Preface

This book contains a selection of the papers presented at the IFIP WG 3.2 Working Conference on Informatics Curricula, Teaching Methods and Best Practice – ICTEM 2002, plus three reports from the groups that worked during the conference. ICTEM2002 took place at the Jurerê Beach Village Hotel, in Florianópolis Brazil, from 10 to 12 July 2002.

The 13 papers selected to be included in this book were chosen from 24 papers presented at the conference. These papers report on several aspects of informatics curricula and teaching methods such as:

- Challenges in Defining an International curriculum
- The diversity in informatics curricula
- Computing programs for scientists and engineers
- Patterns of curriculum design
- Student interaction
- Teaching of programming
- Peer review in education

The working group meetings were organized in three parallel tracks. Working Group 1 discussed the "Directions and Challenges in Informatics Education". The focus of Working Group 2 was "Teaching Programming and Problem Solving". Working Group 3 discussed "Computing: the shape of an evolving discipline." Each WG worked actively and prepared a report with the results of the discussions; these reports are included as the second

part of this book. The success of the conference format and themes discussed encourage the participants to work on organizing new meetings to continue the work on informatics curricula.

We hereby would like to thank IFIP and more specifically IFIP WG 3.2, the Brazilian Computer Society and ACM for the support of this event. We also would like to thank the participants who contributed to the success of the conference, the reviewers who carefully selected and provided feedback for the papers and the conference organizers who have done a great job.

Lillian N. Cassel
Ricardo Reis
Co-editors

July 2002

Benchmark Standards for Computing in the UK

Andrew McGettrick
Department of Computer and Information Sciences; University of Strathclyde; Glasgow, Scotland
andrew@cs.strath.ac.uk

Abstract: In the UK the Government is concerned that standards should exist to ensure that all degrees awarded in institutions of higher education meet certain minimal criteria and therefore are of at least of a certain standard. To this end they have created a set of committees composed of subject experts whose task is to define the required standards for their discipline. The purpose of this paper is to outline the approach taken to address these benchmarking standards for Computing.

Key words: computing education, curriculum standards

1. BACKGROUND

Document [1] laid the foundations for the discussion and debate on benchmarking standards. This led to the formulation by the UK Quality Assurance Agency (QAA) of a requirement for experts to produce benchmarking standards for their discipline, i.e.

> *to produce broad statements which represent general expectations about standards for the award of honours degrees in a particular subject area. Benchmarking is not about listing specific knowledge, that is a matter for institutions in designing individual programmes. It*

> *is about the conceptual framework that gives a discipline its coherence and identity; about the intellectual capability and understanding that should be developed through a the study of that discipline to honours degree level; the techniques and skills which are associated with developing an understanding in that discipline; and the level of intellectual demand and challenge which is appropriate to honours degree study in that discipline.*

This report describes particular aspects of the benchmarking standards for the discipline of Computing. It was produced by a Committee selected jointly by the Conference of Professors and Heads of Computing (CPHC) and the British Computer Society (BCS) as being representative of a broad range of discipline expertise from within the UK. See [2].

2. THE TASK

Within the academic community a wide range of terms are used to describe degrees in the subject area. Computer science, computing science, computing, software engineering, software technology, information systems, artificial intelligence, computer systems engineering and information engineering are among the more common. Indeed the Committee had to provide benchmarking standards that would accommodate in excess of 2,400 different courses. The Committee took the view that the naming of degrees would be the responsibility of individual institutions and accordingly the standards should relate to the discipline and not just degrees with specific titles.

In producing the document, the Committee was conscious of the need to involve the academic community but also to take advice from the professional bodies (including the British Computer Society, the Institution of Electrical Engineers, the Software Engineering Association, the Academy of Information Systems and the AISB) and generally from industry and commerce as well as the public. Accordingly, a wide-ranging consultation process was used to confirm that the balance and the thrust of the document reflected agreed-upon views. Moreover, throughout the development of the standards it was deemed important to keep the academic community informed of developments as they unfolded. A web site was set up to inform interested parties.

