TEACHING PROGRAMMING WITH A COMPETITIVE ATTITUDE TO FOSTER GROUP SPIRIT

Pedro Guerreiro
Universidade do Algarve, Gambelas, 8005-139 Faro, Portugal

Katerina Georgouli
Department of Informatics, Technological Educational Institute, Agiou Spiridonos, Egaleo 12210, Greece

Keywords: Blended-learning, competition-based learning, web-enhancement, automatic judging systems.

Abstract: Socialization is an important aspect of university life. We believe that if students feel that they fit in the group, their commitment will be higher and their results will be better. In introductory programming courses, most tasks are elementary and are usually performed on an individual basis. If we manage to give greater visibility to those lonely activities, students will find out that the difficulties they face are shared by many, and realize that they are not alone. We do that by adding a competitive flavour to the tasks in the course. For example, programming assignments are modelled after programming competitions; quizzes are given after each lecture, students get points for it, and a ranking is kept, much like those in some sports; we organize tournaments, where students' programs play against one another in a computer game. This provides a supplement of excitement to tasks that otherwise might be uninteresting to newcomers, and fosters group spirit. As a consequence, student participation is higher and results were better than before.

1 INTRODUCTION

It is well known among teachers that a very significant factor for students' success or failure on a particular course is the degree to which students get involved in course activities (Felder, 2004; Wang, 2001). Another factor is the group feeling students experience when they have to be involved in common tasks. This is true of traditional classroom courses, and more so in blended learning courses, where interaction between learners might be enhanced by well designed communication tools strengthening their feeling belonging to a community.

There are four components of classroom community, outlined by Rovai (2001): spirit, trust, interaction and learning. Firstly, 'spirit' is the feeling of belonging to a group. Secondly, 'trust' is simply the feeling that the group can be trusted and the group members will give feedback to each other. Thirdly, 'interaction' is the feeling that community members have that they may benefit by interacting with other members of the community. Finally, 'learning' is the sense that community members have that learning can come about due to the community discussing information, that is, the community can construct knowledge.

In a blended learning environment group members may engage in interactive behaviour such as discussions, exchanges of ideas, and class competitions.

Therefore, when designing a blended-learning strategy for a programming course, the question arises: how can we engage our students in course programming activities and, at the same time, instil group spirit in them? Our proposal suggests that we do that by adding a competitive flavour to the course activities: for example, programming assignments are modelled after programming competitions (Roberts, 2000; Skiena, 2003), such as the ACM ICPC (ICPC) and the International Olympiad in Informatics (IOI); quizzes are given after each lecture, students get points for it, and a ranking is kept, much like those in some sports, like tennis; we organize tournaments, where students' programs play against one another in a computer game (Ribeiro, 2007).

These activities can be used from elementary stages, and with unconventional languages. As we write, we are teaching an Introductory Programming

Guerreiro P. and Georgouli K. (2008).
TEACHING PROGRAMMING WITH A COMPETITIVE ATTITUDE TO FOSTER GROUP SPIRIT.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 414-421
DOI: 10.5220/0001528704140421

course, using a functional approach (CC2001), with Haskell.

In principle, students of Informatics should be more willing to accept courses with a strong e-learning component, based on an array of computational tools. We have been capitalizing on that, by gradually reinforcing the importance of the online activities, in our own programming courses. Up to a few years ago, the Internet was used as a repository of course material, namely PowerPoint slides and exercises, and that was it. Later, we introduced forums, where students could exchange ideas and get help on their programming assignments. A major development occurred when we started recording the lectures and making them available on the Internet. At about the same time we introduced Mooshak (http://www.ncc.up.pt/mooshak/), an automatic programming judge, which test programs submitted by students and accepts them if they pass a set of secret test cases, or rejects them, if not (Leal and Silva, 2003). More recently, we adopted Moodle (http://moodle.org/), an open-source learning management system that provides a common interface to most e-learning activities. Full integration with the automatic judge and the grading system might be desirable, but we haven't achieved it yet.

In this paper, we report on the usage of our e-learning platform, with an emphasis on the competitive activities that take place through it. The platform is based on three pillars: Moodle, the learning management system, Mooshak, the automatic judge, and the general availability of the lectures on the Internet.

This study applies to the previous home department of the first author. Although it focuses on a programming course, we believe the main ideas can be used on other science courses with a strong problem solving component.

