Experiment around a training engine

Anne Brygoo, Totou Durand, Pascale Manoury, Christian Queinnec and Michèle Soria
Université Pierre et Marie Curie (Paris 6), UFR d'informatique, France

Abstract: We describe a teaching experiment where an introductory course to computer science is accompanied by use of a computerised training engine. This whole engine relies on the existence of an interpreter of the taught programming language that allows us to offer quizzes as well as exercises with some automatic marking facility. Students may then perform their homework with immediate feedback, without being connected to the internet. However, students' answers are eventually gathered in a central data base where they may be analysed, thus providing the means for 'personal coaching'.

Key words: software to improve the learning process, distributed learning system, improving learning environments

1. INTRODUCTION

The University 'Pierre et Marie Curie' (Paris 6) gave us in 1998 the responsibility for running the first introductory course in computer science (CS) in a cursus named MIAS (mathematics and informatics applied to sciences), a two-year cursus where young students, 18 to 20 years old, study general mathematics, physics, mechanics and computer science before choosing to graduate in only one of these sciences. The CS cursus is made up of three other mandatory courses accompanied by an optional CS project. There are roughly 800 students in the first year, 600 in the second year; 450 get their final degree: 250 choose to pursue mathematical studies, while 160 others choose CS.

These figures show that most of our audience were not bound to become computer scientists, so we decided to introduce the students to the main

D. Passey et al. (eds.), *TelE-Learning*

concept of CS: the 'evaluation process', that is, how a computer turns (executes) a text (a program) into some result.

Of course, we also decided that our teaching should attract students to CS. We therefore favoured a conceptual approach mixed with numerous programming activities. These ideas are not new and are rooted in the well-known SICP book (Structure and Interpretation of Computer Programs, Abelson and Sussman, 1985) used and taught for a long time at MIT.

The teaching is 12 weeks long and every week is made up of one course (1 hour 15 minutes) and one lab session (1 hour 30 minutes) associated with a pre-lab session (1 hour 30 minutes). Thirty students form a group monitored by one teacher. Every group follows a computerless pre-lab session. The lab session is performed in specialised classrooms with 15 computers: each computer is operated by two students together.

Last year, we volunteered to prepare an experiment where 50 students would be taught differently. This experiment is described in the rest of this short paper (for the complete report on this experiment see Brygoo et al., 2002). The main lines of our teaching are detailed in Section 2 (as well as giving details of the SPAD experiment and how it differs from the regular course). Section 3 presents the software architecture of the associated computerised environment. The results of the experiment appear in Section 4, followed by some conclusions and future perspectives.

2. CHOICES, OBJECTIVES AND EXPERIMENT

The goal of the course is to present the 'evaluation process', that is, the general principles that allow a computer to interpret a text as a program whose value should be mechanically obtainable. We chose to use a subset of the Scheme programming language (Kelsey, Clinger and Rees, 1998).

Our course is divided into three seasons. The first season (six weeks long) is devoted to recursion on numbers and lists; the second season (four weeks long) presents trees and grammars and, of course, recursion on trees and the concept of 'abstraction barrier'. The third and final season (two-weeks long) presents the evaluation process as an interpreter for our Scheme subset written in our Scheme subset. The third season does not introduce any new concept. It only gathers many functions (most of which were studied in lab sessions) for a single goal: the evaluation of a small but powerful programming language.

Finally, the most difficult aspect of CS at that level is to master abstraction, that is, to differentiate between syntax and semantics, knowledge and information, aspect and meaning.

2.1 CD-ROM

We created a CD-ROM to support our course. This CD-ROM is targeted for use with the Windows and Linux Operating Systems. Its aim is to provide every student with a means to practice the course at home, that is, reading, programming and thinking. Moreover the CD-ROM software provides immediate feed-back wherever possible — without any internet connection.

The CD-ROM contains the software required to program in Scheme as well as the material for the course, including numerous documents related to the Scheme programming language, pragmatics and community. This extra material is provided since a programming language is not reduced to syntax and semantics but also include pragmatics, folklore, programming guides or tricks, etc.

The CD-ROM favours connection-less self-training, that is, besides a traditional course (in HTML and PDF form) it offers quizzes as well as exercises that do not require an internet connection. More than 85% of our MIAS students have a computer available at home but less than 45% have an internet connection. Despite this increasing level of wealth, (and conversely to educational platforms vendors), we strongly believe that, in the next ten years, most students in the world will still not be constantly connected from home to the servers of their university. Therefore we favoured an architecture where students solve quizzes or exercises, submit their answers and have immediate feedback telling them how good that solution is: the feedback is computed locally and does not require an active connection.

The current version of the CD-ROM contains nearly 400 questions in quizzes and 245 questions within 58 exercises. This gives great latitude to students (and teachers) to choose which exercise to practice (or study).

