ECOOP 2002 Workshop Report: Sixth Workshop on Tools and Environments for Learning Object-Oriented Concepts

Organizers
Isabel Michiels[1], Jürgen Börstler[2] and Kim B. Bruce[3]

Edited by
Isabel Michiels, Jürgen Börstler and Kim B. Bruce

[1] PROG, Vrije Universiteit Brussel, Belgium
[2] Umeå University, Sweden
[3] Williams College, Massachusetts, USA

Abstract. The objective of this workshop was to discuss current techniques, 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.
This was the sixth workshop in a series of workshops on learning object-oriented concepts.

1 Introduction

The primary goal of learning and teaching object-oriented concepts is to enable people to successfully participate in an object-oriented development project.
Successfully using object-oriented technology requires a thorough understanding of basic OO concepts. However, learning these techniques, as well as lecturing about these concepts has proven to be very difficult in the past. Misconceptions can occur during the learning cycle and the needed guidance cannot always be directly provided.
The goal of this workshop was to share ideas about innovative teaching approaches and tools to improve the teaching and learning of the basic concepts of object technology rather than teaching a specific programming language. Teaching tools could be either tools used in environments or specific environments for learning OO, as well as any kind of support for developing OO 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:

- approaches and tools for teaching design early;
- intelligent environments for learning and teaching;
- frameworks/toolkits/libraries for learning support;

– microworlds;
– different pedagogies;
– top-down vs. bottom-up approach;
– design early vs. design late;
– topic presentation issues;
– frameworks/toolkits for the development of teaching/learning applications .

This was the sixth 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], OOPSLA'99 [4], ECOOP'00 [5] and OOPSLA '01 [13], and focused on project courses, classroom examples and metaphors, and tools and environments.

2 Workshop Organization

This workshop was designed to gather educators, researchers and practitioners from academia and industry who are working on solutions for teaching basic object-oriented concepts. 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), two working group sessions (afternoon) and a wrap-up session, where all working groups presented their results. The presentations were split up in a long presentation session, where more elaborate papers with demonstrations were presented, and a short presentation session, where the other participants could present their position about the workshop topic. Table 1 summarizes the details of the workshop program.

To gather some input for the working group sessions participants were asked before the workshop to think about some controversial topics in teaching object-oriented concepts.

3 Summary of Presentations

This section summarizes the main points of all workshop presentations. More information on the presented papers can be obtained from the workshop's home page [14].

Glenn D. Bank (Lehigh University, USA) gave an overview on the usage of the multimedia framework CIMEL to supplement computer science courses. To show how CIMEL supports learning-by-doing by means of interactive quizzes and constructive exercises, Glenn gave a short demonstration of a module on Abstract Data Types (ADTs). The module contained advanced multiple-choice questions, where help and feedback is provided by multimedia personae. Furthermore the module supports the successive construction of a concrete ADT by pointing-and-clicking. A case study with 72 students showed that the multimedia contributes significantly to objective learning and helps students design ADTs

Table 1. Workshop program

Time	Topic
9.25 am	WELCOME NOTE
9.30 am	Teaching Abstract Data Type Semantics with Multimedia presented by Glenn D. Blank
10.00 am	Thinking in Object Structures: Teaching Modelling in Secondary Schools presented by Carsten Schulte
10.30 am	Contract-Guided System Development by Vasco Vasconcelos
11.00 am	COFFEE BREAK
11.30 am	Supporting Objects as An Anthropomorphic View at Computation or Why Smalltalk for Teaching Objects? presented by Stephane Ducasse
11.45 am	Extreme Programming Practice in the First Course presented by Joseph Bergin
12.00 am	Teaching Encapsulation and Modularity in Object-Oriented Languages with Access Graphs presented by G. Ardourel
12.15 am	A New Pedagogy for Programming presented by Jan Dockx
12.30 am	Teaching Constructors: A Difficult Multiple Choice presented by Noa Ragonis
12.45 am	PIIPOO: An Adaptive Language to Learn OO Programming presented by R. Pena
1.00 pm	LUNCH BREAK
2.30 pm	A Measure of Design Readiness: Using Patterns to Facilitate Teaching Introductory Object-Oriented Design presented by Tracy L. Lewis
2.45 pm	Teaching Object-Oriented Design with UML - A Blended Learning Approach by Ines Grützner
3.00 pm	Split into working groups
3.05 pm	First Working group session
4.00 pm	COFFEE BREAK
4.30 pm	Second Working group session
5.30 pm	Wrap-up session