In section 2, we briefly present our programming course and the blended-learning strategy used for web-enhancing it. Next, in section 3, we discuss the range of e-learning activities that the course encompasses for enhancing group spirit and for motivating the students. Then, in section 4, we present the integration of the two platforms we have used for succeeding in the above goals. Finally, we conclude by discussing the experience gained by the use of the e-learning platform, and the perceived effect on the students and on the course results

2 WEB-ENHANCING A PROGRAMMING COURSE

The ACM/IEEE Computing Curricula recommends a three course sequence for introductory programming (ACM/IEEE, 2001), and we have been complying. We are currently using the objects-first approach. The first course is on programming fundamentals, the second on object-oriented programming and the third on data structures and algorithms. In all three courses, we use Java. We report here on our experience with the latest edition (2006-2007 school year) of the second course, Object-Oriented Programming. This is a course on the second semester of the first year of studies of the first cycle of informatics engineering.

Around 300 students registered for the course. About one third are students taking the course for the second time, having failed the previous year. Even though, we have no record of over 100 students. That is, more than 100 students who registered for the course never submitted an assignment for grading. This might seem strange, but it is a consequence of the system of university placement, by which a great number of students enter the university late after the middle of the first semester. It is too late for them to catch up in the course for programming fundamentals, and they seem discouraged to take the object-oriented programming course that comes next, but for which they were registered automatically.

The emphasis of the course is programming with objects and classes, using inheritance and polymorphism, learning the fundamental algorithms, practising Java, developing problem solving skills and understanding software engineering issues. For Java programming, we use Eclipse (http://www.eclipse.org/) and a standard textbook (Horstmann, 2005). There are three 50-minutes lectures and one 2-hours lab per week per student. The overall workload is 68 contact hours plus 100 hours for independent work (projects, self study and evaluation). This corresponds to 6 ECTS credits.

Lectures use both PowerPoint slides and live demonstrations of program development. In the labs, students perform two types of assignments: competition-type problems and scripted tasks. Competition-type problems are problems similar to those used in programming competitions such as the International Olympiad in Informatics (http://www.ioinformatics.org/) and the ACM-ICPC (http://icpc.baylor.edu/icpc/): a problematic situation is described which has to be solved by writing a program. No guidance is provided. In elementary courses, however, students are not prepared to start solving difficult problems immediately. Actually we want to

teach them just that: how to solve programming problems. That's what scripted tasks are for. In this case, the problematic situation is split into a number of tasks that the students must solve one after the other. Those scripts exercise the subjects we are illustrating in the course or the programming techniques we are discussing on the occasion. They are meant to guide the students: do this, do that, consider this aspect, try this new instruction, use this library function, make this experiment, etc.

On a more recent edition of the course, we introduced so-called exercises: numerous, short programming assignments, each requiring writing a few lines of code, focusing on a particular technique or feature of the language.

This approach is carried out in a blended learning environment with three key ingredients: the learning management system, Moodle, the automatic judge, Mooshak, and the systematic recording of the lectures, that are made available online a few hours after they were given in the lecture hall. These are the tools that support independent work by the students. We will now describe them in more detail and the learning activities that students perform with them.

3 BLENDED-LEARNING FOR ELEMENTARY PROGRAMMING

According to the planned workload, students in our course spend more than half of their time working on their own. In our case, this independent work consists basically in reviewing the lectures and preparing the lab assignments. For reviewing the lectures, students can use the PowerPoint slides or actually watch the recording of the lecture. For preparing the lab assignments, students download the task descriptions from the learning management system, program the solutions in Java using the Eclipse environment, and submit them to the online automatic judge.

3.1 Recording the Lectures

We record the lectures with Camtasia Studio (http://www.techsmith.com/camtasia.asp). Actually, we record the computer screen as it is projected on the screen of the lecture hall, together with the voice and image of the lecturer. All this can be handled with minimal setup by the lecturer, and requires no extra staff: the voice is recorded by a computer microphone and the image by a webcam. After the lecture, the recording is edited, in order to remove silent portions and other uninteresting parts, and also to mark the video, so that a table of contents is generated allowing spectators to jump to a specific part of the video. We make the image of the lecturer appear in the lower right corner of the video, but we remove it if it hides any useful information. We currently produce two types of output files, for the Windows Media Player and for the iPod. Editing the recording is a lengthy process, taking about two hours for each hour of recording. This means that, in practise, videos will not be uploaded until a few hours after the lecture.

