be explored and modified cooperatively.

Example applications are created by the teacher without the need of extra effort to include it into the exploration environment. The presentation included a demonstration of a prototype of the system where two users where cooperatively exploring an example mail client. As ongoing work, the authors are still exploring and evaluating the results of using the application. Research is being done in the areas of cooperative system visualization and development. It is also of interest to develop guidelines for the construction of example applications that are rich enough to serve the purpose of learning, but at the same time simple enough to make their exploration feasible.

Torsten Holmer and **Jan Schümmer** talked about the integration of the Exploratorium of the former talk with an existing cooperative virtual learning environment named VITAL [15]. VITAL is a shared environment in which teachers and learners can create and annotate hypermedia documents in synchronous and asynchronous modes. A state of an Exploratorium can be linked to the hypermedia structures of VITAL and thereby integrated into the shared learning and working material. This allows the training application (in the Exploratorium) be tightly linked with the learning environment and therefore a faster switching between reading and annotating and creating and exploring new object structures.

VITAL is written in VisualWorks Smalltalk (V2.5) and is thus available for all platforms supported by VisualWorks. Next steps are enhancing the stability of the prototype, looking for good object designs used in the Exploratorium,

developing concepts for courses which make use of these tools and the evaluation of the whole approach.

The audience agreed that such an approach could be useful for learning. But the point was made that these concepts have a problem in real life, because most universities lack the technology needed to support this learning style. Another problem is that there are not enough teachers for teaching object technology in small groups. This facts still limit the application of cooperative learning software in the domain of teaching object technology.

4 Main Results

During the presentations workshop participants raised many interesting and challenging questions. These were grouped into four working group topics as follows:

- "Natural" concepts vs. specific languages and tools
  - Is object-orientation a more "natural" approach to (software) development?
  - Should we start with (abstract) concepts or (concrete) languages?
  - Is UML a good representation for (object-oriented) models?
  - Is there any specific object-oriented design or object-oriented knowledge for tool design?
- Scope and limitations of the minimalist approach
  - Are we learning by doing or doing by learning?
  - "Good design comes from experience, but experience comes from bad design" [Bergin]
  - Can we teach software engineering principles by learning by doing?
  - What if learners fear to make errors?
- Objects vs. algorithms vs. concurrency
  - Are objects good for everything?
  - Concurrency - how early?
- Tool support for (cooperative) learning
  - Cooperative learning and exploration - when, where, and how?
  - Meta-learning vs. tool overload - do we learn by using a tool or by learning the tool itself?
  - What should a tool support or not support?

Workshop participants voted for the four topics above and we ended up with three working groups. The working group *Objects Vs algorithms vs. concurrency* could not be filled.

The discussions of the working groups are summarized in the subsections below.

4.1 "Natural" Concepts vs. Specific Languages and Tools

One of the issues this group covered was that of richness of languages and how they support expressing models following different formalisms. The group considered it important for languages to support specification, exploration, development, and reflection. Even though each member of the group had his/her favored (object-oriented) language the group agreed to some extent in the fact that the principles of objects could be taught using any of them. However, quite some time was dedicated to discussing the impacts that particular features of languages have on learning. Discussion topics included

- typed vs. untyped languages,
- concurrency,
- hiding complexity,
- declarative specification,
- genericity,
- metamodeling,
- prototyping, and
- support for experimentation

A big problem in using specific languages is that people tend to stick with the paradigm of the first programs they write. Although language does not matter in general, "just Java" is not enough. Custom designed teaching languages might be one solution to these problems. Another solution would be to change curricula and teach (formal) semantics first to build a common ground. Students can then pick specific languages more easily by themselves.

4.2 Scope and Limitations of the Minimalist Approach

Minimalist approaches assume active learners with a considerable amount of knowledge that they will use it in making sense of new information. Reading is kept to a minimum, things are done instead. The approach is very sensitive to the prerequisite knowledge of the learners and one needs to know the audience very well.

It is still unclear how well minimalism fits group processes. There is a big difference between teaching someone a task that has such well-defined interfaces that it can be carried out in isolation (like writing a paper, where the mechanics of keystrokes and formatting are of no concern to the person providing the dictation nor to the recipient of the letter) and a task that interfaces with other tasks (such as most tasks of software development). That greatly limits the context in which minimalist instruction can be used.

Certification courses or examination are another difficulty. How can we *test* what someone has learned? How can learning be tested? If we are to use minimalist instruction, then we cannot draw testing material from the pedagogical material itself (because there is too little of this). That makes it more difficult to develop meaningful tests. What the minimalist system gains in "natural" mastery of a task it may sacrifice to inconsistency across learners or to lack of objective evaluation criteria drawn from the pedagogy.

4.3 Tool Support for (Cooperative) Learning

The aim of this working group was to reason about educational tools for object-orientation; what should such tools support or not support, how should they be built, and and what pedagogy should be used?

The group elaborated on three topics that cross cut the presentations of all position papers:

- Hiding complexity
- Meta-learning
- Level of tool support

The first issue was addressed by three presentations. The *MagicDraw* library, presented by Kim Bruce, hides more or less complex code inside specific classes. Students are freed from handling repaint loops, listeners, or concurrency. *Karel J Robot* takes this approach even further, but with a narrower application area. Mary Beth's example-based tools (based on the minimalist approach) focus on concrete topics to learn. Details are hidden in layers that can be revealed step by step.