to solve a problem. Currently modules on Inheritance and ADTs are available. Materials for a CS0 course are under development. Glenn noted as a drawback that designing new multimedia lectures is expensive. Topics must therefore be chosen carefully to make its use cost effective.

Carsten Schulte (University of Paderborn, Germany) reported on results from the LIFE3 projects, which investigates teaching concepts for object oriented programming in secondary schools. He proposed an apprentice-based learning approach that combines top-down and bottom-up teaching techniques with active learning. UML is used as a visual programming/ modelling language. Their approach is supported by CRC card modelling and FUJABA, an environment that supports static and dynamic modelling using UML ([KNNZ 00]). Consistent models can be directly executed from within FUJABA by means of complete Java code generation. This allows teachers and learners to concentrate on object-oriented concepts and the modelling of objects and their interactions. Executable models are used early on to support active learning. Carsten mentioned the ab-

sence of source code as the major advantage of their approach. Students learn to think in object structures and are able to communicate design/ modeling ideas in terms of objects and their interaction. They *talk* objects and concepts, not code. Preliminary results from an empirical study are very promising.

Vasco Vasconcelos (University of Lisbon, Portugal) reported on a three-course sequence focusing on design-by-contract and quality (here: correctness). Each course is accompanied by a group project. Pre- and postconditions are embedded into Java code (@pre/@post) and tools are used to generate Java code monitoring the assertions.
In their first course students are introduced to the basic concepts of imperative languages, plus objects and classes. Their second course focuses on (formal) correctness proofs. Object-oriented analysis and design, inheritance and polymorphism are delayed until the last course in the sequence.
The main drawback of this approach is that students have difficulties in doing analysis and design for problems involving more than a couple of entities. However, students are now able to reason about algorithms within the context of more complex systems than before. Vasco noted that students run quite early into limitations of the specification language. He also highlighted the importance of tools to monitor assertions while programs are running. The tools currently available are mostly immature and not aimed at undergraduate students.

Stéphane Ducasse emphasized in his talk that a language for teaching object-oriented programming should support the anthropomorphic metaphor promoted by this paradigm. He demonstrated that all the cultural aspects of the Smalltalk language, i.e., the vocabulary and the syntax, fully support the object metaphor of objects sending and receiving messages. The syntax of Smalltalk allows one to read code as a *natural* language.
In addition, he stated that also the programming environment should support the metaphor. He showed that Smalltalk environments offer an important property they named *liveness* or *object proximity* that promotes the anthropomorphic perception of objects. By providing excerpt from a forth coming book, he showed how Squeak with the Morphic framework reinforces this ability to perceive objects as living entities [15, 16].

Joseph Bergin (Pace University, USA) discussed the applicability of eXtreme Programming (XP) practices in introductory computer science courses. His interests were not in XP per se, but in good practices and pedagogics to improve teaching.
His experiences show that most XP practices can be applied in some form in introductory courses. However this requires some changes to course management. Teachers must encourage students to work together (pair programming) and provide assignments that can be developed incrementally, embracing correction and re-grading (small releases, continuous integration and refactoring). Furthermore the teacher must be available all day for questions for example by means of an interactive web site (on-site customer). Planning is supported by time recording in small notebooks á la PSP (planning game). Test first programming is sup-

ported by the tool JUnit [12], which is introduced at the beginning of the course. Other XP practices are more straightforward.