2.2 The SPAD experiment

The university decided, a year ago, to experiment with distance learning at the MIAS level. In September 2001, a group of nearly 50 students began to be taught in a new way in mathematics and CS for one semester. First, the students only needed to spend three days (instead of five) at the university. Second, every student received a CD-ROM containing the computerised teaching material. We decided, in CS, to organise the students' week on a new basis: we compressed the course and merged the pre-lab and lab session into a single weekly 'pedagogical rendez-vous' (2 hours). We also reduced the size of a group to 15 students so every student might practice alone on a computer.

However, since we wanted to follow the progress of our SPAD students, the CD-ROM software locally stores all their answers and transfers them, later on, to the servers of the university when an internet connection is made. Most often, with a few days lag, we get a precise view of our groups enabling us to advise our students on a personal basis via e-mail or a shared forum.

Within the constraints, we organise the week as follows:

- Tuesday was the day of the pedagogical rendez-vous. Weekly assignments (course, quizzes and exercises) were prescribed.
- The course should be read and first-level quizzes completed for Friday.
- Questions about the course should be posted (on a forum) by Sunday.
- Exercises should be completed by Monday.

The organisation of the week rules the content of the pedagogical rendez-vous. We answer student's questions, we present briefly the most delicate points of the written course and, finally, we supervise the students practising quizzes or exercises as in a normal lab session.

3. TRAINING ENGINE

The CD-ROM contains a PDF version of the written course so it may be searched or printed, as well as an HTML version chopped into pages centred on a single topic. This course is intended to be the main document containing all sort of links to quizzes, exercises or other pages with extra information.

Since the course heavily uses a programming language, we also provide a programming environment for that language: we chose DrScheme by the PLT team from Rice University. DrScheme is a useful environment with a lot of well thought-out pedagogical features. It runs on a variety of Macintosh, Windows and Linux systems and is easily installed on all sorts of computers.

Quizzes and exercises are written in Scheme, they are installed as an additional package to DrScheme where they run as separate threads. As we regularly improve and extend this package, students get used to updating their configuration (with a simple click).

3.1 Quizzes

Teachers' quizzes help students to understand the course before getting involved in exercises. Quizzes adhere to the structure of the course and provide questions on every topic of the course. Quizzes are ranked from easy to difficult and from optional to mandatory. Students are aware of this

ranking. Quizzes are not marked; answers are just checked for correctness wherever possible.

After reading a topic of the written course, the student is offered some quizzes as simple links. Technically, clicking on such a link directs the browser towards a web-server embedded within DrScheme. This web-server loads the required quiz (a Scheme file) and starts evaluating it. This program (the quiz) is made of a succession of standardised questions at various abilities for the student (Queinnec, 2000). We distinguish three levels of quizzes:

- Simple applications of the course.
- Questions on the course itself.
- 'Meta-questions' that replace knowledge from the whole course.

The standardisation of questions makes it easier for students to recognise the type of question they have to solve. It also makes it easier for teachers to write quizzes since only the varying parts are to be specified. The HTML decoration is therefore totally unrelated to the scientific content.

To sum up, most of the quizzes allow students to program short items in Scheme, without the complete DrScheme programming environment, with the sole power of a browser.

3.2 Exercises

An extra menu item within the DrScheme programming environment allows students to choose an exercise. Exercises are made up of a series of questions. A question asks for the definition and the test of one (or more) Scheme function(s).

First, the student writes the required function, followed by some tests. He may then hit the 'Check' button to get some feedback for his work. As developed below, the feedback consists of a mark associated with some comments justifying this mark. The marking process takes into account many syntactical or semantical aspects of the program into account but is not intended to replace the teacher. The mark is an indicator that tells the student how correct the program is. Additionally, if the mark is above a given threshold, an (HTML) solution is displayed.

3.3 Traces

For all quizzes and exercises, solutions and their evaluations (most often a number) are time-stamped and stored (more or less immediately) in a data base of a central server of the university. We developed, for our own usage, some SQL web-based forms making inquiries of the data base to display the state of any particular student with respect to the quizzes or exercises of any

given week, thus allowing us to offer some comments (by e-mail) on his answers: this is what we call 'personal coaching'. We also question the data base to display the state of a whole group with respect to the quizzes or exercises of any given week, to allow us to write a page entitled 'Weekly advice' or post in the forum any comment upon a popular mistake or habit.

These forms also allow some statistical analyses to determine which questions (quiz or exercise) have a high failure rate because either the question is poorly worded or not feasible at that place in the course.

4. RESULTS AND PERSPECTIVES

A first and surprising result is that the 50 students in the SPAD group are rather representative of the whole group of regular students: we observed students that were always absent, students loosely interested in CS, students that produced regularly poor results, students with a huge background in windows-based software but unable to master recursion, as well as students without any former programming experience but with good results.

The continuous use of a computer for that course showed unanticipated effects. SPAD students used the computer to read the course, followed the links towards quizzes, performed quizzes on screen, and switched between the browser and the programming environment. Moreover, they were alone on a computer during a lab session, and therefore, they were much more at ease with computers than the rest of the MIAS students: the computer became a helper device rather than an opponent to be tamed. SPAD students used the computer as a specialised co-worker.

