Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts⋆

Jürgen Börstler[1], Isabel Michiels[2], and Annita Fjuk[3]

[1] Umeå University, Sweden
[2] Vrije Universiteit Brussel, Belgium
[3] University of Oslo, Norway

Abstract. This report summarises the results of the eighth workshop in a series of workshops on pedagogies and tools for the teaching and learning of object-oriented concepts. The submissions to this year's workshop mainly covered curriculum issues, tool support for teaching, and case studies. Several contributions dealt with teaching object-orientation to non-Majors (junior high-school students, non-Science students). This aspect permeated most of the discussions at the workshop and is also reflected in the conclusions. The workshop gathered 19 participants from nine different countries.

1   Introduction

Object-orientation has nowadays become the dominant software development paradigm. In most educational programs object-orientation is the first (and sometimes the only) paradigm students encounter. Successfully applying object-oriented methods, languages and tools requires a thorough understanding of the underlying object-oriented concepts. Despite this importance there is still no accepted approach to effectively teach or learn basic object-oriented concepts.

Studies show that there is a great mismatch between language used and paradigm taught. In Australia for example about 82% of the introductory programming instructors use an object-oriented language. Only about 37% however teach their courses using an object-oriented approach [1]. Most educational publications however suggest to introduce objects early on.

Using traditional programming languages, concepts could be introduced step by step. Abstract and advanced concepts, like for example modules and abstract data types could be handled as an afterthought. In the object-oriented paradigm, the basic concepts are tightly interrelated and cannot easily be taught and learned in isolation, making these tasks much more challenging.

⋆ The title of this report should be referenced as "Report from the ECOOP 2004 Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts".

Switching to object-oriented development is not just a matter of programming languages. Focusing on the notational details of a certain language prevents students from grasping the "big picture." Most students therefore have difficulties taking advantage of object-oriented concepts. Many traditional examples are furthermore not very suitable for the teaching and learning of object-oriented concepts. Many popular examples (like for example 'Hello World') actually contradict the rules, guidelines and styles we want to instill in our students [2].

Educators must therefore be very careful when selecting or developing examples and metaphors. Rules and misconceptions that students develop based on doubtful examples will stand in the way of teachers and learners as well.

This was the eighth in a series of workshops on issues in object-oriented teaching and learning. Reports from most previous workshops in the series are available [3, 4, 5, 6, 7, 8]. Further information and links to the accepted contributions of most workshops can be found at the workshop series home page [9].

The objective of the workshop series is to share experiences and discuss ideas, approaches and hypotheses on how to improve the teaching and learning of object-oriented concepts. The organisers particularly invited submissions on the following topics:

- successfully used examples and metaphors;
- approaches and tools for teaching (basic) object-oriented concepts;
- approaches and tools for teaching analysis and design;
- ordering of topics, in particular when to teach analysis and design;
- experiences with innovative CS1 curricula and didactic techniques;
- learning theories and pedagogical approaches / methods;
- representation of learning resources;
- distance education / net-based learning;
- collaborative learning;
- guiding the learners;
- learners' view(s) on object technology education;
- development of the learner's competence.

2 Workshop Organisation

Participation at the workshop was by invitation only. The number of participants was limited to encourage the building of few small interest groups working on specific topics. Potential attendees were required to submit position papers.

Out of the 11 position papers that were submitted, 9 papers were accepted, of which 7 were formally presented at the workshop. Two papers were rejected. All accepted contributions were made available on the workshop's web site some weeks before the workshop, in order to give attendees the opportunity to prepare for the discussions. All formal presentation activities were scheduled for the morning sessions. The afternoon sessions were dedicated to discussions in small working groups.

After the formal presentations all attendees had the opportunity to present their main "message(s)" in a round-robin presentation (materials had to be

