Informatics in Education, 2007, Vol. 6, No. 1, 153–162

Visualize and Open Up

Michela PEDRONI, Till G. BAY
Chair of Software Engineering, ETH Zurich
Clausiusstrasse 59, 8092 Zurich, Switzerland
e-mail: {pedronim,bay}@inf.ethz.ch

Received: December 2006

Abstract. Motivating students of the Nintendo generation for Computer Science can only be achieved by providing them with an exiting and fresh CS1 course. The article describes the experience of redesigning the introductory programming course at ETH Zurich and shows how the combination of state-of-the-art visualizations with open project assignments enlivens students' enthusiasm for programming. It shows the setup and the involved libraries, provides example applications that were built in the course, and presents the data gathered in the evaluation of the open assignment.

Key words: CS1, introductory programming, open assignments, visualization, multi-media.

1. Introduction

Today's students are used to graphical programs; the levels are set by the computer games they play in their spare time. Graphical and multimedia applications are what they want to produce. Together with others (Guzdial and Soloway, 2002; Feldman and Zelenski, 1996; Becker, 2001), we believe that to "make CS courses fun" (Mahmoud, 2005) and motivate students by using state-of-the-art graphics/multimedia libraries together with freedom for creativity is critical to their success in learning.

Over the past three years, we have significantly changed the way we teach programming to first semester CS students by taking a new approach, called the Inverted Curriculum (Pedroni and Meyer, 2006). The Inverted Curriculum is an objects-first, component-based approach relying on a large software framework with a strong visual aspect. Using this approach, students start out as consumers of library components by using their abstract interfaces, before they progressively discover the implementations. At the end of the course, they are capable of producing similar software elements themselves. This results in a topic introduction that is outside-in, starting with the notion of object, method call, and class interface, while the internals, such as control structures, local variables, and assignments, are covered later in the course. The Inverted Curriculum allows students to produce interactive graphical applications right from the start, taking advantage of the power of provided libraries.

A project assignment on which students work in small groups over a period of 4–5 weeks complements the course redesign. While in the first iteration of the course

(2003/2004) the project assignment specified in detail what kind of application the students should produce, in the subsequent two iterations the project was completely opened up. The main intentions of using an open project instead of small, controlled tasks were:

- **Teamwork.** Since teamwork is encouraged, students learn to collaborate with their peers.
- **Quality of code.** Style, good design, and clean work are emphasized as students need to use the code previously produced by advanced students to finish the project.
- **Project development.** Students experience all the phases of the project development starting with the formulation of a project idea, defining the design of the system, implementation, testing, and the delivery of the final product.
- **Adaptation to skills.** With the extreme diversity of student backgrounds in introductory programming, an immediate goal is to keep the course work feasible for the students with little experience, while not boring the experienced programmers of the class. The open project assignment helps to reach this goal since the tasks to be done can be adapted to the capabilities of each of the teams.
- **Creativity.** Keeping the topic of the project open stimulates the imagination of the students and allows for great creativity.
- **Motivation.** Learning to program is difficult and needs a lot of practice. Only if students are motivated to invest the time, this can be achieved.
- **Visibility.** The outcome of the project is entirely the students' own merit; it is an achievement they can be proud of and show to relatives and friends.

The idea of open project assignments is not novel, but only a few instructors have reported on their experience with it. Sindre, Line, and Valvag (2003) let students freely choose what kind of game they produce. Parberry *et al.* follow their example also using games and state "[...] that the element of creativity, student morale, the quality of the resulting games, and the outcomes all suffer when any kind of constraint is placed on the game being developed" (Parberry *et al.*, 2005). We agree with Parberry and believe that we can and should go even one step further. We refrain from narrowing down the domain of projects and therefore put no limitations to students' creativity, thus motivating exceptional results.

This paper presents the implementation of the open assignment and accompanies it with the results and feedback gained. Section 2 describes the general setting and the libraries used during the course, while Section 3 explains the implementation of the open project assignment. Section 4 expands on the results and feedback provided by the students, and Section 5 presents the conclusions.

2. Setting

2.1. *Course Setup*

Since winter 2003 the Introduction to Programming course for first semester Computer Science majors implements the ideas of the Inverted Curriculum. The number of students

that participated in the courses so far is approximately 600 (250 in 2003/2004[1], 180 in 2004/2005, and 170 in 2005/2006).

