Int. J. Engng Ed. Vol. 25, No. 3, pp. 534–546, 2009
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Making the Computing Professional Domain More Attractive: an Outreach Program for Prospective Students*

BRURIA HABERMAN[1,2], CECILE YEHEZKEL[1], HANANIA SALZER[3]
[1] *Davidson Institute of Science Education, The Weizmann Institute of Science, P.O.B 26 Rehovot 76100, Israel. E-mail: bruria.haberman@weizmann.ac.il cecile.yehezkel@weizmann.ac.il*
[2] *Computer Science Department, Holon Institute of Technology, Holon 58102, Israel.*
[3] *The Science and Technology Education Center, School of Education, Tel-Aviv University, Tel Aviv, Israel. E-mail: salzerha@post.tau.ac.il*

The recent rapid development of the field of computing has posed challenges in educating newcomers, in particular, in attempting to bridge the gap between school and the contemporary world of computing. To meet this challenge, we developed a novel outreach program for prospective high-school students aimed at exposing them "directly by leading experts" to state-of-the-art computing research and development. The program includes enrichment meetings and software development projects under the supervision of experts. Six hundred students participated the last four years in enrichment activities; 86 of these students developed high-level software projects. A long-term formative evaluation of the outreach program has been conducted. So far, the study's findings have indicated that the program contributes to developing a culture of learning befitting the dynamic world of industrial computing, thus providing the students with an entry point into the computing community of practice.

Keywords: Software engineering; project development; mentoring; self-learning; software design skills; scaffolding activities

INTRODUCTION: THE "COMPUTER SCIENCE, ACADEMIA & INDUSTRY" PROGRAM

THE RAPIDLY DEVELOPING WORLD of computing-based technology will need a continuous, heavy flow of talented students to be educated and trained as engineers in order to carry out the achievements needed to maintain our students in the forefront of our profession and to keep abreast of technological innovations of the 21st century. Since the number of students opting for computing and specifically software engineering education has seemingly not been as large as before, it is important to plan outreach activities for prospective students aimed at making the field more attractive. This is particularly challenging due to the existing gap between school education and the "real world" of computing that is mainly related to content, learning culture, and professional norms:

- The basic fundamentals and core technologies that are introduced in school constitute the basis for students' understanding of computing; however, they rarely include state-of-the-art computing research and development as well as new, rapidly evolving directions in the field. Hence, we suggest that in addition to learning fundamentals, students should also become acquainted with enrichment through advanced topics.
- The traditional style of teaching and learning in school is usually designed so that students can acquire explicit knowledge based on a thorough understanding of the topic learned. However, this approach alone might fail to educate students to become self-learners who are capable of navigating in the rapidly growing world of knowledge [1, 2]. Hence, students should also experience a "taste-based" breadth-oriented learning approach.
- School projects enable students to experience software design and development; however, they do not resemble actual software engineering industrial processes, and the products are rarely applicable to real-world situations. Hence, students should be encouraged (1) to participate in small-scale activities that simulate "real-world" situations, and (2) to develop comprehensive projects under the guidance of professionals who are representative of the computing community of practice.

Based on the above considerations, we developed the "Computer Science, academia, and industry" extracurricular program, designed especially for talented high-school students who major in

* Accepted 3 March 2009.

computer science (CS) or software engineering (SE). The main goal of the program was to bridge the gap between school and the "real world" of computing by "opening a window" to the academic and practical challenges that characterize the CS/SE community of practice. Specifically, the program aims at exposing students "directly by leading experts" to state-of-the-art research, advanced technologies, software engineering methodologies, and professional norms. In addition, the program explores the synergy between theory and practice so that students will gain a broad sense of contemporary research and development in the field. The program is planned so that students will experience a style of learning that will prepare them to navigate in the ever-changing, dynamic world of digital and other rapidly increasing knowledge.

The two-year program blends formal and informal learning and includes enrichment meetings, field trips, and software development projects under the supervision of experts. Through these activities, the program aims at bridging computing and software engineering education to the fundamental studies of computer science at the high-school level. A detailed description of the program (related to its setting and contents) is presented in [3].