Table 1. Workshop program

TIME	TOPIC
9:00	**Welcome and Introduction**
9:15	**Session 1: Curriculum Issues**
	Why Structural Recursion Should Be Taught Before Arrays in CS 1, *K. Bruce (15 min)*
	Teaching Object-Oriented Programming–Towards Teaching a Systematic Programming Process, *M.E. Caspersen (15 min)*
	Object-Orientation by Immersion–Teaching Outside the CS Department, *M. Lindholm (10 min)*
	Discussion (10 min)
10:05	**Session 2: Tools**
	OCLE, a Tool Supporting Teaching and Learning UML and OCL, the Understanding and Using of Metamodeling, Abstraction and Design by Contract, *D. Chiorean (10 min)*
	Online Assessment of Programming Exercises, *G. Fischer (10 min)*
	Discussion (5 min)
10:30	**Coffee Break**
11:00	**Session 3: Case Studies**
	Junior High School Students' Perception of Object-Oriented Concepts, *M. Teif (15 min)*
	Lego as platform for learning OO thinking in primary and secondary school, *C. Holmboe (15 min)*
	Discussion (10 min)
11:40	**Round-robin Presentations of Attendee Positions (1 overhead each)**
12:10	**Split up into Working Groups**
11:30	**Lunch Break**
13:30	**Parallel Working Group Sessions**
15:00	**Coffee Break**
15:30	**Parallel Working Group Sessions (cont.)**
16:15	**Working Group Reports, Discussion, Wrap-up**
17:00	**Closing**

submitted in advance). After that three working groups were formed to discuss in more detail the topics they found most interesting or relevant. The full workshop program can be found in table 2.

The workshop gathered 19 participants from nine different countries, all of them from academia. A complete list of participants together with their affiliations and e-mail addresses can be found in table 4.2.

3   Summary of Presentations

This section summarises the main points of the presented papers and the most important issues raised during the morning discussions. More information on the presented papers can be obtained from the workshop's home page [10].

Table 2. Overview of the presentations

	Bruce	Caspersen	Lindholm	Chiorean	Fischer	Teif	Holmboe
Target Group							
– CS juniors	X	X			X		(X)
– CS seniors				X			
– Non-CS			X		(X)	X	X
– Pre-Univ.						X	X
Target Area							
– OOP	X	X			X	(X)	(X)
– OO concepts	X	(X)	X	(X)		X	X
– Design			X				
– Modelling		X		X			(X)
– Formal Spec.				X			
– Testing				(X)	(X)		
Paper Type							
– Exp. report	X	X	X	X	X		
– Case study						X	X
Main message	Teach structural recursion before arrays	Teach systematic techniques and processes explicitly	Use sufficiently complex examples from a familiar domain	Tool support is essential for teaching meta-modelling	Automatic assessment is both feasible and effective	Misconceptions don't differ with age	OO is more about a way of thinking as a way of programming

3.1   Curriculum Issues

Kim Bruce (Williams College, MA, USA) described how rearranging topics helped his CS1 students to better understand object-oriented principles, in particular encapsulation. The first half of the course originally took an objects-first, event-driven approach using the `objectdraw` library, developed at Williams College. The students caught on very well to object-orientation using this approach. However, when introducing non-object-oriented structures, like arrays, strings and files in the second part of the course the students do no longer apply the object oriented principles they have learned. Students for example resisted to encapsulate arrays in purposeful objects and returned to directly manipulating arrays and their elements.

Moving the topic of structural recursion from the end of the course to the beginning of the second part of the course (i.e. before introducing arrays), proved to be a major improvement. Kim argued that recursive structures are much easier to understand than recursive procedures. In their course they use recursive pictures, where the base cases are realised as Java interfaces. The recursion is then an implementation of a Picture interface that has an instance variable of the interface type for the recursive part. Such recursive structures are also very useful to reinforce important object oriented concepts like dynamic method dispatch

and interfaces. They also provide a smoother transition to arrays. This revised topic order also helps students to better understand the motives behind explicit encapsulation of arrays (compared to direct manipulation).

The audience found this approach very promising. However, it was not quite clear whether this approach could be used with more mediocre student groups. The concept of structural recursion is not that easy to grasp and even Kim's well designed examples would require a thorough understanding of object-oriented concepts. Kim argued that they hide many language specific details in their library. This makes it easier for the students to concentrate on the important things.

Michael E. Caspersen (University of Aarhus, Denmark) claimed that teaching should focus on systematic techniques to develop programs by means of conceptual modelling and that it is mandatory to train students in the process of applying systematic development techniques. He and his colleagues use a model-driven objects-first approach with a strong focus on systematic techniques and programming processes. In this approach the actual programming is not just a matter of a certain programming language. Students are gradually exposed to more complexity by means of conceptual models and not by means of more complex constructs in a certain programming language.

Systematic (implementation) techniques are taught on different levels. On method level the students deal with algorithm patterns and loop invariants. On class level the students deal with class specifications (contracts). On class structure level the students deal with specification models (class diagrams). This separation of concern makes it easier for students to choose the appropriate technique for the problem at hand. Michael also noted the importance of actually demonstrating how experts use these techniques in practise. The lecturers do a lot of "live" programming and modelling in class, where they say aloud what they do and why the do it in a particular way. They also show videos of actual developers working on real projects.