Having the image of the teacher is a bit superfluous, of course. What matters most is the sound. Besides, the camera being fixed and the teacher moving the frame is often empty or just showing half of the teacher. Also, depending on the classroom arrangement, the teacher may not look at the camera except when he is typing at the computer. Thus the live recording of the teacher is very naïve and crude but it does convey a message of sincerity and proximity that the students appreciate.

One could expect that by having the lectures online without much delay, students would stop coming to the lectures altogether. In fact, that has not happened. We have been using this system for five years now, and the pattern of student attendance in our lectures is not different from that of other courses that do not record. On the other hand, it has removed from students feelings of guilt or embarrassment for not coming to the lecture, thus contributing to a friendlier, more relaxed environment.

Recording the lectures is very easy and it is a little surprising that it has not become common practise yet, now that almost all teachers use a laptop in their lectures and that all students have access to the Internet.

The sequence of recordings does not constitute by itself a video course for e-learning, even if it can be used as such by interested learners who are not regular students. Unless the teacher follows a script very closely, the contents of each lecture will be partly improvised, in response to the students' reactions, and this may be distracting for people who have not experienced at least some of those lectures in the auditorium. Still, if we are able to provide a suitable learning path, the set of videos together with complementary material can be made into an effective blended-learning course. We have had interesting feedback from colleagues from secondary schools who mentioned they are planning to use this idea for helping students who cannot come to school for health reasons.

3.2 Competition–based Learning

In programming courses, most assignments are programs: students are asked to write programs to solve particular problems or to perform certain tasks. When the work is done, students hand in the programs they wrote, together with a written report. This is the standard procedure, but, to use the programming jargon, it does not scale, if done by hand: it is acceptable if you have a few students and a few small programs, but it becomes impossible if you have 200 students, and want them to hand in new assignments every week. The solution is to use an automatic judging system such as Mooshak (Leal and Silva, 2003).

Mooshak was designed to manage programming competitions such as the ACM International Collegiate Programming Contest (http://icpc.baylor.edu /icpc/). Originally it was not a pedagogical tool. However, the facilities it provides can be very useful in programming courses.

Within Mooshak, a contest is a set of programming problems or tasks, to be solved by the contestants before a given deadline. Contestants submit their source files using a browser. Mooshak recompiles and links them with the appropriate libraries, and runs them with a set of secret test files. The resulting output files are compared with the "official" ones and if they all match exactly the task is "accepted". If one of the tests fails, the reply is "wrong answer". In this case, no indication is given of the nature of the discrepancy. It's up to the contestants to figure out, using their wit, the causes of the failure, and then correct it.

The great novelty of this system is that students obtain immediate feedback, and this has great pedagogical value. Most of the times, students are 100% sure that their programs fully satisfy the stated requirements and they are surprised and disappointed when they get "wrong answer". In the past, their wrong solution would be submitted to the teaching assistant, and it might be falsely be taken as correct, if the assistant was not rigorous or did not have time to experiment all the interesting cases. In any case, the assistant would typically return the assignment a few days after, and if corrections were necessary, they would be more painful because the assignment was over in the student's mind.

All programming assignments – exercises, problems and scripts are carried to be judged automatically. When we started this approach, we used scripts only, in place of the so-called "projects" that were, at the time, the only meaningful programs the students would write, typically at the end of the semester. The observation at the time was that the quality of those projects was very unsatisfactory, because the students were only novice programmers. That was understandable, but it left everybody, students and teacher, unhappy. With scripts, students could be guided to program interesting systems, which would be beyond there current capabilities, if they were left on their own.

With time, students became quite good at following scripts. We were surprised that when they were given a problem without guidance, they would feel uneasy. Many would simply give up instantly, knowing that the grade points they had got with the scripts were sufficient for passing.

To counter this attitude, we increased the number of problems and gave them more weight in the final grade.

Still, Mooshak it was difficult to use Mooshak early in the course, because Mooshak was designed to handle input-output programs, i.e., programs that read their input from the console and write their output to the console too. This means that students must know how to read data, not a trivial thing in many programming languages. Furthermore, output must conform exactly to what was specified, not a character more, not a character less. Again, this is not an issue after a while, but it discourages students on their first rounds with Mooshak.