Introduction to Programming is a mandatory 8 ECTS credits course for CS masters and the only Computer Science course in their first semester. In the second semester, a course called Data Structures and Algorithms is held as a follow-up to Introduction to Programming. The other courses of the first semester are mostly math courses, laying the basic knowledge for advanced studies in CS.

The course consists of seven weekly lessons, where four are plenary lectures held by the professor and three are held in groups of up to 25 students by graduate and doctoral student tutors.

The semester stretches over 14 weeks with Christmas break after week 9. We divide the semester into two parts: the first part from week 1 to week 8 where students are handed out weekly assignments that they are supposed to solve alone. In week 9 (before Christmas break), students get the description for the project assignment which they solve in small groups until the end of the semester. Furthermore, students solve up to three sit-in assignments spread over the semester to help them assess their current status and get a feeling on how they are performing compared to their classmates. All the handed in weekly assignments, mock exams, and the project are corrected (but not graded) by the tutors.

Instead of grading assignments in first-year courses, ETH has the policy that students need to get a certificate to be admitted to the exam. The exam determines whether students are allowed to move into their second year of study. It takes place after the second semester toward the end of the semester break. To get the admittance certificate, students need to do about 70–80% of the weekly assignments and mock exams (not necessarily correctly, but showing a clear effort), and to submit the project assignment. While having no grading during the semester is quite unusual, it allows even newcomers to programming to take a long-term approach to learning and prevents them from getting obsessed about their grades during the semester.

2.2. *Technical Foundation*

The technical building grounds of the course are the libraries used to develop the multimedia applications. The approach focuses on the use of libraries early in the Computer Science education. By using libraries that were written by others, students learn to read code. This clearly is one of the most important skills an engineer of our field needs to provide. A consequence of this activity is that programming patterns are studied in practice rather than in the abstract. By exploring the libraries, students see how others have solved a problem.

The libraries provide a vast API which makes them suitable for the open assignment. Students can choose to use only the simple mini frameworks provided by the libraries,

[1]The number of students in 2003/2004 is exceptionally high. This is probably due to the fact that in 2001, 2002, and 2003 some of the Swiss high schools shortened the duration of high school by one year and thus had two age-groups graduating at the same time.

or they can go for their full power. By using the mini frameworks, complexity remains hidden and the students can concentrate on their own application design (e.g., they can use a ready made keyboard handling framework). But if they want to do more, they can use the finer grained and more complicated API that is also provided (e.g., they can devise their own keyboard handling facility).

2.2.0.1. The two libraries used heavily in the courses are described in the following two subsections. They are both mostly developed and maintained by the students. Many of the students that worked with the libraries in their CS1 course, choose to contribute later either as part of a thesis or voluntary work. These students thus help to improve the quality as well as the capabilities of our foundation. Of course many of the students are motivated to contribute to the libraries because of the bugs they have encountered when using them.

2.2.1. *EiffelMedia*

Recently Carter (2006) has shown that the number one reason for boys to study Computer Science is their interest in computer games whereas for girls it was the possibility to use computers in other fields – in both cases one can argue that the visualization capabilities are of great importance. *EiffelMedia* (Bay, 2006b) provides visualization capabilities that are state-of-the-art and impress and motivate the students. As mentioned above, the library is maintained together with the students and is open source. *EiffelMedia* has a rich set of frameworks that allow building multimedia applications such as games like Antworld shown in Fig. 1. The features of the library include 2D graphics, sound support, video decoding, 3D graphics, networking support, a widget toolkit and many more.

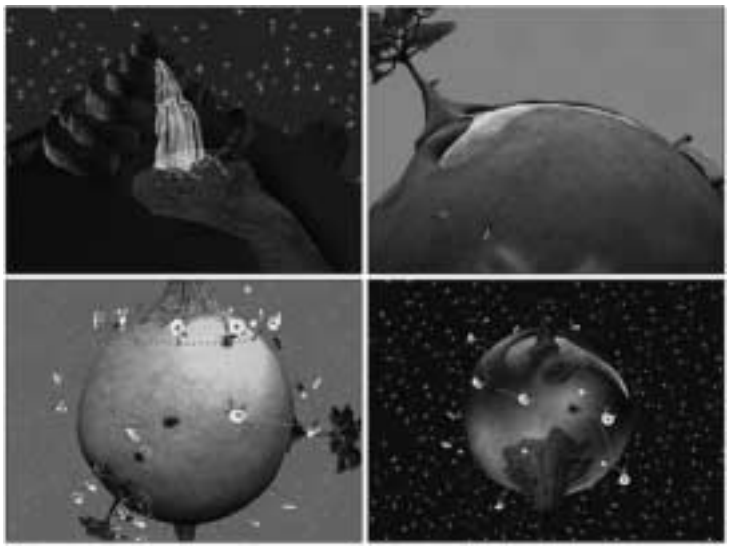

Fig. 1. Day and a night in *Antworld*.