Gilles Ardourel started his talk by pointing out several issues one must face when teaching object-oriented languages: they are evolving fast, provide only informal documentation and can be quickly obsolete. He believes that even when teaching a specific language, you can prepare your students to changes and give them a broader view on languages by using language-independent notations to describe language mechanisms.
The author encountered these issues when teaching access control mechanisms in statically typed class-based languages. These mechanisms manage implementation hiding and define interfaces adapted to different client profiles. At his university, they use access graphs as a language-independent notation for teaching encapsulation and modularity in object-oriented languages.
In access graphs nodes represent classes or a set of classes, and arrows (labelled by sets of properties) represent accesses from one node to another. They explained code examples with *graphs of allowed accesses in a class hierarchy*, then they used *graphs of accesses allowed by an access control mechanism* to describe the mechanisms.
This notation has shown to provide a clear and unambiguous view on access control, and helped students in understanding the mechanisms. They discussed the definitions of the mechanisms and the implication on their code, asked for comparisons with other languages, and provided more feedback. They learned access control easily with some examples about implementation hiding, modularity or substitutability.
Finally, he concluded by stating that he advocates the language-independent description of OOPL mechanisms (thus introducing language design topics in a programming course) and the use of visualization techniques for improving OO teaching.

Jan Dockx explained a new pedagogy for programming used at the Catholic University of Louvain, Belgium. The course is mainly based on good Software Engineering Practices as discussed in the contract paradigm by B. Meyer, extended with behavioral subtyping. Java is used as the example language, but the presenter emphasized that object-oriented concepts are far more important and independent of any given language. In the past, they had a more traditional approach, with first a course on the science of programming, followed by an algorithms and data structure course.

Noa Ragonis' talk started by addressing the difficulties that arise when teaching constructors in object-oriented programming (OOP). She presented different structures of declarations for constructors, their semantic context, their influence on programming, and aspects of students' comprehension. They found that the version of declaring a constructor by initializing all attributes from parameters is preferred, even though it seems difficult to learn. Other *simpler* styles caused serious misconceptions with the students.
She pointed out that instantiation is a central concept because it parallels existence of an object in the real world, a metaphor that is often used in teaching.

Constructors and instantiation are complex concepts, which are difficult to learn and teach, but we can't avoid them and talk about objects without talking about their instantiation. A further complication arises in the instantiation of a class that includes attributes of another class.
From their experience and research in teaching OOP to novices the authors found that using the professional style of declaring a constructor to initialize attributes from parameters, allows them to emphasize the following good OOP principles:

- The constructor is very important, so it is better to expose it than to conceal it,
- Initializing an object with values for its attributes is more in accordance with the real world,
- Creation of simple objects before creation of composite class objects is also more in accordance with the real world,
- Initializing values to attributes in the constructor method avoids access to default values,
- In OOP, you have to learn parameters very early (for mutators), so the parameter mechanism in these paradigms is not an extra demand,
- Assigning values to parameters in each instantiation emphasizes the creation of different objects with the same or different values,
- Learning this constructor pattern in simple classes makes it easier to understand composite classes, and to understand that their attributes are the objects and not the attributes of the objects.

Rosalia Pena talked about their use of a pedagogical language, called PIIPOO, that is used throughout the curriculum for teaching any programming paradigm, thus a language that evolves when teaching another paradigm. Their aim is to minimize the preoccupation of the student with semantic/syntactic details that demand a new language study while one is acquiring the tools of the new paradigm. This evolution of the pedagogical language reinforces the diachronic conception of the programming constructors, providing continuity. So, PIIPOO keeps some programming constructors from Pascal, is OO compatible, adapts and/or removes other constructors, and incorporates new ones required for the OO paradigm.
Moreover, their language drives the novice to use properly the OO environment, avoiding misunderstandings. PIIPOO undertakes so few syntactic and semantic changes as possible to deal with the new way to tackle problem solving,allowing a better concentration on concepts first.
The presenter concludes that once the students' mind is set on the OO world, it is easier to undertake the study of a commercial language, to understand its peculiarities and have a better understanding of the language characteristics.