3. AUDIENCE

The final Benchmarking Standards document had to meet the needs of four particular groups at least. These were the academic reviewers who would carry out reviews of departments, the general public who wish to be informed about the discipline, course developers, and finally external examiners. The manner in which the Committee set out to address these needs is given below.

3.1 Academic Review

Ultimately this process of academic review would involve an assessment of each Computing department in the UK; academic reviewers would have to make judgements about whether degree courses met the standards and had to be given guidance on how to address these benchmarking standards

3.2 The Public

To be accessible to a wide audience the standards had to be couched in language that was non-technical and non-threatening; yet, it was important to convey the sense of a new and exciting discipline that had the potential to open up a wide range of possibilities for study and future career opportunities

3.3 Course Developers

To stimulate the design and development of new and imaginative courses the Committee included a section on diversity of course provision; in addition, the standards were phrased in a manner intended to encourage novelty and not to constrain unduly

3.4 External Examiners

For this group (who as visitors to departments would have to agree to and preside over the awards of degrees) it was decided that guidance would be provided in terms of what should be sought, for example, in reviewing examination papers, in looking at final year projects, in guidelines that might apply for examination boards and so on. It was specifically not the intention that the benchmarking standards would be used when considering, for example, the award to each individual student

Student experiments in object-oriented modeling

Torsten Brinda
Department of Didactics of Informatics
University of Dortmund
44221 Dortmund Germany
e-mail: torsten.brinda@udo.edu

Abstract Exploration modules (EMs) and structures of knowledge as essential components of the author's concept "Didactic system for object-oriented modelling (OOM)" for improving the OOM education are introduced as new learning aids for student-centered learning. The didactic criteria "Basic concepts on different abstraction levels" and "Synchronization, transformation and evaluation of views" for the design of EMs are developed. A methodology for designing a "didactic map" of object-oriented basic concepts as a process oriented learning aid is described. The activities of learners when using EMs with the solution of complex problems, e.g. modelling a library system, are illustrated. The strategy for inclusion in the Informatics teacher education is connected with some words on the concepts' efficiency.

1. MOTIVATION

Good career prospects in the Informatics field led to an enormous increase in the number of Informatics study beginners at German universities from the middle of the nineties. According to the German Federal Office of Statistics, the total number of Informatics study beginners (first university semester) has risen steadily from 4611 in 1995/96 to 11496 in 1999/2000. The Informatics faculties can hardly cope with this crowd. For example, in the semester 2000/01 the University of Dortmund had to schedule lectures repeatedly per week due to the entry of about 1100 beginners. The load on students and lecturers was enormous. According to information from the German "Fakultätentag Informatik" (steering committee of all accepted German Informatics faculties) the ratio of graduates in relation to study beginners has fallen from 50% in 1999 to 45% in 2002. This reflects an

increase in the number of university dropouts and also in the study length of students. There is a need for a change in the study processes. In the year 2001, the German Ministry of Education and Research started to support about 100 university collaborative research projects in the field "New Media in Higher Education". About a fourth of these projects were Informatics projects. Multimedia e-learning materials were developed to support student-centered learning and to relieve overburdened Informatics faculties. The work of consortia like the "European Consortium of Innovative Universities" [4] extends this approach to an international level.

Besides the development and distribution of e-learning materials, the Informatics study must be developed further by including new learning forms and learning aids so that student-centered learning, self- paced learning, and the preparation for lifelong learning become essential structuring elements of the learning process. Learning concepts, which should be taken into account in this context more strongly, include active and explorative learning [5, p. 150]. Traditional learning scenarios in higher Informatics education do not include enough such natural learning forms. An auditorium is a difficult environment for learning by discovery. Within the approach of "discovery learning" the teacher should design the learning process as a sequence of problem situations, each involving a learning task, which stimulate the learners' research interest. Suitable result-oriented learning aids are necessary to support learners in their explorative learning process.