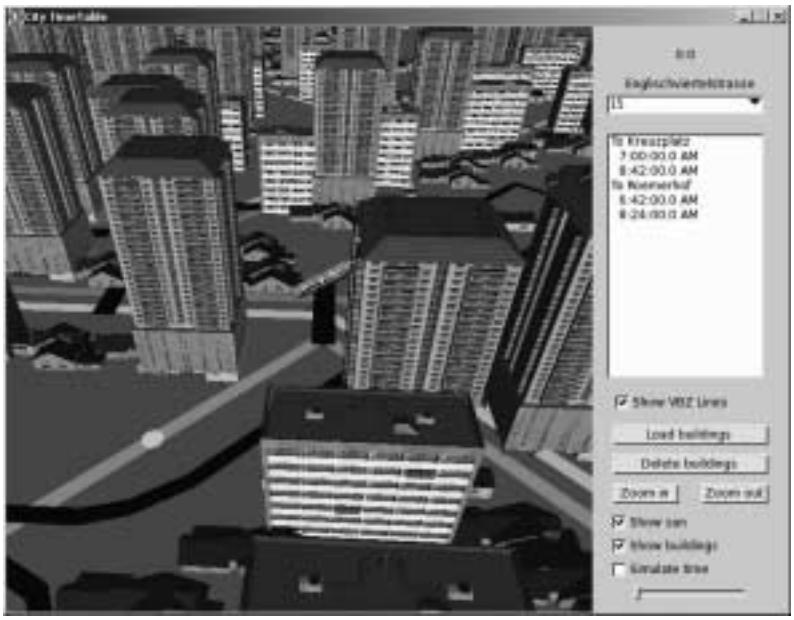

Fig. 2. 3D model of a city in *Traffic*.

2.2.2. *Traffic*

The second library the course relies on is *Traffic* (Pedroni, 2006). *Traffic* models the public transportation system of a city consisting of transport lines (e.g., metro lines, bus lines, light rail) and places (metro stations, landmarks). It contains city modeling classes, visualization facilities for the display of city maps and additional supporting classes for building applications. *Traffic* comes with several applications: One of them – *Flathunt* – is a strategy round-based game, others are either example applications to show certain features of the library, or they accompany the textbook *Touch of Class* (Meyer, 2006), currently under development. Both – *Traffic* and its applications – have been specially developed for the use in classroom with the Inverted Curriculum. *Traffic* uses *EiffelMedia* to visualize the city model.

Both of the libraries are particularly well suited for using in an open project because they offer a wealth of functionality and highlight visualization and multimedia. With this foundation, we feel comfortable to give students the freedom of an open project assignment.

3. Procedure

For the open project assignment students were required to choose a partner of equal strength, so that they could design their project idea to be challenging for both of them and fitting their programming skills. In previous years, the number of students per group was

three, but this was changed to two students in 2005 to prevent organizational problems such as code synchronization and task distribution among students.

Students used a wiki to organize their project. As a first task, each group generated a new page for their project and a description of what their application should do. Using this platform, they were able to discuss issues online, upload any of the intermediary results (such as the documentation and source code), and – if interested – they could browse other groups' projects at any point.

We guided students by officially giving them two options in deciding on the project idea: Option A was to implement an extension to *Flathunt*/*Traffic* or *EiffelMedia* with examples of possible projects of reasonable size. Since the students had already been using these libraries during the course, most of them chose this path. Option B was totally open, telling them to do whatever they wanted with the one restriction that their idea had to get approved by the tutor. As a result of the openness, almost every group was asking approval from their tutor to also get feedback on the feasibility of their project.

As a consequence of timing problems that some of the students had in previous years and feedback stating that the project description should emphasize the importance of the software design phase, students had to meet four milestones. The first milestone was due after a very short period of four days. For this milestone, they had to hand in a first project idea and a description on the wiki. The next milestone was planned for right after Christmas break where they had to give a more precise description of the requirements and the task distribution among the partners. The third milestone followed one week later, where they should provide a document describing the OO design of their system. The last milestone was two weeks later (in the last week of the semester) and they had to hand in the code and a short developer guide.