All approaches proved successfully to enable students to manage nontrivial tasks early. The groups main conclusions about complexity hiding can be summarized as follows:

- Leave students in a state of perceived total control, leaving mysteries (like Java's `public static void main(String[] args)`) causes only confusion;
- Build layers on top of existing tools or environments that function as filters and try to gradually introduce all difficult parts;
- Avoid a gap when unhiding. The revealed details should fit with the knowledge gained so far, "old" knowledge should not become invalid;

Even the issue of meta-learning cross cut several presentations. The example-based tools used in minimalist instruction try to minimize meta-learning, whereas the Pico approach, presented by Theo D'Hondt, proposes the opposite. Learners must master the Pico framework in order to learn about object-oriented semantics. The saying that *to really learn a programming language, you have to write an interpreter/compiler for it* points into the same direction.

A problem with meta-learning are its high start-up costs. It is not sure, if these costs will always pay back. In a traditional course environment, with limited schedules, the start-up costs might be prohibitive. Nevertheless, learning by means of a learning environment could mean using an authoring tool to adapt the learning environment itself [7].

As its last issue the group discussed general requirements for learning/teaching tools. Unfortunately the group ran out of time, but could agree on the following list of minimum requirements:

- A tool should be flexible enough to adapt it to different groups of users and application domains;

- It should support different views (of the problem and the solution) on different levels of abstraction;
- It should support syntax-directed editing and enforce good programming style in some way;
- It should support visual programming to make learning more fun.

5 List of Participants

The workshop had 18 participants from 9 countries. Twelve participants came from academia and 6 from industry. All participants are listed in table 2 together with their affiliations and e-mail addresses.

Table 2. Workshop participants

Name	Affiliation	E-mail Address
Isabel Michiels	*Vrije Universiteit Brussel, Belgium*	imichiel@vub.ac.be
Alejandro Fernández	*GMD-IPSI, Darmstadt, Germany*	casco@darmstadt.de
Jürgen Börstler	*Umeå University, Sweden*	jubo@cs.umu.se
Maximo Prieto	*Universidad Nacional de La Plata, Argentina*	maximo@sol.info.unlp.edu.ar
Eleonore Lundström	*Umeå University, Sweden*	eason@informatik.umu.se
Martine Devos	*EDS - Portfolio Development, Strat.*	mdevos@eds.com
Xavier Alvarez	*EMN, France*	xavi@emn.fr
Stephen Eldridge	*UMIST, Manchester, UK*	see@co.umist.ac.uk
Jan Schümmer	*GMD-IPSI, Darmstadt, Germany*	jan.schuemmer@gmd.de
Torsten Holmer	*GMD-IPSI, Darmstadt, Germany*	torsten.holmer@gmd.de
James O. Coplien	*Bell Labs (& VUB), USA*	cope@bell-labs.com
Mary Beth Rosson	*Virginia Tech, USA*	rosson@vt.edu
Theo D'Hondt	*Vrije Universiteit Brussel, Belgium*	tjdhondt@vub.ac.be
Kim Bruce	*Williams College, USA*	kim@cs.williams.edu
Joseph Bergin	*Pace University, USA*	jbergin@pace.edu
Dan Rozenfarb	*Universidad de Buenos Aires, Argentina*	drozenfa@dc.uba.ar
Mikal Ziane	*Laboratoire de l'universite Paris6, France*	Mikal.Ziane@lip6.fr
Laszlo Kozma	*Eötuös Lorand University, Hungary*	kozma@ludens.elk.lu

References

1. Bacvanski, V., Börstler, J.: Doing Your First OO Project–OO Education Issues in Industry and Academia. OOPSLA'97, Addendum to the Proceedings (1997) 93–96
2. Börstler, J. (ed.): OOPSLA'97 Workshop Report: Doing Your First OO Project. Technical Report UMINF-97.26, Department of Computing Science, Umeå University, Sweden (1997) http://www.cs.umu.se/~jubo/Meetings/OOPSLA97/
3. Börstler, J. (chpt. ed.): Learning and Teaching Objects Successfully. In: Demeyer, S., Bosch, J. (eds.): Object-Oriented Technology, ECOOP'98 Workshop Reader. Lecture Notes in Computer Science, Vol. 1543. Springer-Verlag, Berlin Heidelberg New York (1998) 333–362
   http://www.cs.umu.se/~jubo/Meetings/ECOOP98/
4. Börstler, J., Fernández, A. (eds.): OOPSLA'99 Workshop Report: Quest for Effective Classroom Examples. Technical Report UMINF-00.03, Department of Computing Science, Umeå University, Sweden (2000)
   http://www.cs.umu.se/~jubo/Meetings/OOPSLA99/CFP.html
5. Burns, A., Davies, G.: Concurrent Programming. Addison-Wesley (1993)
6. Carrol, J. M.: The Nürnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill. MIT Press, Cambridge (1990)
7. Goldberg, A.: What should we teach? OOPSLA'95, Addendum to the Proceedings. OOPS Messenger **6** (4) (1995) 30–37
8. Manns, M. L., Sharp, H., McLaughlin, P., Prieto, M.: Capturing successful practices in OT education and training. Journal of Object-Oriented Programming **11** (1) (1998)
9. Pattis, R. A.: Karel The Robot: A Gentle Introduction to the Art of Programming, 2nd Edition. Wiley (1995)
10. Stein, L. A.: Interactive Programming in Java. Morgan Kaufmann (2000)
11. Pedagogical Patterns pages.
    http://www-lifia.info.unlp.edu.ar/ppp/
    http://csis.pace.edu/~bergin/PedPat1.3.html
12. LIFIA, Universidad Nacional de La Plata, Argentina
    http://www-lifia.info.unlp.edu.ar/
13. GMD-IPSI, Darmstadt, Germany. http://www.darmstadt.gmd.de/concert
14. European Master in Object-Oriented Software Engineering.
    http://www.emn.fr/MSc/
15. VITAL. http://www.darmstadt.gmd.de/concert/software/vital.html