According to Michael the approach has been very successful. The drop-out rates have decreased from about 50% five years ago to about 10%.

Morten Lindholm (University of Aarhus, Denmark) described his approach to teaching object-oriented concepts to students of the Faculty of Arts. The goal of the two-semester course Programming & Systems Development is to give students a basic understanding of computers and software development. A problem in course development had been to find a suitable textbook for students from the humanities or social sciences. Current textbooks usually aim at students with a Science or Engineering background and are therefore full of problems, examples and exercises that draw on this background. Most solutions (i.e. example programs) lack therefore objects rooted in the "real" world, as for example `System.out`.

Morten suggested to use language philosophy and logic as the foundation for object-orientation and a way to reason about ways of describing (a model of) the world in a machine. To motivate the students examples, assignments, and

projects should always be taken from a domain familiar to the students. There should also be a mapping to the real world in order to grasp what an object is.

The students in this course sequence are exposed to a wide range of techniques to give them a broad view of object-orientation and design (CRC-cards, UML, Pair Programming, Design Patterns, Unified Process). These techniques are then trained by means of team projects. This approach seems to work well except for the actual coding part.

In his talk Morten highlighted the importance of "sufficiently complex" examples[1]. The real power of object-oriented modelling cannot be used in trivial examples. Several attendees choose to disagree here. Micheal Caspersen agreed that examples should not be trivial and, but not more complex than necessary to make their point. He would rather vote for "sufficiently simple" examples. As another important point it was mentioned that every program the students write, should reflect the programs the students use (i.e. no batch-like processing for example).

3.2   Tools

Dan Chiorean (Babes-Bolyai University, Romania) presented OCLE (Object Constraint Language Environment), a tool for the teaching and learning of OCL, UML, metamodelling, and abstraction. Dan presented several examples how the tool can be used to view, evaluate and transform example specifications. OCLE supports modelling on the user model level as well as the metamodel level. By grouping both kinds of models in one project, the user models can then be (statically) validated against the corresponding rules defined in the metamodel.

This kind of parallel modelling on two different levels of abstraction helps the students to appreciate metamodelling and also to better understand the semantics of OCL and UML. The tool can visualise different views of the same information at the simultaneously. This enables students to understand how decisions made at metamodel level affect instances on the user model level.

The tool can even generate (Java) code from OCL specifications. By doing so one can trace the effects of changes in the metamodel down to the generated code. It was however not clear whether the code generation would scale up to more complex examples.

Gregor Fischer (University of Wrzburg, Germany) reported about the usage of Praktomat (see [8]) to automatically assess programming exercises. All attendees agreed that multiple-choice tests are not very reliable for assessing programming skills. However, manually assessing large numbers of exercises in a consistent manner would be very difficult, if not infeasible.

Praktomat is capable of several types of tests; formal (syntax, compiling, coding style and required documentation), program structure, specification requirements, and functional tests. Functional testing is done by means of testing operations written in JUnit. Functional testing against the results of master so-

---

[1] The idea of using *sufficiently complex* examples was originally propagated by Kristen Nygaard using his famous Restaurant example.

lution (working as an oracle) did not work as reliable. This would however put less restrictions on the structure of the solutions that must be delivered by the students.

The tool results are very reliable. Less than 2% of the programs passing all automatic tests were rejected by a human inspector. It was however not clear whether fundamentally good, but imperfect solutions would be rejected by the tool. The examination work load decreased by a factor of four. Developing suitable exercises on the other hand is much more expensive. Gregor did not have actual data on this part, but claimed that there still is a reasonable pay off.

Programming is a skill that requires a lot of training. Automatic assessment can provide the immediate feedback that is necessary to handle large student groups and/ or large numbers of exercises. In the current setting students can perform as many tests as they like before finally submitting their programs. The audience was not sure whether this will teach students bad habits. Nevertheless, since the tool has been introduced Gregor and his colleagues have observed considerable quality improvements in students' solutions. The tool did on the other hand, not affect the course's overall passing rate.

Gregor noted further that currently all test are public. This helped many students appreciate proper testing. For the future it is planned to integrate testing and test development into the course (but how will the tests be tested).

