

Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts

A Short Summary

Jürgen Börstler

Umeå University,
Sweden

jubo@cs.umu.se

Kim Bruce

Williams College, MA,
USA

kim@cs.williams.edu

Isabel Michiels

Vrije Universiteit
Brussels, Belgium

Isabel.Michiels@vub.be

1 Introduction

Object-Oriented Technology (OOT) has become a major topic in most computer science curricula and most universities now teach OOT from the very beginning. This change has consequences for many subsequent courses. It is therefore necessary to carefully evaluate when and how to introduce the basic concepts of OOT to keep ripple effects of changes in computer science curricula at a minimum.

Many aspects must be considered to help students acquire object-oriented thinking besides working in a particular object-oriented programming language. For example whether to focus on the syntax of a specific programming language or the underlying concepts of the paradigm or whether to focus on programming, design issues, or the structure and mechanisms of existing applications.

With the imperative paradigm there was no need to introduce high-level and abstract structures, like modules and abstract data types early on. Students started at statement level and successively built up simple (monolithic) programs. High-level program structures could be handled as an afterthought. This is very different in the object-oriented paradigm, where several related concepts have to be handled early on (variable, value, type, object, and class). Even the most simple examples and exercises will involve most of these concepts. Only when students have become familiar with the basic concepts and principles of the paradigm, they can concentrate on the notational details of a certain object-oriented programming language. Otherwise they will not grasp the ‘big picture’ and have difficulties to take advantage of the paradigm.

Therefore we need to carefully develop pedagogies/ didactics, tools and examples for learning and learning the concepts of OOT.

2 Workshop Organisation

This workshop was the sixth in a series of workshops on issues in object-oriented teaching and learning. Previous workshops were held at OOPSLA'97, ECOOP'98, OOPSLA'99, ECOOP'00 and OOPSLA'01 and focused on team project courses, effective classroom examples and metaphors, and tools and environments. 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. Participants were selected on the basis of position papers submitted in advance of the workshop. Furthermore, participants were asked to come up with controversial topics/ideas that could be discussed by e-mail in advance of the workshop.

The workshop was organised into two presentation sessions (all morning) and two working group sessions (afternoon). In a wrap-up session at the end of the workshop all working groups presented their results.

Follow-up workshops are planned for future OOPSLA and ECOOP conferences. We have also planned to develop a web page with links to all workshops in this series as well as related information. Information about the current workshop can be found at <http://prog.vub.ac.be/ecoop2002/ws03/>. An extended version of this report will be published in the ECOOP Workshop Reader.

3 Summary of Presentations

This section briefly summarises 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.

Glenn Blank (Lehigh University, USA) gave an overview over 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 ADTs (Abstract Data Types). 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 to objective learning and helps students design ADTs to solve a problem. Currently modules on Inheritance and ADTs are available. Materials for a CS0 course are under development. Glenn noted 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 concepts in upper secondary schools. They propose an apprentice-based learning approach that combines top-down and

bottom-up teaching approaches with active learning. The 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 absence 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). Tools are used to generate Java code monitoring the assertions. In the first course students are introduced to the basic concepts of imperative languages, plus objects and classes. The 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 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 quite early run 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 (University of Berne, Switzerland) reported on his teaching experiences with the Squeak environment. He especially focused on the importance of ‘naturalness’ of a language and environment. According to Stéphane objects are not ADTs, they ‘are’ encapsulation, late binding and an anthropomorphic view. Squeak’s concept of morphs enables programmers to easily interact with (programmed) objects directly. It is very easy to develop small custom designed examples for specific teaching/learning purposes. Since the environment is very rich, students can very early develop substantial applications.

Joseph Bergin (Pace University, USA) discussed the applicability of XP (eXtreme Programming) practices in introductory courses. His interests are 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 supported by the tool JUnit ([JUnit]), which is introduced at the beginning of the course. Other XP practices are more straightforward.

Gilles Ardourel (LIRMM, France) reported on a case study that evaluated access graphs, a language independent graphical notation for access control mechanisms. The results of the case study suggest that access graphs work better than textual documentation to explain access control mechanisms. In a test involving Java and C++ code students using access graphs performed better than those using textual documentations.

Jan Dockx (University of Leuven, Belgium) presented the new pedagogy for programming used at their university. Courses based on good software engineering practices as discussed in the contract paradigm by B. Meyer, extended with behavioural subtyping.

Noa Ragonis (Weizmann Institute of Science, Israel) presented results from a case study on the teaching of constructors using Java. The case study was targeted at 10th grade high-school students with no prior formal knowledge in computer science. Objects can be instantiated and initialised in many different styles. Some of these styles can lead to misconceptions that hamper further learning. An example is the case where the initialisation of the instance variables is done together with their declaration (quite common in Java). Students may believe that class and object are the same since we only have concrete ‘constants’ as values. The case study found that ‘professional’ constructors that initialize attributes from lists of parameters should be preferred, since other ‘simpler’ styles caused serious misconceptions.

