A Model-First Approach to Teaching Introductory Object-Orientation

Jens Bennedsen
IT University West
Fuglesangs Allé 20
DK-8210 Aarhus V
Denmark
jbb@it-vest.dk

Michael E. Caspersen
Department of Computer Science
University of Aarhus
Aabogade 34, DK-8200 Aarhus N
Denmark
mec@daimi.au.dk

Over the years there have been ongoing discussions on the content of an introductory programming course. In order to define a common curriculum including an introductory course, ACM and IEEE established the Joint Task Force on Computing Curricula 2001. The charter was: “To review the Joint ACM and IEEE CS Computing Curricula 1991 and develop a revised and enhanced version for the year 2001 that will match the latest developments of computing technologies in the past decade and endure through the next decade”. In the final report [10], the role and place of programming in the curriculum is discussed. Is programming what needs to be taught first (what they call a programming-first approach) or are there other topics that need attention first? The conclusion is: “the programming-first model is likely to remain dominant for the foreseeable future”.

The report describes three implementations of a programming-first curriculum based on three programming paradigms: The imperative, the functional and the object-oriented paradigm. The object-oriented paradigm has gained much interest in the past decade resulting in many textbooks (e.g. [2, 3, 6, 8, 9]) and much interest among teachers on implementing the object-first strategy (e.g. [1, 4]).

In [7] three perspectives on the role of a programming language are described:

Instructing the computer: The programming language is seen as a high-level machine language. The focus is on aspects of program execution such as storage layout, control flow and persistence. In the following we also refer to this perspective as *coding*.

Managing the program description: The programming language is used for an overview and understanding of the entire program. The focus is on aspects such as visibility, encapsulation, modularity, separate compilation.

Conceptual modelling: The programming language is used for expressing concepts and structures. The focus is on constructs for describing concepts and phenomena.

These represent a widespread three-level perspective on object-oriented programming as represented by the three abstraction levels for the interpretation of UML class models [5]: conceptual level, specification level and code/implementation level.

When designing a programming course one decides how much time, effort and focus are given to each of the three perspectives. It is possible just to focus on the first, instructing the computer, and ignore the two others. This results in a course where the details of the programming language are in focus but where the students do not learn the underlying programming paradigm. If on the other hand one just focuses on conceptual modeling (using a case-tool to generate code), the result is a course where the students cannot produce code by themselves. We find it vital to balance the three views on the role of the programming language by including conceptual modeling. The primary advantages are

- a systematic approach to programming

- a deeper understanding of the programming process
- focus on general programming concepts instead of language constructs in a particular programming language.

Most of the descriptions and discussions of the object-first strategy tend to focus on *instructing the compute* and *managing the program description*. To our knowledge, no introductory programming textbook exists that include conceptual modeling, and we have only been able to find one article discussing the adoption of conceptual modeling in CS1 [1]. It is our experience from many years of teaching CS1 that including conceptual modeling perspective has a great impact on the student's skills and their understanding of the programming process. It is our firm conviction that the general omission of conceptual modeling is the major reason for the problems identified in [10, p. 23]: "*Introductory programming courses often oversimplify the programming process to make it accessible to beginning students, giving too little weight to design, analysis, and testing relative to the conceptually simpler process of coding. Thus, the superficial impression students take from their mastery of programming skills masks fundamental shortcomings that will limit their ability to adapt to different kinds of problems and problem-solving contexts in the future*."

[10] generally ignores conceptual modeling in the object-first recommendations for CS1. Aspects of conceptual modeling are only mentioned briefly and the recommended time to be used on the subject is four core hours!

[1] Alphonce, C., and Ventura, P.J.: "Object-Orientation in CS1-CS2 by Design", *Proceedings of Innovation and Technology in Computer Science Education*, Aarhus, Denmark (2002).

[2] Arnow, D., Dexter, S., and Weiss, G., *Introduction to Programming Using Java: An Object-Oriented Approach*, Addison-Wesley (2004).

[3] Barnes, D.J., and Kölling, M. *Objects First with Java – A Practical Introduction using BlueJ,* Pearson Education (2003).

[4] Cooper, M. et al.: "Teaching Objects-First in Introductory Computer Science", *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, Nevada, USA, (2003) 191–195.

[5] Fowler, M., *UML Distilled – A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley (2000).

[6] Horstmann, C.S., *Big Java*, John Wiley & Sons (2001).

[7] Knudsen, J.L., and Madsen, O.L., *Teaching Object-Oriented Programming is more than Teaching Object-Oriented Programming Languages*, DAIMI-PB 251, Department of Computer Science, University of Aarhus, Denmark (1990).

[8] Lewis, J., and Loftus, W., *Javav Software Solutions: Foundations of Program Design*, Addison-Weslsy (2003).

[9] Niño J., and Hosch, F.A., *An Introduction to Programming and Object-Oriented Design Using Java*, John Wiley & Sons (2001).

[10] The Joint Task Force on Computing Curricula (IEEE Computer Society and Association for Computing Machinery). Computing Curricula 2001 (final report), December 2001. Available on-line at http://www.computer.org/education/cc2001/final