In order to solve this issue, and get more success earlier (meaning, from the first lab), we introduced exercises. With exercises, students merely write a function with a specified interface, and submit it, Input and output is performed by a "framework" in which the students function is inserted, thus freeing the students from those worries. This involved some hacking with Mooshak, but proved very effective, and helped raising the morale in the class.

With Mooshak, the results of all the submissions are public: if my task is "accepted", all my colleagues will know that immediately; is it gets "wrong answer", likewise. This is very instructive: one might think that publicizing failures is counterproductive, but in the end, everybody knows that doing things wrong when you are learning is normal. You don't have to feel ashamed or anxious for not getting the program right immediately: nobody does. And you don't have to be surprised that it is so hard to finalize all the tasks, even the simpler ones: you can watch that everybody else is going through the same troubles, and eventually succeeding. So will you, by working well, calmly and cautiously. All the above instils group spirit which according to Rovai (2002) allows learners to challenge and to nurture each other which in turn affects positively their ability to cope and to learn. On the contrary, what is strange and becomes suspicious is someone who always gets things right at the first attempt. Actually, although the teachers do not take any explicit measures against cheating, some situations like those have arisen and were checked. On the other hand,

we believe Mooshak induces some type of "social cheat control" that works by itself: even if the teachers are not aware that some student or group of students is not playing fair, their colleagues will know. According to Rovai (2002), when there is trust the members of a community will feel safe and subsequently expose gaps in their learning and feel that other members will respond in supportive ways. So, strengthening the community feeling the need of cheating will be minimized.

Mooshak also ranks the students in each contest by the number of submissions and then by the total time, adding in some penalty for submissions that have not been accepted. Although we do not use this ranking in our pedagogy, some students enjoy being the first to submit, thus appearing in the top places. Others, who have not been the first, take the challenge, and try not to get too behind. As a result, most students finish their assignments well before the deadline.

This implicit competition has a number of interesting side effects. For example, although we propose many problems and tasks, students do not complain for excess of work, even if that might be the case for many. We believe this is because they see in the ranking that some colleagues have submitted all tasks in the assignment a few hours or days after it has been published. If those colleagues have done it all say within 48 hours, how can they not be able to do it in two weeks? Also, the common excuse that "there was not enough time to complete the assignment" all but disappeared, and was replaced, when it was the case, by the more objective "I did not have time to complete the assignment (because of other obligations)".

Quite often, after they have submitted, those first students show up in the forums giving advice to their colleagues. Their opinion can be much more convincing that the teacher's, in many cases. For example, in one occasion, some students complained that one of the tasks required a technique that had not been studied yet. Some of the students who had already solved the problem came forward in the forum explained that that was not the case. The complaint was readily defused and teachers did not have to intervene.

Students quickly got used to be rigorous about Mooshak deadlines. In the first assignments, some students postponed their submission to the very last minute and then, as something went wrong, they had to go through the embarrassment of asking a special extension. This phenomenon disappeared as the semester went by.

Overall, Mooshak added a sparkle of excitement to programming, first by the fear of getting "wrong answer" and the joy of getting "accepted", then by the public recognitions of one's achievements, and finally by the implicit competition that it substantiates.

In fact, some students were caught by this competitive spirit, and they formed teams to compete in a nationwide tournament in preparation for the national round of the ACM-ICPC. This tournament is organized by a group of universities and has five stages, one per month from March to September, skipping July and August (http://www.di.uminho.pt /tiup/).

In this latest edition of the course, we pushed the competitive stance a bit further, with the final assignment: we used an IBM game for Eclipse, called Code Invaders (http://www.alphaworks.ibm.com /tech/codeinvaders) and students were called to program the behaviour of a spaceship that was to fight other spaceships, programmed by their colleagues, for energy resources in space. During the preparation phase, students could test their programs fighting against a number of standard spaceships that came with package. They could also upload their own spaceships to the server, thus making them available to their colleagues. At the end of the preparation phase, all teams uploaded their final spaceships separately, and we held a live tournament on the last lecture.