3.3   Pre-university Case Studies

Mariana Teif (Israel Institute of Technology, Israel) presented results from a junior high school course (7th-8th grade) on object-oriented concepts. In the context of a research project, she studied difficulties and misconceptions the students faced while learning basic object oriented concepts. The students developed small programs using a Java-based Turtle-graphics environment. Students did not have to present practical programming skills at the end of the course. Examination was by means of concept-level questions.

The study identified four types of misconceptions. Students did for example confuse an object's attributes with its parts (components) or actions (knowing how to do certain things). They had also difficulties accepting sets as objects and considered set-subset (hierarchy) relationships and whole-part relationships as equal to the class-instance relationship. A follow-up study with undergraduate students as subjects revealed the same results.

Mariana discussed a range of possible explanations for the found misconceptions, which might be based on classification theory. Further research and analysis is planned to explain the underlying reasons for the problems.

Christian Holmboe (University of Oslo, Norway) presented preliminary results from a case study done in the context of the COOL[2] project on teaching primary and secondary school children general principles of object-orientation and computer functionality.

---

[2] **C**omprehensive **O**bject-**O**riented **L**earning

The study was carried out as an intensive four day course, with two different age groups (6th and 9th grade respectively) at different schools. The object-oriented concepts covered were classification, specialisation and aggregation. The students engaged in various types of classroom activities, like drawing, building with Lego, or written textual definitions. The "products" of these activities were discussed with respect to features, attributes and different kinds of relationships to emphasise the usefulness of different kinds of "specification" mechanisms.

In a group project the students then explored and programmed a Lego vehicle using a small Robot API based on LeJoS, a Java-based development language for Lego. The API and an object diagram for the vehicle were carefully explained and set up on the walls of the classroom. The results of the students' group work were discussed in relation to object-oriented principles, but without using strict object-oriented terminology. After that the students did some exercises using the Restaurant example.

The students solved quite interesting tasks after only four days. Interviews after the course (using another example) showed that the students gained quite some understanding of the covered object-oriented concepts. Teaching (object-oriented) design and technology in this way might enhance the students confidence about information technology. This might help to get more people interested in Information Technology and Computer Science. Further studies of the gathered data are however necessary to draw any safe conclusions.

4   Working Group Discussions

For the afternoon sessions participants formed three working groups to discuss specific topics in more detail. The following subsections summarise the discussions on the topics discussed in more detail.

4.1   Balancing Focus Between Process and Product

The discussion was initiated by trying to establish a common understanding of the notions of "process" and "product." A product is a concrete result of a (planned) development activity, like for example the final executable program code, a diagram or a model. A process is the sum of the developers' actions to produce these products. To develop a product we also have to take care of the pragmatics of software development, like for example the usage of tools and the application of certain guiding principles and rules.

In his analogy with cabinet making, David Gries described nicely how the teaching about products, processes and pragmatics should (not) be didactically and pedagogically organised [11]:

> Suppose you attend a course in cabinet making. The instructor briefly shows you a saw, a plane, a hammer, and a few other tools, letting you use each one for a few minutes. He next shows you a beautifully-finished cabinet. Finally he tells you to design your own cabinet and bring him the finished product in a few weeks.

You would think he was crazy! You would want instructions on designing the cabinet, his ideas on what kind of wood to use, some individual attention when you don't know what to do next, his opinion on whether you have sanded enough, and so on. (p82)

One way of achieving a balance between product and process in teaching, is to expose students to how experienced programmers work. Like how a master shows his or her profession to apprentices. Approaches like projector-programming and live-programming in the classroom were considered important by all group members. However, when a teacher demonstrates the process of programming (with help of a projector or the like), she must also think aloud and make typical errors. Programming is not a straight forward process. Students must be reassured that there are no single correct solutions that will unfold automatically when "doing it right."

4.2   Student Diversity and Core Computer Science

Teaching Computer Science to different groups of students rises the question about a thorough definition about the core of our field. What do we need to teach to students who take only few Computer Science courses? The definition of a "real" core (see 1) of Computer Science must necessarily be much narrower for students outside our field compared to the definition as for example proposed by the CC2001 [12].
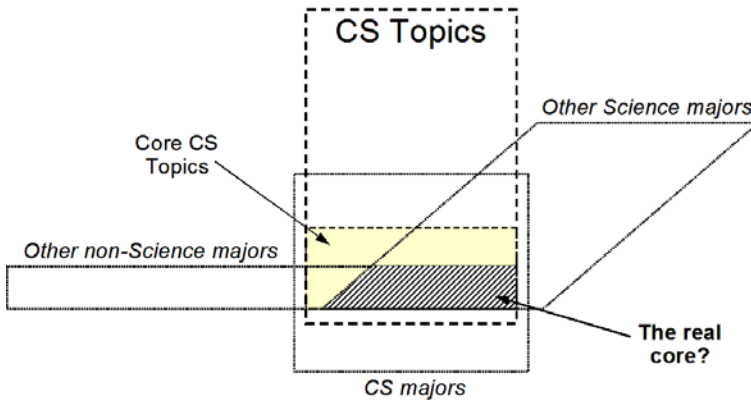