Tracy Lewis talked about a research program to tackle the problem of teaching introductory object-oriented design. A design readiness aptitude test has been developed to measure the cognitive state where one is able to understand design abstractly. The idea is (this was still work in progress) that an instrument will

be developed for gradually discussing design decisions using programming and design patterns, based on the level of the design readiness measurement. The pedagogy is rooted in learning-by-doing and based on minimalist instruction, constructivism and scaffolded examples.

Ines Grützner (Fraunhofer IESE, Germany) proposed a blended-learning approach for on-the-job training, intermixing traditional classroom education with e-learning approaches. Using traditional classroom education only causes problems because of often tight project schedules, short development cycles and a heterogeneous audience. Pure e-learning approaches on the other hand, lack social communication and expert guidance. Furthermore, developing e-learning courses is often quite expensive.
In the proposed blended-learning approach, online courses are used in the beginning of a training period in order to bring all trainees to a common knowledge and skills level. Traditional classroom education can then be used for teaching advanced concepts, as well as for performing group work and practical exercises. A transfer program developed according to this blended-learning approach consists of the following steps:

- Kick-off meeting of all participants, their teachers, and tutors
- Online learning phase to provide knowledge and skills in applying UML
- Traditional course on object-oriented design with UML
- Final project work

The approach has been used in training developers and managers in using the Unified Modeling Language (UML).
Results show that the approach solves typical problems of both classroom and online education. By using online-courses in pre-training phases it can be assured that all participants have achieved a minimum experience level before the classroom training starts. As a consequence, the duration of classroom training can be shortened. Social communication is supported, since trainees already know each other as well as their trainers from the pre-training phases.
She concluded that both approaches have their strengths and weaknesses but the synergy effects when used in combination clearly outweigh the isolated benefits of the approaches.

4 Discussion

Before the workshop, participants were asked to think about challenging, controversial topics they wanted to discuss. This resulted in an interesting discussion by email just before the workshop. Some of these points will be presented in this section, as well as some introductory controversial topics presented by the organizers of the workshop. At the workshop, we organized a vote for selecting 3 subjects to discuss in 3 work groups.

4.1 Controversial topics

To start up a lively discussion, Kim and Jürgen had prepared a few controversial topics, which are briefly presented below:
Inheritance considered harmful
Kim started by pointing out the keys to a basic understanding of objects: state, methods and dynamic dispatch. This means that class definition and use as well as method invocation need to be explained early on. The explanation of dynamic dispatch (inclusion polymorphism) however should not rely on sublassing. Subclassing involves many new concepts like protection mechanisms, constructors and the 'super' construct, etc and is much too complicated for beginning students to grasp. Therefore should subclassing not even be mentioned early on.
In Java for example interfaces support dynamic dispatch as well and are much *cleaner* a concept than general subclasses. In fact, subclassing boils down to code reuse, which is normally taught at a later stage in the curriculum, and it is pretty complicated. We should therefore avoid introducing inheritance at the very beginning.
No magic please
Jürgen discussed principles for successful early examples. The traditional "HelloWorld" example has been critisized a lot lately [19]. But "HelloWorld" and its variations are not only bad examples of object oriented programs, they are also bad examples by means of the "no magic" measure. With "magic" we refer to examples or topics that are made more complex than necessary, for example by involving several new and possibly interelated concepts. Such examples involve for example language idiosyncrasies, like public static void main (String[] args) and the usage of overly complex library classes (e.g. input handling) early on. No concepts must be introduced using flawed examples, for example exemplifications using exceptions to general established rules. Java strings for example are not real objects, since String objects cannot be modified. The main method is not a real method, since there are no messages sent to it and its parameters seem to be supplied by superior forces. This list can be made much longer and educators have to think twice before presenting seemingly simple examples.