The exercise had pedagogical value, because the techniques required to program the spaceships were precisely those that had just been studied in class. Besides, because of the competitive aspect of the assignment, and unlike common assignments that have a closed specification, students did practise their programming much more extensively than usual, trying to devise better strategies for their spaceships.

As a further incentive, one of the assistants uploaded his own spaceship. Students were thrilled when they could beat it, and boasted about that.

This exercise had a great impact: the word about it spread out and we heard of students from more advanced years, and from other universities, regretting that they did not have a chance to do a similar thing in their courses. This is good to know, of course, but what really matters is that students realized indirectly that their own course was admired a bit enviously by others, and this made them feel proud of being a part of it.

We planned this exercise also as a means of making the results of our course more understandable to non-programmers, namely the students' families and friends. Computer programs are abstract entities, which can give tremendous pleasure to create, but this pleasure often stays with the creator and cannot be transmitted. With programs with a strong visual component, such as this game, students can indeed exhibit their newly acquired skills to their own entourage (Ribeiro and Guerreiro, 2007].

3.3 Using the Learning Management System

The online meeting point for students and teachers was the course Moodle page. Moodle is the learning management system in use at our department, even though there is no explicit directive mandating its use. Therefore, it is adopted or not depending on the preferences of the professors responsible for each course. We use it extensively, for posting information about the course, for publishing assignments, for collecting the reports of the students must submit in relation to their scripted tasks (the tasks themselves are submitted to Mooshak), for conducting surveys, for managing forums, and for online quizzes. The first four are quite common, so we will comment only on our utilization of forums and quizzes, in those aspects that relate to the somewhat unusual competitive approach that we advocate.

3.4 Forums

One of the characteristics of our course is transparency: all those interested in the course may follow what we, teachers and students, are doing. That is one of the reasons why we record the lectures and publish them. Also, the course page in Moodle is open to guests, and we have had indeed interesting feedback from people not related to the university.

As sense of community and social presence is related to learning as some research suggests (Culter, 1995); Russell, 1999; Rovai, 2002) we put a big effort on building and nurturing a sense of community, letting the students express themselves in all issues. This is a delicate subject, since students may be reluctant to make any negative remarks or criticism, for fear of reprisals. Our strategy to gain their confidence has been to provide many forums, one for each important issue. For example, even before classes started, a forum was opened so that students could discuss whatever they wanted about the preliminaries of the course. Every time grades are published, a forum is created so that students can publicly report errors. Right after the exam, we had a forum where students could exchange their impressions on the exam. And each posted assignment has a corresponding forum.

The forums for the assignment are very popular and students use them mostly to get help. One might fear that other students would simply post their own solutions, in response to the questions, thus ruining the exercise for the others, but that has happened only once or twice, early in the semester, and we grabbed the opportunity to explain that that was not adequate. From then on, more advanced students, some of which have taken the course in previous editions, do provide help, but always in a way that does not defeat the purpose of the exercise.

Typically, once a new assignment is published and the corresponding forum is opened, the first discussions concern clarifications of the problem. In that phase, the teachers must pop in and respond promptly. After the first students have had all corresponding tasks accepted in Mooshak, which usually happens around 48 hours after the beginning, they take over, and teachers merely have to watch from afar, intervening only sporadically. This is a very interesting pattern, not only because it saves work to the teachers, but because it helps create a good atmosphere, and gives visibility to the leaders of the class. In these circumstances, their advice can be more effective than the teachers'.

We also used the forum for students to publish the graphical output of their programs, for all to see and enjoy. One of the assignments had to do with creating fractals, drawing them on a window, and making a simple animation. Although the automatic judge can perform a certain kind of validation on graphical programs, in this case we wanted students to experiment and invent their own fractals, and therefore it was not appropriate to accept these submissions. Instead, the evidence that the task had been accomplished lied on having the fractals and the animations posted in the forum. Actually, some students posted their work in other sites, such as YouTube (http://www.youtube.com/), and merely added the links in the forum.

Having all works visible in a forum had some surprising effects. First, the students could freely comment on their colleagues results, and again, we could observe the student leaders encouraging the others. Second, many students first published a simple fractal, only to guarantee that the assignment had been completed but returned later, to publish more substantial examples. Of course, this healthy emulation would not be possible if the assignments had been handed in to the teachers alone, even if a selection of best works was to be published later.

