AN INVERTED CURRICULUM FOR CS1

Michael E. Caspersen[1]

Abstract *- Most introductory programming courses and textbooks are structured according to the constructs of the adopted programming language and not on the basis of those language independent concepts, principles and techniques of programming that the students should master by the end of the course.*

We present and discuss the inverted curriculum for our introductory object-oriented programming course. and our experiences from teaching this course for four years. We identify four levels for the systematic construktion of programs, and the structure of our programming course is based on these four levels: the modeling level, the design level, the class level, and the algorithmic level.

Index Terms – Inverted curriculum, pedagogical patterns, object-oriented programming, systematic programming.

INTRODUCTION

It is our firm conviction that the primary aim for an introductory programming course is that students learn fundamental principles and general techniques for systematic construction of programs; consequently, we have designed an introductory programming course where we are focusing on such principles and techniques from the very first day and thereafter unfold these throughout the course as the students learn more and more of the adopted programming language (currently Java).

MEYER'S VISION

Our view is not a novel one as is evident from many papers from past SIGCSE conferences [1, 5, 6, 8, 9]. Bertrand Meyer [7] coined the term "the inverted curriculum" (or "consumer-to-producer-strategy") meaning that important topics and concepts should be covered first by using classes (solely through their abstract specifications), and only then the students shall learn about the internals of classes. A simplified variant of Meyer's vision is the objects-first approach which is prevailing in many new textbooks, but still many of these new books are structured on the basis of the constructs in the programming language and not on the basis of the concepts, principles and techniques that the students are supposed to master by the end of the course.

The fundamental concepts, principles and general techniques that are our primary concern are: objects, classes, state, control flow, parameterisation, informal design by contract (functional specifications, class invariants, loop invariants), conceptual and object-oriented modeling, design with interfaces, polymorphism, frameworks and algorithmic patterns (sweep, searching, merging, etc.).

THE EARLY BIRD APPROACH

Most of core concepts, principles and techniques are well-known, but rarely tought thoroughly in an introductory programming course; we do this, and we do it up front as early as possible. Our approach is an instance of the *early bird pedagogical pattern* by Bergin [2]. Our over all pedagogical model is a so-called spiral approach; we don't expect students to learn everything about a topic at first encounter, but we want to present and stress big ideas early.

In particular we focus on techniques for systematic programming. We identify and teach systematic techniques at four differeing levels: the modeling level (from problem description to UML model), the design level (from UML model to code structure), the class level (from interfaces and code structure to fully programmed classes using class invariants) and the algorithmic level (from functional specifications to implementations using algorithmic patterns and loop invariants).

We have given this course for 4-5 years, and our experience is very promissing [3, 4]. In a future paper we will expand on our approach and our experiences.

REFERENCES

[1] Astrachan, O. and Reed, D., "The Applied Apprenticeship Approach to CS1", *SIGCSE Bulletin*, 27, 1, 1995, pp. 1-5.

[2] Bergin, J., "Fourteen Pedagogical Patterns", www.csis.pace.edu/~bergin/PedPat1.3.html.

[3] Caspersen, M.E. and Christensen, H.B., "Here, There and Everywhere — On the Recurring use of Turtle Graphics in CS1", In *Proceedings of the Fourth Australasian Computing Education Conference, ACE 2000*, Melbourne, Australia, 2000, pp. 34-49.

[4] Caspersen, M.E., and Christensen, H.B., "Frameworks in CS1 – a Different Way of Introducing Event-Driven Programming", *SIGCSE Bulletin*, 34, 3, 2002, pp. 75-79.

[5] Decker, R. and Hirshfield, S., "Top-Down Teaching: Object-Oriented Programming in CS 1", *SIGCSE Bulletin*, 25, 1, 1993, pp. 270-273.

[6] Hilburn, T.B., "A Top-Down Approach to Teaching an Introductory Computer Science Course", *SIGCSE Bulletin*, 25, 1, 1993, pp. 58-62.

[7] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1997 (2nd edition).

[8] Pattis, R.E., "The 'Procedures Early' Approach in CS 1: A Heresy", *SIGCSE Bulletin*, 25, 1, 1993, pp. 122-126.

[9] Reek, M., "A Top-Down Approach to Teaching Programming", *SIGCSE Bulletin*, 27, 1, 1995, pp. 6-9.

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