Teaching Programming and Problem Solving: The Current State

Elliot Koffmann (US, chair), Torsten Brinda (GER, Rapporteur)

Participants

Juan Alvarez (CL), Amruth Kumar (US), Maria Lucia Blanck Lisboa (BR), Juris Reinfelds (US), Peter Van Roy (BE), Raul Sidney Wazlawick (BR).

1. PARADIGMS AND LANGUAGES

Several approaches are currently used for introductory courses in Informatics. For example the ACM 2001 Curriculum report presents a choice of paradigms for the introductory programming courses: imperative first, objects early, objects first, functional first. These courses are taught in a variety of programming languages, most notably Java, C, C++ and in the recent past Pascal. Normally one language is used in the programming courses. The Advanced Placement Examination in the US and Informatics departments in many countries are transitioning to Java due to its practical relevance and support of object-oriented programming. In some institutions, the introductory programming sequence is expanding from a two-course to a three-course sequence, e.g. programming basics, object-oriented programming (OOP), data structures and algorithms (DS+AL) or OOP1, OOP2, DS+AL.

2. PROBLEMS

Diversity of student background

In the introductory courses, the teachers have to cope with very different pre-knowledge of their students. Some students start with an excellent secondary education in Informatics, others have acquired a lot of knowledge

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35619-8_15

L. Cassel et al. (eds.), *Informatics Curricula and Teaching Methods*

© IFIP International Federation for Information Processing 2003

in their spare time or in jobs on the side, others come with hardly any pre-knowledge. Some students who have experience will need to unlearn bad habits. Moreover, many students start their Informatics studies with insufficient mathematical and logical thinking capabilities and they have difficulty with the abstractions needed for success in Informatics. A lot of them also have incorrect expectations about Informatics study.

Students without prior experience

Some students without any prior computing experience will outperform those who have prior experience. We need to make sure that students without experience are not discouraged from taking introductory programming courses. We also need to keep in mind that learning programming in college is a new experience for these students. For example, they have all taken History and Mathematics courses, but many of them have no experience in learning how to solve problems or in algorithmic thinking.

Faculty resource and training

The dropout rate in the introductory courses is high. However, this is not just because of problems on the side of the students. Faculty shortages are a big problem that often leads to non Computer-Science-trained faculty, or faculty without an object-oriented background, teaching introductory courses. It is more harmful to use faculty with inadequate training to teach an object-oriented programming course than an imperative one. We need to make sure that experienced faculty are used in these foundation courses. Rather than accommodate all student demands for programming courses, we should reduce section offerings if qualified faculty are not available.

Under emphasis on modelling and theory

In the courses there is often a focus on syntax details and getting programs to run, not on problem solving, information modelling, understanding and reasoning about programs. So, programming and problem solving are still taught as an ad-hoc art without any underlying theory. Teaching focuses on individual languages and tools rather than giving the student a unified view of programming as a discipline. Instructors in later courses complain that their students cannot program.

Laboratories

Despite recommendations by ACM and ITiCSE working groups on laboratories for closed (i.e. supervised) laboratories, many schools use open laboratories. In many schools, students are still working without direct supervision. This is because of a lack of resources (instructors, computers, and space). In many cases, this leads to plagiarism, which is widespread in many countries.

Lack of women in Informatics courses

The percentage of women students in Informatics remains small. To some extent, this is because women tend to be intimidated by their male

peers who spend more time on the computer playing games and surfing the Web. This leads women to form the incorrect belief that they will be at a disadvantage in a programming course, so they avoid taking a course for majors and enroll in a "terminal course" instead.

3. TRENDS

3.1 Paradigm shift

A paradigm shift from imperative to object-oriented problem solving is under way in many countries. Currently there are a number of approaches of how best to change from imperative to object-oriented thinking. There is also a trend away from text-based (e.g. "Hello World") programs to more student-oriented ones, because text-based examples do not motivate students of the multimedia generation. Problem-based learning approaches are becoming more popular in an increasing number of universities. Interdisciplinary projects (less lecture-based, more problem-based) and teamwork with students in other disciplines are also becoming more prevalent. Teamwork and collaborative learning in general are increasingly encouraged at an early stage.

3.2 Use of software development tools

Software engineering techniques (e.g. graphical modelling languages) for "programming in the large" are becoming more popular in introductory programming courses. There is an effort towards more rigorous software development and this includes teaching students how to prepare more and better documentation. A number of commercial and free software development tools are also in use to make students familiar with professional environments at a very early stage in their education. Tools, which generate code from user-specified object properties and class relationships, support this process, because students easily can see the advantage of the models they constructed beforehand.