The first stage of the program (Stage-A) is designed for 11th grade students and consists of a seven-month enrichment workshop. Each monthly (after school) meeting consists of two lectures by leading representatives of the community of practice and related group activities. Contemporary topics are introduced, the industry's professional norms are discussed, and advanced technologies and methodologies are demonstrated. The program's activities are supported by a web site that provides an opportunity for experts to communicate with students between meetings.

The second stage of the program (Stage-B) is designed for a small group of graduates from the first stage. During the 12th grade, the students develop comprehensive software projects in a variety of scientific and industrial fields under the apprenticeship-based supervision of professional mentors (e.g. scientists and engineers from academia and the hi-tech industry). The project development process lasts 9–10 months. Examiners chosen by the Ministry of Education evaluate the students' final products; the students' grades are considered as part of their high-school Matriculation Diploma.

The program has been conducted for the last four years and serves as an example of a successful partnership among academia, industry, the Ministry of Education, and K-12 educators. Six hundred students around the country participated in enrichment activities; 86 of these students developed high-level software projects.

A long-term formative evaluation of the program has been conducted regarding:

1) students' attitudes towards the "different-from-school" style of learning that characterizes the program,
2) their performance in developing projects.

So far, the study's findings have indicated that the program contributes to developing a culture of learning befitting the dynamic world of industrial computing, thus providing the students with an entry point into the computing community of practice. For example, we found that project development experience under the supervision of professional experts may motivate students to acquire more in-depth knowledge in computing, as well as to promote creativity, enhance self-learning and inquiry ability, and contribute to establishing professional norms [3, 4].

In order to analyze project development processes and their outcomes, we collected data on the timing of mentor-students' face-to-face meetings, email, phone conversations, and students' post-project reflection and attitudes. In addition, we characterized profiles of project developers and identified the difficulties that students encountered during the different phases of project development.

PROJECT DEVELOPMENT ACTIVITY

Background and motivation

The basis for encouraging prospective students to enroll in academic software engineering studies should be initiated in the early stages of their education and should focus on two main phases:

1) portraying the essence of the field to newcomers,
2) developing desirable habits, skills, and attitudes.

Hence, it is important to bridging computing and software engineering education to the fundamental studies of computer science at the high-school level. Indeed, educators have long noted the importance of teaching software designing skills to high-school computer science students (e.g. [5, 6, 7]).

The 2004 Software Engineering curriculum states that incorporating real-world elements into the undergraduate curriculum is necessary to enable effective learning of software engineering skills and concepts (Software Engineering 2004 Curriculum Guidelines) [8]. According to the *Situating Constructionism* learning theory (suggested by Papert & Harel, 1991) [9], meaningful learning-by-making occurs "in a context where the learner is consciously engaged in constructing a public entity." Since a large disparity was found between the thinking habits and attitudes toward the system development process of beginning students and those of expert software developers [10], it is important that novices should "acquire correct programming habits, suitable for the development

of large complex programs. . . . [so that they will be able to] cope with developing large software systems in the future" [6].

Project-based learning and software development assignments, performed by students in meaningful contexts while applying innovative thinking, may facilitate meaningful learning as well as contribute to making computing and software engineering more appealing. Project development enables students to enhance cognitive and reflective skills, and to encounter real life experience as project developers [8, 11, 12]. Moreover, it encourages students to become creative, innovative [13], and independent learners.

The academic CS community believes that the role of projects in the undergraduate curriculum is of great importance, since beyond being a means for effective learning, it also demonstrates the student's mastery of skills appropriate for professional practice [12, 14]. The open question is, to what extent do high-school projects prepare students to gain "real world" software engineering experience?

Traditional high-school projects enable students to experience software design and development processes, and to acquire a system-based perception. However, the high-school setting has several shortcomings:

1) the teachers are not members of the SE community of practice, and they usually lack practical industrial experience;
2) the school labs are unable to provide infrastructure characteristics of high-tech industry;
3) the students develop individual projects (team-projects are not approved for formal assessment by the Israeli Ministry of Education);
4) usually, the specifications of the product are not provided by a real external client (but by the teacher or the student), and the products usually are not developed rigorously according to the specifications;
5) it is common that the projects of students that belong to the same class resemble too much the projects of their classmates. This phenomenon is direct evidence of strictly adhering to patterns in a negative sense.