Brinda and Schubert developed a concept called "Didactic system for object-oriented modelling" as a combination of traditional and new studying concepts [2, 3, 7]. The concept addresses beginners in object-oriented modeling (OOM), selected for the study because of its relevance in a variety of Informatics areas such as software engineering and object-oriented databases. The main goal of the didactic system is to bring a new quality of learning to the OOM field. The didactic system makes it possible for learners to navigate in structures of knowledge, to construct solutions for exercises from exercise classes, and to learn by discovery with exploration modules (EMs). In the context of this paper, EMs can be thought of as software modules (small applets, applications and animations). In a wider sense, learning texts and other media are included to form more complex EMs. Here the main emphasis lies on structures of knowledge and the embedding of EMs in the social process of Informatics study. Exercise classes are discussed in [2].

2. EXPLORATION MODULES AS NEW LEARNING AIDS

Discovery learning strategies are often applied by Informatics students as a work reduction strategy. They discover and then reuse standard algorithms from textbooks. The same strategy is used with software libraries. Available solution parts are adapted and integrated in the solution of new problems to avoid repeated development. This intrinsic motivation, which lies within the subject, led to the concept of the EMs. Learners explore EMs in suitable learning scenarios and in that way learn about basic object-oriented concepts and object-oriented models.

Within the design of EMs, manipulative and perceptive ways of exploration have to be considered. Manipulative exploration allows the exploring person to change something and get immediate feedback. The EM has to provide specific manipulation and observation components, combined with check mechanisms. Perceptive exploration requires structures that can be discovered. Therefore, the EM has to offer multiple cognitive approaches, which show the explored object in different ways and combine or even synchronize different views to visualize complex structures. By didactic analysis of the OOM field indicates that beginners should get to know and design structures (static basic concepts) and understand and control processes (dynamic basic concepts). Starting from an application they should analyze, modify, construct, and assess object-oriented models in the stages of structure element, structure, and model. This results in the exploration promoting features of EMs shown in table 1.

Table 1. Exploration promoting features of EMs

Feature	Description
Basic concepts on different abstraction levels	Structures, their elements and models must be explorable on different abstraction levels to give learners with various pre-knowledge a cognitively demanding entry to object-orientation, i.e. the complexity of views ought to be adjustable. It has to be possible to switch on or off selected views. For beginners as a minimum of one static view (class diagram) and one dynamic view (interaction diagram) are necessary to describe the structure and the time change of an object-oriented model.
Synchronization, transformation and evaluation of views	Synchronization represents a "didactic bridge" between different diagrams. This makes it possible to bring the single views together to a picture of the complete system and to overcome a known cognitive barrier. For some educational purposes, it should be possible to synchronize different views automatically. If learners do this manually, model check functions must be provided to prove the model's consistency.

Input/Output for a CS1 Course in Java
Some Considerations

Elliot B. Koffman
Computer and Information Science Department, Temple University, Philadelphia, PA, USA
koffman@temple.edu

Abstract: This paper discusses considerations for input/output for a first course in Java. It includes the opinions of several computer science educators regarding the pros and cons of using a non-standard package for input/output and the requirements of such a package. It also describes the evolution of a simple package for input that it is easy to implement using the Swing class and gives guidelines for using standard Java for simple input/output.

Key words: Computer Science 1 (CS1), Novice programming in Java

1. INTRODUCTION

1.1 Non-standard packages for input/output

Many Java textbooks for CS1 use simple packages for Java input/output. Some classes that were developed to simplify console input/output are: ConsoleReader [3], Text [2], and SimpleInput [1]. These classes were written because the textbook authors and teachers of CS1 recognized that console-based input in Java is difficult for novices to use. At the same time, a few packages were also written that permitted GUI-like interactivity. Examples of these packages and classes are: simpleIO [7] and Java Power Tools [8]. These packages were developed because their implementers recognized that console-based input/output was tedious and uninteresting to students who were accustomed to applications with Graphical User Interfaces (GUIs). The package developers also felt that novice students should be able to write programs with GUI-like interactivity before

mastering the Java APIs that would enable them to build their own GUIs: AWT and Swing.