The project was complemented by having each group give a short presentation to their tutors and fellow students during the exercise session of the last week of the semester. In each of the exercise groups one of the projects was elected to be shown in a subsequent event called the *Object-oriental bazaar*. The bazaar was the closing lecture of the semester and was open to the public. Students were asked to invite their friends and the department was encouraged to come and see what the first-semester students had achieved. In a first round, the elected projects were presented to the curious public, but the second half was devoted to all the projects where each group was present with a laptop showing and explaining their projects to interested students, tutors, professors or other guests. This happening was greatly appreciated by the students since they felt that we valued their efforts.

During the project time, the plenary lectures held by the professor covered advanced topics such as event-driven programming, an in depth discussion of data structures, and an introduction to software engineering. The work for the professor stayed the same as during the first weeks of the semester. For the tutors the project phase resulted in more individual mentoring tasks which they generally tried to schedule during part of the three exercise lessons per week. Since there were no weekly assignments during the implementation phase, the time that was needed initially for corrections of weekly assignments could be used for answering the occasional e-mails with questions and for preparing

feedback on the projects. Most of the tutors appreciated the project phase because the time spent with students was more interactive while the amount of work for them stayed approximately the same.

4. Results and Student Feedback

The open project assignment of the course in 2005/2006 resulted in over 70 applications (Bay, 2006a). About 50% of these applications were games written with *EiffelMedia* such as two Sudoku solvers, some Battleship implementations, the traditional Pong game, spaceshooters or jump'n'run games. Another 25% were either extensions to *Flathunt* or extensions to the *Traffic* library such as a multiplayer *Flathunt*, a timetable *Traffic* extension, or modifications of the existing game *Flathunt*. The remaining 25% were applications or libraries of any kind, such as an InstantMessenger, a math parser, a browser, collection classes, or an RSS feed reader. The games usually came with 2D graphical user interfaces, but also included five console applications, and a few games with amazing 3D visuals, such as *Antworld* shown in Fig. 1.

The number of source code lines (not counting comments) produced by the students in 2005/2006 ranged from 418 to 9744 while the number of classes ranged from 2 to 114 (see Fig. 3). The average amount of lines of code was 1885 while the average number of classes was 17. Clearly, projects like Antworld (5891 loc, 65 classes) or Hoovercraft (9744 loc, 114 classes) were beyond our expectations. The project with only two classes was a "Connect four" console application and was chosen by the students to train conditionals and loops. As such, this project served its purpose, and students learned much

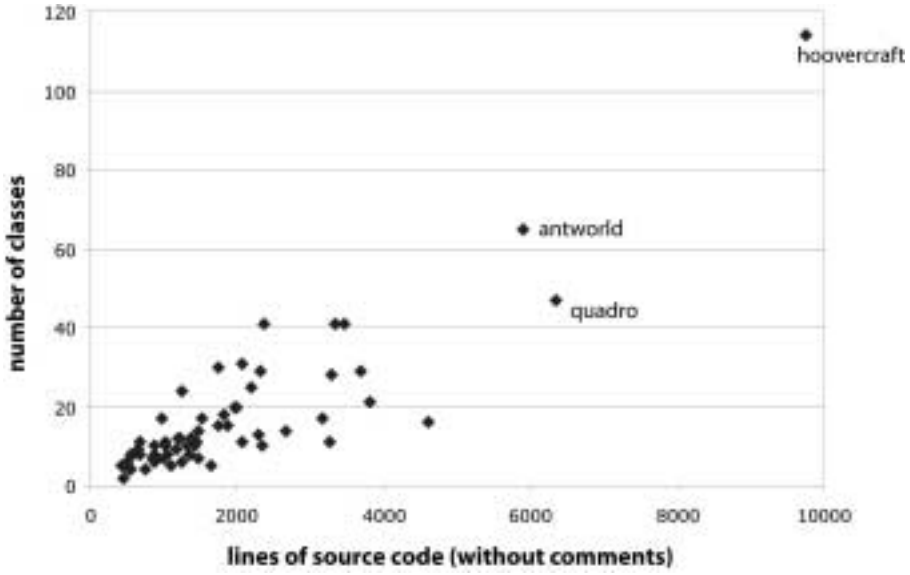

Fig. 3. Lines of code and number of classes for the projects in 2005/2006.

by doing it, but it lacked a good object-oriented design. To ensure that students were aware of where they needed to improve, each project group received feedback and we encouraged them to continue their work by improving or redesigning their applications.