Scherz and Pollack (1999) found that high-school instruction in project development is done intuitively by teachers and focuses only on principles of programming and problem solving. "The underlying assumption is that students who know how to 'write a program' will naturally be able to build a computer system. However, it turns out that teachers and students have reported difficulties in project development—mainly in the non-programming stages such as choosing a subject, planning, analysis and evaluation" [15, p. 88]. Moreover, the reality in school environments has shown that teachers have neither the methodology nor the tools to guide projects continuously throughout the year. "The teacher's role in project development is actually to solve the immediate problems and to help the student out of impasses, rather than coaching, advising and supervising through various project stages" [15, p. 89].

Even though the quality of high-school software projects may provide evidence of students' high-level programming skills, and their in-depth investment in the project, the development processes do not resemble actual R&D industrial processes, and the products are rarely applicable to real-world situations [3].

Guiding principles and a related model

The above considerations motivated us to choose the following guiding principles underlying the instructional model of the project development activity:

- Recruiting role-models: One possible solution to this "gap" problem between the school-project setting and the "real" world is that professionals from academia and high-tech industry will take an active part in educating potential newcomers, specifically in mentoring students when developing appropriate projects. Since role-models act as mediators of information, and they inspire others to understand and appreciate the work of today's scientists and engineers, we believe that an appropriate encounter between novices and representatives of the computing community of practice is very important. This kind of interaction of students with leading professionals may motivate the students to pursue their studies further or to pursue a career in the field of computing and software engineering [4]. Hence, we recruit mentors who are representatives of the following streams of the computing community of practice: (a) faculty members of CS/SE academic departments, (b) M.Sc and PhD students, and (c) scientists or engineers in hi-tech industry.
- Revealing (also) "weak" facets of industrial software engineering: As well as introducing students to state-of-the-art computing research and development, it is also important, as a "preventive treatment", to reveal and discuss the "weak" facets of industrial software engineering.

Wieringa (2005) concludes from an extended analysis of research papers that the reason why industry refrains from adopting requirements for engineering methodologies developed by the academia is that most requirements for engineering researchers do not constitute sound, scientific research [16]. This assertion seems to hold for the other sub-fields of software engineering as well. Parnas (1998) points out that computer science alone is insufficient for educating a software engineer [17]. The core-sciences of an engineering discipline are the fields of mathematics and science on which that discipline's research and education are focused. Salzer and Levin (2007) argue that the intrinsic deficiencies of the software industry cannot be attributed to a lack of knowledge in

computer science, or software engineering's traditional core science. Therefore, they suggest that the core-science of software engineering is not only computer science, a pure mathematical discipline, but also psychology [18]. Taking Wieringa's position [16] one step further, they propose that to investigate the industry's software engineering-related deficiencies, researchers should also ask scientific questions in the field of psychology [18].

Unfortunately, by the time our students will join the workforce, this situation may not change very much. Therefore, we decided to prepare them, by at least admitting our mistakes, and making them aware of what they have not been prepared for by adhering to the traditional, computer science curriculum.

Optimal student-mentor matching

Project development activity is designed for those "cream of the crop" students who exhibit the following characteristics: high motivation, creativity, self-learning and inquiry ability, persistence, consistency, and the ability to follow a time table. Accordingly, the selection of students for this stage is based on the following criteria:

1) the teachers' recommendation;
2) the applicant's resume, which should provide information about his CS knowledge, programming experience, knowledge of programming languages, participation in other relevant enrichment programs, and experience in developing software projects;
3) the applicant's ability to persuade us that he is seriously interested in developing the project, and that he is capable of successfully accomplishing the development and can submit a working product.

Student-mentor matching is the key for successful development of a project. To establish optimal matching, we developed the following *employment fair* model: a special meeting is conducted, where, in a plenum session, the mentors present to the students a variety of project subjects for which they can serve as advisors. After the presentation, a face-to-face mentor-student interaction takes place where students are asked to present to mentors their "CV-like" applicant's resume. The process ends when all possible interactions take place. During the interaction, the students ask the mentors questions about the suggested projects and examine whether the topics seem attractive and can be dealt with and whether they want the mentor to guide them. At the same time, the mentors implicitly investigate whether the students are qualified enough to develop the project that they suggested. Next, the students are required to submit a list of projects in order of their preference, and the mentors are asked to choose students according to their assessment. Finally, the managers of the program perform the mentor-student pair matching [3].