Actually, as far as emulation goes, we noticed that overall the fractals were not as elaborate as last year's. Last year, in a similar setting, one of the assistants published his own fractals, early in the submission period. This immediately established a touchstone and challenged the students to match it or do better. Without such a reference point, the average quality of the fractals was lower.

Indeed, this observation confirms that it is a good idea to add a degree of challenge in the assignments, whenever possible, thus capitalizing on the natural competitive attitude of the students. This was precisely what happened in the Code Invaders assignment, where the first goal was to beat the spaceship of the assistant.

This type of assignment, bearing a competitive twist, stimulates the creativity of the students, not only in finding good solutions to the problem that was presented, but also in solving related problems that were not clearly identified and that they have to investigate by themselves.

3.5 Quizzes

Moodle has a tool for online quizzes. We decided to use it to create a small quiz after each lecture, with 10 questions about what had been discussed. Each quiz would be open for 60 hours after the lecture. The goal was to invite students to review the lecture, shortly after they attended it, or, to have them watch the recording, in case they had not been present. Students would get points for their final mark in the course if their score in the quiz was 70% or more. They could take the quiz as many times as they wanted, with no penalty.

In the final survey, we inquired about the quizzes: 64% said the quizzes are useful, 14% said they are a waste of time, 60% tried to make them all, or most of them. However, the most interesting was the response we got to the question "Do you quit after having reached a score of 70%?" Only 32% said they did, meaning a great majority was motivated to find the right answer to all the questions, and not merely to pass. Actually, we believe they were driven by the puzzle-like nature of the questions, some of which were brain-teasers that challenged their wit, sometimes in areas not directly related to the contents of the course.

In this case, students had no access to the scores of their colleagues. Thus, when they were retaking the quiz to reach a higher score (higher than 70% anyway) it was for their own satisfaction and self-esteem. We find here another example, albeit of a different nature, where a certain form of competitiveness can help raise the level of the participation.

We had experimented with the quizzes on a previous edition of the course, but at that time students were not rewarded with points. Although the novelty made quizzes very popular at first, with hundreds of students replying, very soon the numbers decreased, and by the end of semester only a dozen were doing it. This is a clear sign that as much as an assignment can be fun and admittedly important, if there is not a clear and practical benefit to be withdrawn from it, students will skip it.

Conversely, using the points as a lure, we managed to have all the active students doing the quizzes, adding one more common activity, thus fostering group spirit. There was also a sparkle of competition involved, as some students relished on being the first to solve the quiz, seconds after it was published.

4 PLATFORM INTEGRATION

The fact that we use two systems, Moodle and Mooshak, goes against the conventional wisdom of e-learning, according to which students should be exposed to a single interface. Indeed, the situation is a bit more complicated, because for certain tasks students must use the information system of the university. On the other hand, since students develop their programs using the Eclipse environment, also process of explicitly submitting to the automatic judge is a bit awkward.

Three integration efforts are in order: on the one hand, Moodle plus the information system of the university. This will be a major project that cannot be carried out without proper central support. Then we might want to integrate Moodle and Mooshak. This project is within our reach, both platforms being open source. However, rather than integrating Moodle and Mooshak only, it would be preferable to integrate Eclipse and Mooshak also, so that students could submit their programs to the automatic judge without leaving Eclipse and then have the result available in Moodle. On the other hand, other languages use different development environments, and at times, it is acceptable to use a plain text editor, together with a command line.

5 CONCLUSIONS

Exploiting of the natural competitive attitude of young people can help raise the level of motivation and participation in introductory programming courses, and we can use the tools of our e-learning platforms for that. On the one hand, we can use the learning management system to establish a permanent showcase of what happens in the course, namely to publish the lectures as they are given and to display the performance of the students. In the latter case, we must handle privacy issues carefully, of course. Actually, one of the ways to ensure privacy and at the same time distinguish the best students is to publish rankings for each assignment separately, and refrain from collecting the "current standing". It is acceptable to be late once or twice, or to skip one task from time to time, but it would become embarrassing to appear in the lowest places in some global standing for the course.

We can also design assignments which pit students against one another, individually or in teams. Various disciplines may adapt different strategies in this area. In programming, we can get inspiration in the plethora of existing computer games and programming competitions problems (Paxton, 2007; Ladd, 2005). We can also use assignments in which

the results are published as they are submitted, thus challenging the other students to do better. In both cases, it pays to mix in the work of the assistants, or of students from previous years, so that the students have a clear reference with which they can evaluate their own results, right from the start.