As part of the course evaluation, students stated whether they liked doing the project stretching over several weeks. The overall answers were positive, the average grade always ranged from 3.9–4.0 points out of 5 for all three iterations of the course. In the evaluations for 2004/2005 and 2005/2006, they also rated whether they appreciated being able to freely choose the project task. The mean grades for this question increased significantly from 3.6 (in 2004/2005) to 4.5 (in 2005/2006) out of 5. The increase is mostly due to the fact, that much work had been done on *EiffelMedia* in between iterations and that we guided students more, thus improving their timing.

5. Conclusion

Motivating today's students to learn programming is a challenge and can only be achieved if the courses live up the expectations of the Nintendo generation. The combination of state-of-the-art multimedia libraries with the freedom of open project assignments lets students strive for excellent, innovative results, and reaches the ultimate goal: increased motivation of students. Moreover, the approach allows adapting the level of difficulty to the students' prior knowledge and emphasizes the importance of code quality and teamwork. We encourage open project assignments provided they are embedded in a well structured course, a supportive environment, and a project framework that allows the students to validate their progress.

References

Bay, T.G. (2006a). *Collection of Games and Applications Built in cs1*.
Available online at: `http://games.ethz.ch/`.

Bay, T.G. (2006b). *Eiffelmedia – the Multimedia Library for Eiffel*.
Available online at: `http://eiffelmedia.origo.ethz.ch/`.

Becker, K. (2001). Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, **17**(2), 23–33.

Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. In *SIGCSE '06*: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York, NY, USA, pp. 27–31.

Feldman, T.J., and J.D. Zelenski (1996). The quest for excellence in designing cs1/cs2 assignments. In *SIGCSE '96*: *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York, NY, USA, pp. 319–323.

Guzdial, M., and E. Soloway (2002). Teaching the nintendo generation to program. *Commun. ACM*, **45**(4), 17–21.

Mahmoud, Q.H. (2005). Revitalizing computing science education. *Computer*, **38**(5), 100–99.

Meyer, B. (2006). *Touch of Class – Learning to Program Well with Object Technology and Design by Contract*.
Available online at: `http://se.inf.ethz.ch/touch/`.

Parberry, I., T. Roden and M.B. Kazemzadeh (2005). Experience with an industry-driven capstone course on game programming: extended abstract. In *SIGCSE '05*: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York, NY, USA, pp. 91–95.

Pedroni, M. (2006). *Traffic*.
Available online at: `http://traffic.origo.ethz.ch`

Pedroni, M., and B. Meyer (2006). The inverted curriculum in practice. In *SIGCSE '06*: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York, NY, USA.

Sindre, G., S. Line and O.V. Valvag (2003). Positive experiences with an open project assignment in an introductory programming course. In *ICSE '03*: *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA, pp. 608–613.

M. Pedroni is a PhD student and teaching assistant at the Chair of Software Engineering, ETH Zurich. Her main research interest and focus of her PhD topic is on issues related to teaching object-oriented programming and defining curricula and courses. She is the maintainer of Traffic, a library used to teach object-oriented programming to first semester CS students. She joined ETH in 2003 after a master's degree at ETH Zurich.

T.G. Bay is a PhD student at the Chair of Software Engineering. He is working on a distributed software development platform. He is the maintainer of a comprehensive multimedia library – EiffelMedia. He joined ETH in 2003 after a master's degree at ETH Zurich and EPF Lausanne.

Vizualizacija ir atvirumas

Michela PEDRONI, Till G. BAY

Motyvuoti šiuolaikinius studentus mokytis kompiuterių mokslo galima tik sukūrus įdomų ir naujovišką kompiuterių mokslo kursą. Straipsnyje aprašoma patirtis, kaip perkurtas programavimo pradmenų kursas Ciuricho aukštojoje technikos mokykloje ir parodo, kaip derinant vizualizacijos meną su atvirųjų projektų užduotimis galima sukelti studentų entuziazmą programuoti. Straipsnyje aprašomas įgyvendinimo procesas, rašoma apie konkrečias programuotojo bibliotekas, pateikiamos pavyzdinės kurse parengtos programos, pristatomi duomenys, surinkti vertinant atvirąją užduotį.