Fig. 1 Points of view of the core of Computer Science

The group concluded that object-oriented programming is not a core topic for students outside our field. However, object-orientation as a way to structure and model problems should be included in some way.

It was also pointed out that different groups of students would need different kinds of "windows" to view the core (see 2). Matlab for example could be a
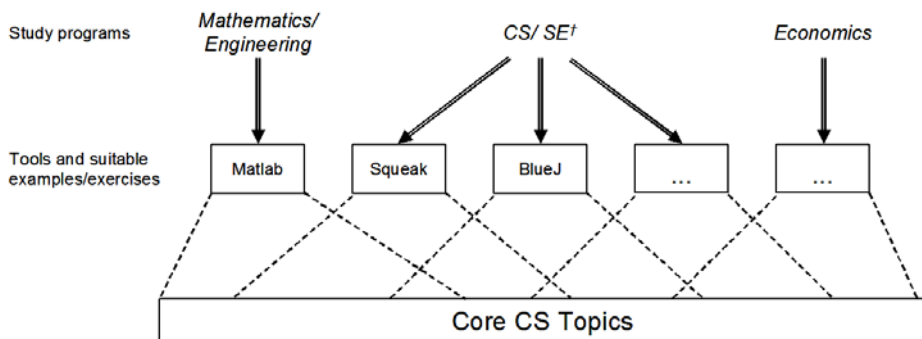
Fig. 2 Windows to approach the core of Computer Science

suitable window for Math or Engineering students to approach Computer Science. For all windows and groups it would be necessary to provide suitable examples.

Table 3. List of workshop participants

Name	Affiliation	E-mail Address
Jrgen Brstler	*Ume University, Sweden*	jubo@cs.umu.se
Isabel Michiels	*Vrije Universiteit Brussels, Belgium*	imichiel@vub.ac.be
Annita Fjuk	*University of Oslo, Norway*	annita.fjuk@telenor.com
Jens Kaasboll	*University of Oslo, Norway*	jensj@ifi.uio.no
Jens Bennedsen	*IT University West, Denmark*	jbb@it-vest.dk
Maria Bortes	*Babes-Bolyai University, Romania*	maria@lci.cs.ubbcluj.ro
Kim Bruce	*Williams College, MA, USA*	kim@cs.williams.edu
Michael E. Caspersen	*University of Aarhus, Denmark*	mec@daimi.au.dk
Dan Chiorean	*Babes-Bolyai University, Romania*	chiorean@lci.cs.ubbcluj.ro
Dyan Corutiu	*Babes-Bolyai University, Romania*	dyan@lci.cs.ubbcluj.ro
Gregor Fischer	*University of Wrzburg, Germany*	fischer@informatik.uni-wuerzburg.de
Roar Granerud	*University of Oslo, Norway*	rgraneru@ifi.uio.no
Arne-Kristian Groven	*Norwegian Computing Centre, Norway*	groven@nr.no
Hvard Hegna	*Norwegian Computing Centre, Norway*	hegna@nr.no
Christian Holmboe	*University of Oslo, Norway*	christho@ifi.uio.no
Morten Lindholm	*University of Aarhus, Denmark*	lindholm@daimi.au.dk
Wilfried Rupflin	*University of Dortmund, Germany*	wilfried.rupflin@uni-dortmund.de
Mariana Teif	*Israel Institute of Technology, Israel*	tmariana@tx.technion.ac.il
Kristina Vuckovic	*Zagreb University, Croatia*	kvuckovi@ffzg.hr

Most importantly examples need to be appealing to students and interesting in itself, but not necessarily depending on context (not everything needs to be real world relevant). Besides that examples must fulfill certain criteria to be meaningful with respect to object-orientation. They should for example comprise at least two classes, where at least one class has multiple instances. There should furthermore happen some real object interaction, where different objects need to (get to) know each other. Examples should also make use of non-static methods and exchange objects as parameters (instead of built-in types only). In addition to that custom designed libraries to hide language specific details would help students to focus on the real problems.