Just before the workshop we launched an e-mail discussion to get some input for the planned working group sessions. The following two sections are organized around two statements that caused quite heated discussions.

The object-oriented paradigm should be taught from the very beginning
Stéphane argued against this statement, since students might not get the full picture of what programming is about. There are situations where other paradigms might be better suited. With Java however you can do "everything," so why bother about other paradigms. He proposed to use a functional language like Scheme to start with. Joe opted that the possibility of designing a curriculum from scratch is not a freedom that every teacher has. At his university, this is certainly the case, so he can only use Java and he is doing the best he can. When the discussion went into the details of pedagogical/academic examples,

Jan reacted that students should be confronted with the "real stuff" instead of "playing with turtles" (using Squeak). He claimed instead that programming in the small is no longer relevant. More focus should be put on design issues instead of pure programming concepts. He questions whether it is really necessary to have a *more intuitive* first programming course.
It doesn't matter which language you use to teach object-orientation
Stéphane disagreed completely. He strongly believes that for teaching OO, the primary aspect that a language should support is the anthropomorphism with the OO paradigm and simplicity. Joe agreed on this, but he argued that you shouldn't teach procedural programming as a prelude to OO. Procedural is not a good first paradigm mainly because it is tied too much to a certain machine model that is not essential and that builds bad habits of thought. Furthermore it is harder to unlearn something than to learn a new concept from scratch. Rosalia argued that many students actually have a procedural background, whether you like it or not. She feels that procedural and OO are just different ways to solve problems, and both must be faced anywere, so between procedural/OO or OO/procedural she would choose the first option. Joe disagreed by saying that a procedural mindset actually hinders students from thinking OO. The more succesfully you teach the procedural approach, the harder it will be for students to learn OO later.

As a summary from the presentations, discussions and the email comments, we came up with the following list of working group topics:

- Is programming in the small still relevant?
- Diachronical walk through the paradigms.
- Problems of large system development.
- Definition of a student-oriented curriculum and of a student-friendly presentation of topics.
- What does objects-first mean?
- When and how should we introduce the main method for novices?
- Programming is a skill (learned through apprenticeship with a master) not a science (which can be studied). Should this change?
- The role of visual presentation (UML).
- Extreme programming in the first course?
- Teaching assertions in the first course?

These questions were discussed under the following headers: Can XP practices be taught in the first course, Should assertions be taught in the first course and What does objects-first really mean?

4.2 Can XP practices be taught in the first course?

The aim of this discussion group was to examine to which extent the 11 key ideas of XP could be used beneficially in early computer science courses.
Out of the 11 key ideas mentioned above, the group picked out 4 of the ideas and discussed their relevance for education:

- testing: Testing is considered to be very important and it can be seen as a form of specification, but less formal. We also believe that writing test themselves, the student learns a lot about the code part that he/she is writing the test for, so writing tests makes students think about the(ir) code. Using Tools like Junit or Sunit for student assigments could be beneficial for students, although someone made the remark that Junit tests are very hard to read.
- the Planning Game: Students need to see how to plan their time for finalizing a student project, because they are not mature enough to do this themselves an important issue to learn because this fits well with the idea of small releases in companies.
- pair programming: Pair programming can be a big help for students which also helps in developing social skills. However, a serious lack of social skills to begin with can form a real problem for the success of pair programming.
- small releases: goes together with the planning game. It is important as a skill, but also as a teaching strategy. We need to help students for finding a plan for their projects to enable small releases, depending on the level of maurity of the student (first year or last year student for example)
- refactoring: We agreed that there is a contradictory part here: XP always supports the saying *if it isn't broken, don't fix it*, but if you don't do it, your code becomes dirty. Students should at least be taught to clean up their code. But when do you clean up your code, at the end of a small release, or at the beginning of the next?