Diversity

One goal is to enable students to choose the project's subject from a variety of subjects suggested by the mentors. The possibility of choosing a subject from a color range set may increase students' intrinsic motivation to develop a project, and encourage them to reach their potential. This goal could be achievable by the recruitment of a diverse group of mentors. Another aspect of diversity relates to the mentoring method, student-mentor communication, and the student's style of self-learning. In our program we opted to enable this type of flexibility to facilitate optimal student-mentor interaction and performance.

Profile of graduates

Our aim is for the project development activity to enhance students':

1) skills such as creativity, self-learning, curiosity, inquiry ability, innovation, system-level perception;
2) good habits such as time-management ability, communication skills, and following specifications;
3) professional knowledge. However, a suitable balance between students' freedom and standards should be maintained.

THE SETTING

The students develop comprehensive software projects in a variety of scientific and industrial fields under the apprenticeship-based supervision of professional mentors. The project development process lasts 9–10 months. Examiners nominated by the Ministry of Education evaluate the students' final products; the students' final grades are considered as part of their high-school Matriculation Diploma. The projects are developed according to the following main stages: choosing a problem/subject; analyzing and planning; implementing and testing. During the development process, the student needs to acquire theoretical and technical knowledge, as illustrated in Table 1.

Table 1. Gaining knowledge and proficiency through project development activity

Phase #	Description
Phase 0	Initial acquaintance with the subject; specification of the system.
Phase 1	Studying the theoretical background needed for developing the project.
Phase 2	Analyzing, planning and identifying the main algorithmic ideas.
Phase 3	Acquiring the needed technical knowledge—studying a suitable programming language and a development environment.
Phase 4	Implementing the project (writing and testing the code).

Some of the students actually participate in "real" industrial projects, thus solving "real-world" problems for a real client; others utilize advanced industrial development tools. The role of the mentor is twofold:

1) to provide the student with guidelines and resources for acquiring theoretical and technical knowledge needed for developing the project;
2) to guide and to control the student's progress in various stages of developing the project, for example, checking whether the product addresses the initial specification and requirements; checking whether the student progresses according to a planned time table; assessing the use of design methods, and assessing the quality of the programming.

During that period the students are requested to submit sub-products (e.g., specification documents and a mid-term report) according to a given timetable. At the end, towards the internal and external examination, the students have to submit a working system and a written report that describes the problem and the targets of the system; it must also document the outcomes of each stage of the development process. In addition, the report must include a printout of the systems' documented code. Meetings involving the whole Stage-B group are devoted to software development issues that are presented and discussed by experts. In addition, in these meetings the students report to the group about their progress in developing their projects. Students are also invited to ask for help through a technical-support forum managed by a counselor. Peer to peer interactions and information sharing in the forum is encouraged. The counselor is a CS undergraduate student, a graduate of our program.

The teachers are actively involved in supporting the students throughout the entire development process. The challenge is to create effective cooperation among all four types of participants:

1) the students,
2) the mentors,
3) the academic management team of the enrichment program,
4) the teachers.

PRELIMINARY SCAFFOLDING ACTIVITIES

Aimed at exposing the students to state-of-the-art computing research and development, the program offers a wide-spectrum of topics, some of which are specifically dedicated to software engineering methodologies, and professional norms. For example, some lectures (of Stage-A) are related to:

a) the development of complex systems—model-based development, advanced software development tools, computing in space;
b) professional norms—standards, the importance of testing, and controlled reuse of code;
c) human aspects of software development such as agile programming.

In addition, meetings related to time management and role-playing activities were conducted.

Why we, in the software industry, refrain from following standards

The students in Stage-A of the program attended a lecture entitled "Why we, in the software industry, refrain from following standards" The lecture, using language and terms from the world of high-school students, highlighted differences between computer science, which is basically a pure mathematical discipline, and software engineering, which is an engineering discipline whose core sciences, according to Salzer and Levin (2007), are computer science and psychology [18].