On the other hand, the suitability of such professional environments for the learning process is being discussed. The most well known point of criticism is that their user interfaces with the huge variety of offered functions are optimized for efficient and fault-free software development, and not for learning software engineering. Moreover, these interfaces often guide users to avoid certain kinds of errors so they cannot learn from these

errors. Thus, special educational versions with reduced functionality are used in some cases.

3.3 Visualization, animation and exploration

There is an increasing use of environments for the visualization of basic modelling and programming techniques, algorithm animation and program visualization. Online tutors and tools for the exploration of concepts are used for learning programming. An increasing number of these tools is available for free on the World Wide Web. We need to have mechanisms in place to verify the quality of free software tools. We must keep in mind that the initial price of software is a small part of the total cost (often free software requires maintenance and support).

3.4 Patterns and generic programming

To learn reuse, good design, and the experience of experts, learners are introduced to patterns and generic programming techniques (e.g. templates) in solution design and implementation.

4. CHALLENGES

Because of the rapid pace of change in Informatics, we need to find ways to keep educators up to date. Universities and Informatics Societies, industry and governments should devote resources to install mechanisms whose purpose it is to keep the faculties up to date. An approach used in Germany to enhance the theoretical foundation of Informatics education was to form departments for "Didactics of Informatics".

In teaching programming and problem solving, we face the following challenges:

- Teaching and learning programming is quite a difficult task. How can we teach programming without overwhelming the students with too many concepts at the same time? What concepts should be taught at each level (e.g. inheritance in the first or second course)? What does the programmer of the future have to know and what not?
- Faculty need to become more familiar with research that has been done to determine attributes for success in programming. We need to find ways to identify students who have these attributes and to encourage them to take programming courses. We need to be flexible in our teaching methods and teach students in ways that help them succeed.

- What will be the granularity level? How do we prepare our students to deal with the different granularity levels of programming?
- What is the best way to shift from the current programming paradigm to the next one? Although Java appears to be the primary teaching language today, that will likely change in the future.
- How do we accommodate diverse populations of students? How do we keep attrition rates low and maintain quality in our courses? How do we increase the number of women who take and complete the introductory programming courses?
- Modelling and programming languages and their libraries are becoming increasingly complex. How do we find a suitable subset for teaching purposes and how can we build up educational software libraries?
- How can we effectively use educational software libraries? Can metaphors and mini worlds be used effectively to teach introductory programming and if so, how?
- How do we make better use of visualization and exploration tools in teaching programming and problem solving? How do we develop such tools?
- Students should be able to see the commonalities between different paradigms and use concepts, where they are needed. Should we, and if so how should we, give students a broader view of programming, in which different paradigms are seen as different facets of a uniform framework?
- Although concurrency is increasingly present, there is no perceivable trend to introduce concurrent programming in first courses. Instead, the concurrency of programs is hidden from the user in various ways (e.g. event handling in Java programs). Should concurrency be included in the introductory sequence and if so, how?
- Multi-language programming is widely prevalent in industry today. How do we prepare our students to move easily towards multi-language programming?
- Because programming is more taught as a craft than as a science we should find ways to encourage a theoretical foundation for programming which is useable by practicing programmers and not only by mathematicians.
- In many institutions, the first programming sequence seems to be driven by short-term industrial demands (for example, what language does local industry need at the moment), rather than what may be best for the students in the long term. We need to find ways to find the right balance between pedagogical and industrial needs, so that we can do what is best for our students. We also need to be vigilant that the latest developments in the IT industry do not overly influence the direction of our courses.

- Currently the emphasis is on writing programs. We should teach students to write readable programs and to become more proficient in program reading.
- Research shows that active learning is more effective than passive learning. How do we incorporate active learning into introductory programming courses? One approach would be to implement problem-based learning as is currently being done in various engineering disciplines.
- Closed introductory programming laboratories require additional space. Institutions need to reallocate space from areas that have a surplus to Informatics.

5. FUTURE DIRECTIONS

We observe the following changes, which Informatics education has to address:

- Informatics is becoming more and more a mandatory subject in secondary education. An important question there is what aspects of Informatics should be common knowledge for all citizens of our society. The answer to this question will have a significant effect on how computing is taught in higher education.
- Perhaps because of the continued growth of Informatics education and subsequent pressures on students, plagiarism is a growing problem that we have to solve.
- Programming is undergoing fundamental changes. It is becoming more net-centric and multimedia-centered. These changes are driven by the increasing use of computing in all areas of society, not just text-based information processing. Traditional text-based programming education is not sufficient to address these changes.
- There is increasing demand from other disciplines to incorporate aspects of Informatics into their own curricula. We cite bio Informatics and computational linguistics as particular examples, but the demand comes from nearly all disciplines. Informatics is continuing to move away from its roots in scientific computation to infiltrate the whole range of disciplines.