We certainly believe that these XP practices can make a significant contribution in learning object-oriented concepts. There is an upcoming half-day workshop on this topic at OOPSLA 2002 [18].

4.3 Should assertions be taught in the first course?

The second working group discussed the objects-first approach to CS 1 as well as the use of assertions in CS 1.

Objects-First It was agreed that good scaffolding is essential to an objects-first approach because students need good first examples. Ideally one would like to arrange examples so that all parameters, instance variables, etc., are themselves objects. It is useful to avoid primitive types initially as much as possible. It was suggested that if you want to do objects-first, Smalltalk is good, because everything is an object. In discussing the use of students acting out roles as objects, it was pointed out that a difficulty with such an approach to objects-first is that students don't always follow scripts! It may require the presence of a referee to get them to behave. The group also discussed particular tools as an aid to the first few weeks of an introductory course. Blue Jay may help avoid the magic of static void main, etc., when starting, but it was felt that the Blue Jay developers may need to provide more examples. Karel may also be good environment for starting. Girls seem to enjoy it as much as boys.

Assertions in the First Course The use of assertions in the first course was controversial. To be successful, it was felt that students need to read lots of

assertions before they are ready to write their own. It was also felt that in many cases quantifiers are needed in order to really express assertions. Yet these cannot be part of the language (e.g., Eiffel, Java), and thus must be simply treated as special comments, with little or no language support. Another criticism was that methods in the first course are often too simple to have meaningful assertions, though there was general agreement that assertions can be useful in preparation for writing loops, and they often help in understanding boundary conditions. In brief, an invariant essentially specifies the loop. However, the concern of many participants was why introduce a topic in CS 1 that is a stretch when there are already too many topics!

4.4 What does objects-first really mean?

This group started with the question *What is a student-oriented curriculum?* The possible answer might be that a student-oriented curriculum/course is one where the students are prepared to fit the course (instead of the other way around). To develop a student-friendly presentation you need to know your audience. The more uniform your audience and the better you know your students, the easier is it to find the "right" level of presentation.
They went on to discuss teaching approaches/materials. Participants pointed out that there is an apparent lack of interesting, well-designed and well-coded examples. Students get to see very few *exemplary* non-trivial examples. On the other hand, it is very difficult to use large examples, since students do not like to work with existing programs. Most of them have difficulties giving up control and rely on existing code, especially when the code was developed by other students. Scaffolding (as proposed in Tracy's approach) is an essential technique to cope with these problems.
The group also liked the idea of using metaphors for teaching purposes and spent some time trying to find/define a few. However, it turned out to be more difficult than expected to find convincing metaphors.

5 Conclusions

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.

Ongoing work was presented by a very diverse group of people with very different backgrounds, which resulted in a broad range of topics like tool support, environments, courses, teaching approaches, languages for teaching, etc...

Summarizing what was said in the debate groups, we conclude that :

- we want to first focus on teaching the concepts, learning novices how to *think* in an object-oriented way,

– the controversial topic *inheritance considered harmful* didn't appear to be so controversial after all since everyone agreed that inheritance shouldn't be the first topic taught in a computer science course,
– using XP practices for a computer science course might have significant benefits, especially testing and small releases. We agreed that students should be taught to write tests, and we agreed that by writing tests themselves, they learn a lot about the software they are writing the tests for as well,
– building on the objects-first approach, we agreed that good first examples are essential and ideally that the parameters and instance variables we use are all objects. This led us to propose using a pure object-oriented language like Smalltalk for teaching purposes since it fully mirrors the object paradigm of objects sending messages to other objects,
– using assertions was controversial: although we agreed that using assertions could be useful in preparation for topics, like writing loops, people agreed that it takes a lot of time for students being able to use them properly and that it is best therefore to concentrate first on the list of more important object-oriented concepts.

6 List of Participants

The workshop had 20 participants from 12 countries. Eighteen participants came from academia and only two from industry. All participants are listed in table 2 together with their affiliations and e-mail addresses.