1.1.1 Educator comments on using non-standard packages

In early April, 1999 there was a thread on the SIGCSE listserve that discussed the use of non-standard packages for teaching CS1 in Java. The messages seemed to be split fairly evenly between those who advocated their use and those who were against their use. The following comment by H. Conrad Cunningham (University of Mississippi) succinctly summarizes the two points of view:

"On one hand, use of simplified I/O or GUI packages can make Java easier to teach and use in CS1/CS2. So the various packages that are available with various textbooks can be helpful. And we want to convey to our students that it is easy to add on significant capabilities to Java and other such languages.

On the other hand, there is concern that use of a non-standard, add-on package might be confusing to first year students. Students tend to take what they learn as being part of Java. When they move to another environment, they are somewhat lost and confused. They have the overhead of relearning an I/O package or installing the one they are accustomed to. There is thus some sentiment to only teaching the "standard" (whatever that means with Java)."

A. Joseph Turner (Clemson University), makes the following point in favor of using packages:

"I haven't tried teaching CS1 or CS2 using only the classes in the standard Java library, but unless you want to limit yourself to reading only a character at a time or a line at a time (as a string), you don't want to use only the I/O facilities in standard Java. (There are some ways to do a bit better than reading only characters or lines, but the overhead and complexity of doing so are out of the question for beginning students as far as I am concerned.)"

David Arnow (Brooklyn College of City University of New York) makes the following argument against the use of packages:

"There is another argument in favor of NOT providing a special I/O package for student use in CS1/Java. While constructions such as

```
BufferedReader br = new BufferedReader(
                        new InputStreamReader(
                            new FileInputStream(
                                new File("myinput"))));
```

may be introduced as "boilerplate", they also provide an opportunity to:

- emphasize the idea of classes as models, as repositories of particular sets of behavior.
- illustrate the use of composition
- foster the notion and UTILITY of abstraction

In connection with the latter, students, even CS1 students, can appreciate the fact (and the accompanying discussion) that the last two lines of the above code can be replaced with

System.in

or

new URL("http://www.acm.org").openStream()));

or

all sorts of other things eventually

because each of these things play the "role" of an InputStream. In other words, if one's goal is to take an object-oriented approach in CS1 Java's I/O classes can be viewed as an asset, not a liability. (As a side note, this i/o stuff usually turns out to be the LEAST of the students' problems in CS1.)"

Michael G. Branton (Stetson University) advocates teaching basics of the AWT first instead of using packages:

"You can also do it with Java without that much fuss. I've taught the course 3 times now using AWT. you don't need very much of it to put a text field or 2 and a button on the screen, folks. There seems to me a lot of hand-wringing over this, but I've found that my students don't find this difficult to deal with at all. It's the algorithm development that's still the "hard part." My experience is that teaching them enough of the GUI for CS1 doesn't take much time at all. I still have plenty of room for the topics I consider important, and text fields and buttons give them something very concrete (in a virtual sort of way :-)) to think of as objects right off the bat."

Finally, Judy Bishop (University of Pretoria) makes the following points in favour of non-standard packages:

- "Java is an extensible language, with extensions coming via packages (APIs). In today's world, the ability to harness this power is more integral and more important for fresh minds to understand in *week one*, than a long-winded object instantiation, four brackets deep.
- An I/O class is an excellent case study in its own right as it illustrates exception handling, as well as tokenizers and string functions. It also does not have to be long: our Text class is 2.5 book pages of code, providing 10 methods for input, output and file opening.
- If a lecturer feels very strongly about teaching Java at the rock face, and is not convinced by point 1, then I would still suggest that when writing your own package! comes along in the course (which it ought to at some stage), then an I/O or Graph package is an ideal candidate."

What have we learned from this thread?