4.3   Student Attitudes

Student attitudes was another topic discussed in this group. The audience agreed that students are not as good as before in solving problems on their own. They quickly give up on exercises and they often rely on help from others when they are stuck with a certain problem. Possible solutions could be (as proposed by Michael Caspersen) to show students the process of working towards a reliable solution. However, it was argued that this is not sufficient. Teachers often ask themselves questions as part of this (implicit) process in specific orders. When this process is not made explicit to students, they do not know how to proceed once they are on their own. The group concluded that we definitely need to teach more programming methodology. This must however be done in a systematic way to enable the students to choose the "right" tool for the right problem. To help students build up confidence some kind of peer-based *help to help yourself* approach, like Supplemental Instruction, can also be very useful.

5   Conclusions

The objective of the workshop was to share experiences and to discuss ideas, approaches and hypotheses on how to improve the teaching and learning of object-oriented concepts.

Ongoing work was presented by a very diverse group of people with very different backgrounds, which resulted in a broad range of topics, see table 3. The workshop attendees represented a broad range of experiences, which are often not easily accessible and transferable. We can definitely conclude that there are many approaches that can work in certain environments. We can also certainly conclude that there is no is no one-size-fits-all approach to teaching object-oriented concepts and principles. The challenge is to find the right approach (or view on things) for each type of student.

To summarise our discussions we want to suggest the following.

- We must expose students to the real programming process, instead of an idealised linear one. Software development is an inherently incremental process. Students must be made aware of that there is no single correct solution to

virtually any programming problem. There is not even a single correct process that will "automatically" reveal a correct solution. This is even more important when students are faced with analysis and design problems.

– Students need "tools" to help them escape when they get stuck on their own, instead of waiting for someone to help them out. The teaching of systematic techniques and explicit processes can help a lot. But, especially weaker students, also need help to improve their self-confidence, for example by means of supplemental instruction.

– Carefully developed examples and exercises are very important. They must be designed with the target group of students in mind. They must also be valid and meaningful with respect to the object-oriented paradigm.

– Object-oriented programming is not a given core subject for non-Majors in Computer Science. However, basic object-oriented concepts or principles, like encapsulation and abstraction, should certainly be taught to any student to enable them to cope with complexity.

References

1. de Raadt, M., Watson, R., Toleman, M.: Introductory Programming: What's Happening Today and Will There Be Any Students to Teach Tomorrow? In: Lister, R., Young, A.: Proceedings ACE2004, Conferences in Research and Practice in Information Technology, Vol. 30, Australian Computer Society (2004) 277-282.

2. Westfall, R.: "Hello, World" Considered Harmful. Communications of the ACM **44** (10) (2001) 129-130.

3. 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).

4. 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 (1998) 333-362.

5. 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).

6. Michiels, I., Börstler, J.: Tools and Environments for Understanding Object-Oriented Concepts. In:Malenfant, J., Moisan, S., Moreira, A. (eds.): Object Oriented Technology, ECOOP 2000 Workshop Reader, Lecture Notes in Computer Science, Vol. 1964. Springer (2000) 65-77.

7. Michiels, I., Börstler, J., Bruce, K.: Sixth Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts. In: Hernández, J., Moreira, A. (eds.): Object Oriented Technology, ECOOP 2002 Workshop Reader, Lecture Notes in Computer Science, Vol. 2548, Springer (2002) 30-43.

8. Michiels, I., Börstler, J., Bruce, K., Fernndez, A.: Tools and Environments for Learning Object-Oriented Concepts. In: ECOOP 2003 Workshop Reader, Lecture Notes in Computer Science, Vol. 3013, Springer (2004) 119-129.

9. Workshop series homepage: Pedagogies and Tools for the Teaching and Learning of Object Technology. `http://www.cs.umu.se/research/education/ooEduWS.html`

10. ECOOP04 workshop homepage. `http://www.cs.umu.se/~jubo/Meetings/ECOOP04`
11. Gries, D.: What Should We Teach in an Introductory Programmming Course. Proceedings of the fourth SIGCSE Technical Symposium on Computer Science Education. ACM SIGCSE Bulletin **6** (1) (1974) 81-89.
12. CC2001 Task Force: Computing Curricula 2001-Computer Science. Final Report (2001).