The students were asked to prepare for their projects (in Stage-B of our program, as well as in their further professional life) by paying attention to the differences between what they were doing in the last ten or so years at school and a real industrial project. Taking into consideration that even in school learning there is a difference between regular problem-solving (of algorithmic problems) and project work [15], and, needless to say, an essential difference between school projects and the industrial environment, we discussed several issues, including those presented in Table 2.

In attempting to manage students' expectations from the projects ahead of them, students were told that not only does the school vs. industry difference make them unfit for industry—they are in the very same position as were most of us veterans, when we were "newbies". Therefore, it is suggested that they learn not only from our success stories, but more than that—from our mistakes.

A time-management workshop

(Stage-B) students attended a short time-management workshop at a stage in which they had already experienced some of the project development, had sensed the complexity of the process, and hopefully realized the need for time-management scaffolding tools. Actually, the workshop was conducted in preparation of the mid-term report.

The workshop was aimed at demonstrating the concept of buffer-management, one of the key constituents of Critical Chain Project Management, or CCPM [19]. In the industry, CCPM uses the project buffer to control the uncertain portion of time estimates. The underlying assumptions are that:

1) we cannot tell ahead of time which task's actual time consumption will be above the optimistic, low estimate,
2) when a project uses educated time estimates, then the time overflow will be the equivalent of

Table 2. Differences between school projects and industrial environment

	Industry (Engineering motivated)	**High School, University (Science motivated)**
Who is the client?	The financier	The student, the teacher
Where the needs come from?	The client and other stakeholders	The teacher's assignment, the student's motivation
How are the project and team managed?	The manager's authority, expertise	All are equal, some refrain
Where does the motivation come from?	Job security, advance in job position	Knowledge, grades
What new knowledge is gained?	Expertise, proficiency in familiar fields	New fields
What determines success?	Timely delivery, devotion, quality. (Hardship of task is not a goal.)	Solving riddles, inventing the wheel
Where does the work product go?	Maintenance . . . maintenance . . .	Build and throw away

the overflow caused by about 30% to 50% of the tasks consuming as much as their pessimistic maximum estimates.

Thus, the project's length is estimated as the total of all the optimistic estimates, plus 30% to 50% (depending on the project manager's confidence) of the difference between the two total estimates. This difference is called the project buffer, since it is not assigned to any task; instead, every time a task overflows its optimistic estimate, the project manager reduces the project buffer by the amount of time overflow.

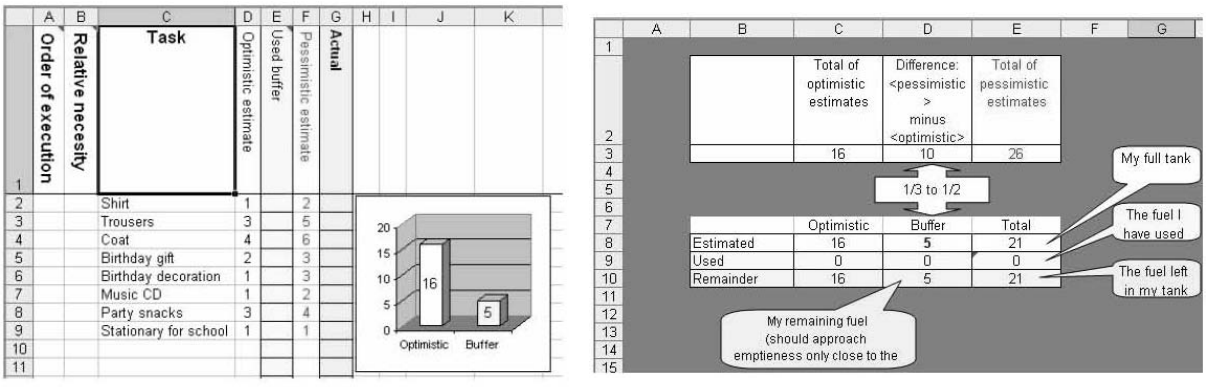
The workshop was trimmed down to the special case of a project having a single "employee" (the student) and having no interaction with other projects. Therefore, of the three buffers of CCPM, only the project buffer was demonstrated.