Rosalía Peña (University of Alcalá, Spain) introduced a new pedagogical language called PIPOO. PIPOO is (more or less) an object-oriented extension of Pascal aimed at reducing paradigm shift problems cause by their procedural-first approach using Pascal. PIPOO keeps Pascal’s imperative kernel, adapts and/or removes constructs that are not compatible with the OO paradigm, and incorporates new ones required for the OO paradigm. The evolution from Pascal to PIPOO reinforces the students’ diachronic conception of the programming constructors, providing continuity. This enables students to concentrate on the new (OO) concepts. Once their mind is set on the OO world, it is easy to understand the peculiarities and characteristics of other OO languages.

Tracy Lewis (Virginia Tech, USA) discussed 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. An instrument will then be developed for gradually discussing design decisions using programming and design patterns, based on the level of design readiness. 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 at common knowledge and skills levels. Traditional classroom education can then be used for teaching advanced concepts, as well as for performing group work and practical exercises. 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.

4 Summary of Discussions

Discussions started already before the workshop. There were interesting arguments whether programming-in-the-small still is relevant. Workshop participants agreed that we need a (new) pedagogy that concentrates on basic concepts and design issues instead of language specific details. The focus should be shifted somewhat from programming to problem solving.

During the working group sessions discussions mainly focused on the following questions

- Can XP practices be taught in the first course?
- What does objects-first really mean?
- Should assertions be taught in the first course?

Five **XP practises** were discussed with respect to 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. There was a belief that by writing test themselves, the student learns a lot about the code that he/she is writing the test for. Writing tests makes students think about the(ir) code. Using Tools like JUnit or SUnit could be beneficial for students, although someone made the remark that JUnit tests are very hard to read.

The Planning Game. Students need to be taught how to plan their time for finalising a project. This is an important issue, since it fits well with the idea of small releases in companies.

Pair Programming. Pair programming can be a big help for many students, since students tend to learn a lot from their peers. It also helps in developing important social skills. However, many students are lacking the social skills to make pair programming work successfully.

Small releases goes together with the planning game. It is important as a skill, but also as a teaching strategy. There is a need to help students developing plans for their projects to enable small releases, depending on the level of maturity/experience of the student (first year or last year student for example).

Refactoring. XP always supports the saying ‘if it isn’t broken, don’t fix it’, but if you don’t fix it, code becomes dirty. Students however, 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?

To sum-up it was concluded that these XP practices can make a significant contribution in learning object-oriented concepts.

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. BlueJ may help avoid magic of static void main, etc., when starting, but it was felt that the BlueJ developers may need to provide more examples. Karel may also be good environment for starting—girls seem to enjoy 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 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!

5 List of Participants

Name	Affiliation	E-mail
Gilles Ardourel	LIRMM, France	ardourel@lirmm.edu
Joseph Bergin	Pace University, USA	berginf@pace.edu
Glenn Blank	Lehigh University, USA	Glenn.blank@lehigh.edu
Jürgen Börstler	Umeå University, Sweden	jubo@cs.umu.se
Kim Bruce	Williams College, USA	kim@cs.williams.edu
Martine Devos	Avaya Research, USA	mmdevos@avaya.com
Jan Doeckx	University of Leuven, Belgium	Jan.Doeckx@cs.kuleuven.ac.be
Stéphane Ducasse	University of Berne, Switzerland	ducasse@iam.unibe.ch
Andres Fortier	Universidad Nacional de La Plata, Argentina	andres@sol.info.unlp.edu.ar
Ines Grütznér	Fraunhofer IESE, Germany	gruetzne@iese.fhg.de
László Kozma	Eötvös Loránd University, Hungary	kozma@ludens.elte.hu
Tracy Lewis	Virginia Tech, USA	tracyl@vt.edu
Boris Mejias	Vrije Universiteit Brussel, Belgium	bmejias@vub.ac.be
Isabel Michiels	Vrije Universiteit Brussel, Belgium	Isabel.Michiels@vub.ac.be
Khalid Azim Mughal	University of Bergen, Norway	khalid@ii.uib.no
Kristen Nygaard	University of Oslo, Norway	Kristen@simula.no
Rosalía Peña	University of Alcalá, Spain	rpr@uah.es
Noa Ragonis	Weizmann Institute of Science, Israel	ntnoa@wis.weizmann.ac.il
Carsten Schulte	University of Paderborn, Germany	carsten@upb.de
Vasco Vasconcelos	University of Lisbon, Portugal	vv@di.fc.ul.pt

References

- [JUnit] JUnit home page, <http://www.junit.org>, accessed Jul 2002.
- [KNNZ 00] H.J. Köhler, U. Nickel, J. Niere, A. Zündorf: Integrating UML Diagrams for Production Control Systems, Proc. of the 22nd International Conference on Software Engineering (ICSE), Limerick, Irland, Jun 2000, 241-251.