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. I. Michiels, J. Börstler: Tools and Environments for Understanding Object-Oriented Concepts, ECOOP 2000 Workshop Reader, Lecture Notes in Computer Science, LNCS 1964, Springer, 2000, p. 65-77. `http://prog.vub.ac.be/~imichiel/ecoop2000/workshop/`
6. Burns, A., Davies, G.: Concurrent Programming. Addison-Wesley (1993)
7. Goldberg, A.: What should we teach? OOPSLA'95, Addendum to the Proceedings. OOPS Messenger **6** (4) (1995) 30–37

Table 2. Workshop participants

Name	Affiliation	E-mail Address
Isabel Michiels	*Vrije Universiteit Brussel, Belgium*	imichiel@vub.ac.be
Jürgen Börstler	*Umeå University, Sweden*	jubo@cs.umu.se
Kim Bruce	*Williams College, USA*	kim@cs.williams.edu
Rosalia Peña	*University Alacala Henares, Madrid, Spain*	rpr@uah.es
Khalid Azim Mughal	*University of Bergen, Norway*	khalid@ii.uib.no
Laszlo Kozma	*Eötuös Lorand University, Hungary*	kozma@ludens.elk.lu
Jan Dockx	*Katholieke Universiteit Leuven, Belgium*	Jan.Dockx@cs.kuleuven.ac.be
Vasco Vasconcelos	*University of Lisbon, Portugal*	vv@di.fc.ul.pt
Ines Grützner	*Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany*	gruetzne@iese.fhg.de
Gilles Ardourel	*LIRMM, Montpellier, France*	ardourel@lirmm.fr
Carsten Schulte	*University of Paderborn, Germany*	carsten@uni-paderborn.de
Tracy Lewis	*Virginia Tech, Virginia, USA*	tracyL@vt.edu
Joe Bergin	*Pace University, USA*	jbergin@pace.edu
Noa Ragonis	*unizmann institute of Science, Nettovot, Israel*	
Glenn Blank	*Lehigh University, Betlehem, PA*	glenn.blank@lehigh.edu
Martine Devos	*Avaya Research, USA*	mmdevos@avaya.com
Stéphane Ducasse	*Software Composition Group, University of Berne, Switzerland*	ducasse@iam.unibe.ch
Kristen Nygaard	*Department of Informatics, University of Oslo, Norway*	kristen@simula.no
Boris Mejias	*Vrije Universiteit Brussel, Belgium*	bmejias@vub.ac.be
Andres Fortier	*Universidad Nacional de La Plata, Argentina*	andres@sol.info.unlp.edu.ar

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. Stein, L. A.: Interactive Programming in Java. Morgan Kaufmann (2000)
10. Pedagogical Patterns pages. `http://www-lifia.info.unlp.edu.ar/ppp/` `http://csis.pace.edu/~bergin/PedPat1.3.html`
11. European Master in Object-Oriented Software Engineering. `http://www.emn.fr/MSc/`
12. JUnit home page. `http://www.junit.org`

13. OOPSLA01 workshop. http://www.cs.umu.se/%7Ejubo/Meetings/OOPSLA01/
14. ECOOP 2002 Workshop homepage. http://prog.vub.ac.be/ecoop2002/ws03/
15. Squeak homepage. http://www.squeak.org,
16. http://www.iam.unibe.ch/~ducasse/WebPages/NoviceProgramming.html
17. Köhler, H.J., Nickel, J, Niere, J., Zündorf, A.: Integrating UML Diagrams for Production Control Systems, Proceedings of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, June 2000, pages 241-251.
18. OOPSLA '02 Workshop on Extreme Programming Practices in the First CS1 Courses. http://csis.pace.edu/~bergin/XPWorkshop/ http://www.oopsla.org
19. Westfall, R.: Hello, World Considered Harmful. Communications of the ACM **44** (10), Oct 2001, 129-130.