The workshop consisted of a human play-role simulation game accompanied by a spreadsheet budget-management demonstration. The students received a shopping list that presented for each item an estimated price range (i.e. an optimistic (the lowest) estimate and a pessimistic (the highest) estimate). A short story showed them how important each item would be. The money we gave them (candy-bars representing units of money) was just enough to buy all items on the list, provided that each item would cost no more than the lowest estimate shown on the shopping list. The students' task was to decide on the order they would purchase the items (when paying the actual price) to ensure that once they ran out of money, everything that they did buy was more important than everything they did not.

We used two pre-programmed spreadsheets (see Figure 1) to construct the plan and then to continuously follow-up the budget, in particular, the buffer.

The two spreadsheets were connected through formulae that executed automatic calculations with every input into the first spreadsheet. The input into column A, *Order of execution*, immediately affected the order of the rows. The input into column G, *Actual*, immediately recalculated the *Used buffer* and the bar chart in the first spreadsheet, as well as rows 9 and 10, *Used* and *Remainder* (respectively), in the second spreadsheet. The spreadsheet was intended for demonstration only.

The spreadsheets were brought to the workshop preloaded with the data shown in Figure 1. The bar chart on the right emphasized the two constituents of the budget: the total of the optimistic prices, and the buffer (Figure 1.a). The spreadsheet in Figure 1.b summarized the details in the former spreadsheet. The students were told that the buffer was calculated as 50% of the difference between the totals of the two estimates (high and low) (see Figure 1.b). A corresponding number of candy-bars were positioned in two transparent jars: a



(a) (b)

Fig. 1. Buffer-management spreadsheet. a) Detailed list of tasks b) Summary.

large jar contained the budget's optimistic portion, and a small one—the budget's buffer. The number of candy-bars in each jar was equal to the numbers initially shown on the spreadsheets' bar-chart (16 and 5).

As the students executed each purchase, they "paid" with the candy-bars. For each item, they were allowed to draw from the large jar only the number equal to the item's optimistic price estimate. When the actual price was higher than that, they had to fill in the difference from the smaller jar, thus reducing the buffer. However, when the actual price was smaller than the item's optimistic price-estimate, they had to transfer the difference from the larger to the smaller jar, thus preserving the gain for future use by increasing the buffer. As each actual price was entered into the spreadsheet, the spreadsheet's bar charts automatically exhibited the same behavior.

Three rounds of the game were played. In the first one, we instructed the students to define the order according to reasons not relevant to the real goal. In the second and third round, the students tailored the order to the real goal, and successfully achieved it. During the workshop, we used four different notions to refer to a consumable resource: money (as the price of goods), candy-bars (as the tangible materialization of money), fuel in a fuel tank (see Figure 1.b, to which students could easily relate something that can run out), and, of course, time (as the resource, which was needed to complete a task). We interchangeably referred to all four notions during the workshop, thus giving the students a chance to abstract away the intangible properties, which were common to all four, and that were important to comprehend buffer management. This technique was further augmented by repeating the whole cycle several times from planning through execution and buffer-monitoring.

The students learned to translate uncertainty in price estimates to price ranges, and to compute the project buffer from these ranges. They learned to calculate the increase and decrease in the buffer, depending on the items' actual price compared to the estimated optimistic one, and to monitor the state of the buffer throughout the project's life. In the discussion that followed, the students asked questions, which showed that they were able to readily transfer the make-believe game to the subject matter of time management.

We emphasized to the students that time management (like other aspects of management) should not end with an initial plan, but should continue through the whole life of the project. Near the workshop's end, when students already comprehended the notion of buffer, we discussed the benefits of keeping a tab on the buffer, how to identify when it is deployed too fast, and how to respond to that.

IMPLEMENTATION

The program has been run for the last four years (see Table 3); it serves as an example of a successful partnership among academia, industry, the Ministry of Education, and K–12 educators. Six hundred students around the country participated in enrichment activities; of these, 155 students were selected as candidates for stage B, 86 of whom succeeded in developing high-level software projects.