We generated some statistics to compare the 750 regular students with the 50 SPAD students. The results showed no significant difference between both populations except that extremely good students seemed less rare. But we also report a refined perception of the experiment, based upon the final questionnaire that was completed by all students, and several discussions with the SPAD students.

SPAD students did enjoy the experience: the main points that were brought out were:

- they feel free to organise their work, but very much appreciate the weekly prescriptions they are given (see Section 1.2).
- although they meet their teachers only once a week (and would rather have two rendez-vous), they feel closely connected with them via e-mail and the daily maintained forum.
- they appreciate learning programming by practising (even for those who had no experience with a computer, and first had to struggle a lot).

5. RELATED WORK

We share with ELM-ART, an intelligent tutoring system on the WWW to teach Lisp (Brusilovsky, Schwarz and Weber, 1996; Weber and Specht, 1997), a number of goals and means. We teach a similar language (Scheme is an heir of Lisp) and our web-server (for quiz) is written in Scheme (whereas ELM-ART uses CL-HTTP written in COMMON LISP).

There are many differences though. Our system uses primarily the programming environment DrScheme. This allows students to write Scheme programs with great comfort, but this is not the case with any web-based system we know. It also allows for more interesting exercises where we provide some libraries to be assessed by the students. Students have access to all the debugging means provided by the programming environment to perform their assignment.

The way we mark exercises by comparison to the teachers' solution is very easy to put into practice. This solves one of the major difficulties highlighted in many works (Joey, Chan and Luck, 2000) which is to write these marking functions. The work is reduced to writing at least one solution, then to deciding to which (possibly dynamically-generated) set of inputs, the solution and the student's answer should be compared. Given that we use Scheme and run our tests on the values themselves (i.e. their representation in memory) instead of their printed representation, we are free from the burden of specifying any precise IO format: this is quite similar to the Boss2 solution (Joey, Chan and Luck, 2000) that uses Java interfaces to hide implementation details.

6. CONCLUSIONS

As is the case for nearly all teaching experiments, we feel that the experiment is a success both for students who liked the freedom they gained with this organisation and for teachers who experienced a new means of teaching.

The material we developed for the SPAD experiment is not only useful for SPAD students, it brings the opportunity for all our MIAS students (and possibly other French-speaking students around the world) to practise at home, with feedback, the quizzes and exercises supporting our course.

Moreover, we gathered a number of pedagogical resources offering new teaching possibilities: computerised homework with automatic submission (via the trace system), computerised examination during lab sessions, computerised revision (with quizzes) during free-hours. We are eager to explore these new fields with our colleagues.

Eventually, we collected a number of traces that we plan to explore further. Many elaborated studies may take place, for instance computing the difference between two consecutive answers to the same question. This will allow us to understand better the learning process in order to improve our teaching.

The web site and the CD-ROM we developed for this course are freely accessible (but mostly in French) at http://www.infop6.jussieu.fr/deug/2001/mias/mias-a/deugspad, and http://www.infop6.jussieu.fr/cederoms/VideoScm2001/.

REFERENCES

Abelson, H. and Sussman, G.J. with Sussman, J. (1985) Structure and Interpretation of Computer Programs. Cambridge, Mass.: MIT Press

Brusilovsky, P., Schwarz, E.W. and Weber, G. (1996) ELM-ART: An intelligent tutoring system on world wide web. In C. Frasson, G. Gauthier and A. Lesgold (eds.) Intelligent Tutoring Systems (ITS'96), Vol. 1086 of Lecture Notes in Computer Science, 261–269. Berlin: Springer-Verlag

Brygoo, A., Durand., T., Manoury, P., Queinnec, C. and Soria, M. (2002) Experiment around a training engine (Complete version). Available at: http://www.spi.lip6.fr/~queinnec/Papers/ifip2002.ps.gz

Joy, M.S., Chan, P.-S. and Luck, M. (2000) Networked submission and assessment. In Proceedings of the 1st Annual Conference of the LTSN Centre for Information and Computer Sciences, LTSN-ICS.

Kelsey, R., Clinger, W. and Rees, J. (eds.) (1998) Revised K report on the algorithmic language Scheme. Higher-Order and Symbolic Computation, 11, (1), 7–105

Queinnec, C. (2000) The influence of browsers on evaluators or, continuations to program web servers. In ICFP '2000 – International Conference on Functional Programming, Montreal, Canada.

Weber, G. and Specht, M. (1997) User modeling and adaptive navigation support in www-based tutoring systems. In A. Jameson, C. Paris and C. Tasso (eds.) Proceedings of the Sixth International Conference on User Modelling (UM'97), Cagliari, Italy.

BIOGRAPHIES

The authors teach in the computer studies department of University Pierre et Marie Curie. Their combined teaching experience roughly amounts to a century.