Carrying out many small quizzes is an effective way of helping the students to remain synchronized with the pace of the course. As a side benefit, we build up a pool of questions than can be used in other occasions. Also, by analysing the answers to each question, we may identify issues that need to be reinforced. And in classes with many students, in which many do not come to lectures and some drop out during the semester, with the frequent quizzes we can keep track of set of students that remain active.

If presented regularly with questions that test the students' ingenuity applied to the contents of the course, the quizzes can be addictive, and students solve them not as an obligation but for fun and excitement. They also discuss them with their colleagues constantly, and in the process they discuss the course issues that go with them, which is another benefit.

Another interesting idea is to design students' competitions modelled after popular television contests (O'Shea 2006). Given that some of these contests have become part of the popular culture, we believe they can engage students outright. One such game is 1 vs. 100 (http://en.wikipedia.org/wiki/1_vs._100). This game involves multiple choice questions (which we could reuse from the quizzes). Many players participate in each match, which is a good thing, because we can accommodate large numbers of students. As an extra piquant, the winners could get bonus points. The infrastructure could be build using the computers in the labs and simple ad-hoc software.

From the point of view of the teachers, all these activities should be carried out harmoniously from within a single platform, with a consistent interface. This requires a certain effort of tool integration that has yet to be done. Students of informatics, on the other hand, seem comfortable in jumping from tool to tool to perform the various activities. This observation will not hold us back from trying to make the learning experience of our students more rewarding, by providing the most adequate combination of tools for the job.

REFERENCES

ACM/IEEE, 2001. Computing Curricula. *Computer Science Volume.* Available: http://www.sigcse.org/cc2001/

Cutler, R. H., 1995. Distributed presence and community in cyberspace, *Interpersonal Communication and Technology: A Journal for the 21st Century*, 3(2). Retrieved January 2007 from: http://www.helsinki.fi/science/optek/1995/n2/cutler.txt

Felder, R., 2004. Teaching engineering at a research university. Problems and possibilities. *Educatión Química*, 15(1), 40-42.

Horstmann, C., 2005. *Java Concepts*. 4th ed., Wiley.

ICPC. The ACM-ICPC International Collegiate Programming Contest (accessed November 2007), http://icpc.baylor.edu/icpc/

IOI. International Olympiads in Informatics (accessed November 2007), http://www.ioinformatics.org/

Ladd, B., Harcourt, E., 2005. Student competition and bots in an introductory programming course. *Journal of Computer Sciences in Colleges*, 20(5), 274-284.

Leal, J. P., Silva, F., 2003. Mooshak: a Web-based multi-site programming contest system, *Software Practice & Experience*, 33(6), 567-581.

O'Shea, P. 2006, On using popular culture to enhance learning for engineering undergraduates, Journal of Learning Design, 1(3), 73-81,

Paxton, P., 2007. Programming Competition Problems as a basis for an Algorithms and Data Structures Course. *Journal of Computer Sciences in Colleges,* 23(2)*,* 27-32.

Ribeiro, P., Guerreiro, P., 2007. Increasing the appeal of programming contests with tasks involving graphical user interfaces and computer graphics. *Olympiads in Informatics*, vol. 1, pp. 149-164.

Roberts, E., 2000. Strategies for encouraging individual achievement in introductory computer science courses. *Thirty-First SIGCSE Technical Symposium on Computer Science Education*, pp 295-299, ACM Press.

Rovai, A., 2001. Building Sense of Community at a Distance. *International Review of Research in Open and Distance Learning*, vol. 3(1). http://www.irrodl.org/index. php/irrodl/article/view/79/153

Russell, T. L., 1999. The no significant difference phenomenon. Chapel Hill, NC: Office of Instructional Telecommunications, North Carolina University.

Skiena, S., Revilla, M., 2003. *Programming Challenges: The Programming Contest Training Manual.* New York, NY: Springer-Verlag.

Wang, H. H., & Fwu, B. J., 2001. Why Teach? The Motivation and Commitment of Graduate Students of a Teacher Education Program in a Research-Oriented University. *Proceedings of the National Science Council*