The subjects of the students' projects are usually related to topics of the enrichment meetings (Stage-A) and actually reflect the mentors' background. Most of the projects mentored by industry representatives have practical characteristics—for example, computerized homes, managing a multimedia-shop, programming a robot, missile detection, and computerized aquarium care. On the other hand, the projects sponsored by the CS faculty and (M.Sc. and Ph.D.) CS students focus on theoretical or research-based subjects such as computerized graphics, image processing, automatic text categorization, modeling-based development of a control system, disassembling and reassembling DNA, simulation of the theory of natural selection, utilizing neural networks for the recognition of characters in a picture, and games based on learning machine theory.

EVALUATION

A long-term formative evaluation of Stage-A of the program has been conducted regarding students' attitudes towards the "different-from-school" style of learning. Pre- and post-questionnaires were administered to all attendees before and after each enrichment workshop, with the aim of identifying the expectations of newcomers, as

Table 3. Participation in the program—Four years of experience

Years		2004–2006	2005–2007	2006–2008	2007–2009
Stage A	# Students	71	140	180	210
	# Schools	9	20	30	30
Stage B	# Candidates	25	50	80	85
	# Graduates*	13	28	45	unknown**

* Students who succeeded to successfully finish their project.
** Stage B of the cycle that will begin in June 2008 and will end in May 2009.

well as the attitudes of graduates. So far, the study findings related to students' attitudes have indicated that the program contributes to developing a culture of learning befitting the dynamic world of industrial computing, thus providing the students with an entry point into the computing community of practice [3].

After a pilot implementation of the first cycle of Stage-B, which was accompanied by an informal field study (project development period: 2005–2006), we conducted a formative evaluation of the project development activity. The goal of our study was to assess students' project management style, their communication with their mentors, and their use of resources. We were also interested in students' reflection on various aspects of project development (such as challenges, difficulties, benefits, etc.). Twenty-two students of the second cycle (project development period: 2006–2007) agreed to answer a reflective questionnaire after they had developed their projects. A further, ongoing investigation is currently being conducted with the third cycle of students, aimed at constantly improving the mentoring model as well as the monitoring of the project development processes.

Students' project management style

One main goal of our study was to assess students' project management style and their communication with the mentors. To this end, we asked the students to retrospectively assess the time they invested in developing their projects (hours per month) and to report the number of meetings they had with their mentors (per month).

The project development process usually takes 9–10 months starting in July (the first month) until April (the 10th and last month) when the final external assessment of the projects is performed by examiners of the Ministry of Education. The following graphical view (Figure 2) illustrates students' assessment of meetings per month and invested time per month (average, N=22). The graph clearly shows that students tended to invest a meaningful amount of time at the beginning of the process, which decreases towards the intermediate period (between the 3rd and the 5th month). A two-step increment in students' investment of time can be observed starting at the end of the 5th month until the beginning of the 9th month (when a final internal assessment takes place). At the 10th month, an additional intensive effort is made to make final changes and improvements until the final external examination. It seems that the requirement to submit a midterm report at the beginning of January (the 7th month of development), along with the requirement of presenting it in a peer-to-peer meeting, resulted in a boost in time-investment maintained by the students until the final examinations.

We examined each of the graphical views of

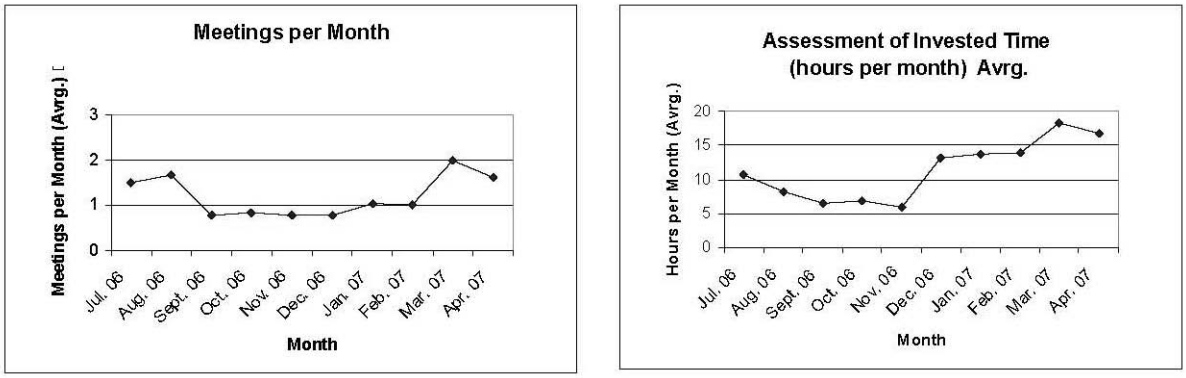


Fig. 2. Meetings and invested time per month during the project development course (Average, N=22)

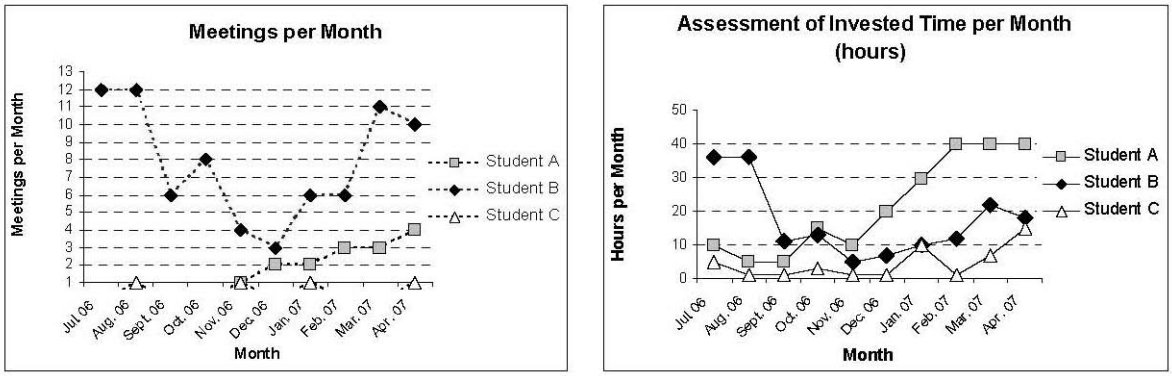


Fig. 3. Three profiles of meetings and time invested per month during the project development course: Student-A, Student-B, and Student-C).

twenty-two study subjects and found a diversity of behaviors. A deeper investigation of three students, characterized by three dissimilar profiles, was performed; these profiles could be considered as reflecting distinct groups of students related to their project management behavior; to obtain a valid categorization, a further investigation, which includes a qualitative study, should be performed. Figure 3 illustrates three profiles of student behavior related to time management and communication with mentors.

Student-A exhibits the behavior of someone who does a large amount of independent work, along with a few meetings with his mentor, which started after a period of self-study and later on continued with a slight increase towards the completion of the project. In contrast, Student-B was coached intensively by his mentor during the whole period; apparently he was unable to achieve autonomous self-investment. Student-C exhibited consistent self-investment and had regular meetings with his mentor every two months; apparently the meetings were devoted to the student's reporting on his progress and his consulting with his mentor, along with the mentor's fine-tuned coaching, control, and support.

Student assessment of resources used for developing projects

In addition to classifying students' time management and frequency of communication with their mentors, we were interested in their use of resources; specifically, we sought to investigate whether there was any relationship among these three dimensions.

Students used a variety of resources during the project development activity [4]:

- Bibliographic Resources—the Web, professional articles, professional books.
- School—The school teacher, school learning (materials and methods).
- (Informal) Human Resources—A classmate the student's age, a family member, an adult acquaintance.
- Mentoring—The mentor.
- Self-Studying—The student.

One main goal of our study relates to student assessment of the resources (N=22). We analyzed assessment of the resources used during phases 1–4 of the project development (see Table 1 above). Importantly, the findings indicated that during the entire development process the students exhibited self-efficacy, since they relied more on themselves than on other resources. Interestingly, during the entire development process, the Web was perceived as the most significant bibliographic resource. Specifically, to achieve adequate acquaintance with the needed theoretical knowledge, self-studying and the Web were perceived as most significant, which may imply that the mentors' guidance motivated the students' self-inquiry and self-study. However, during the problem-solving activities,

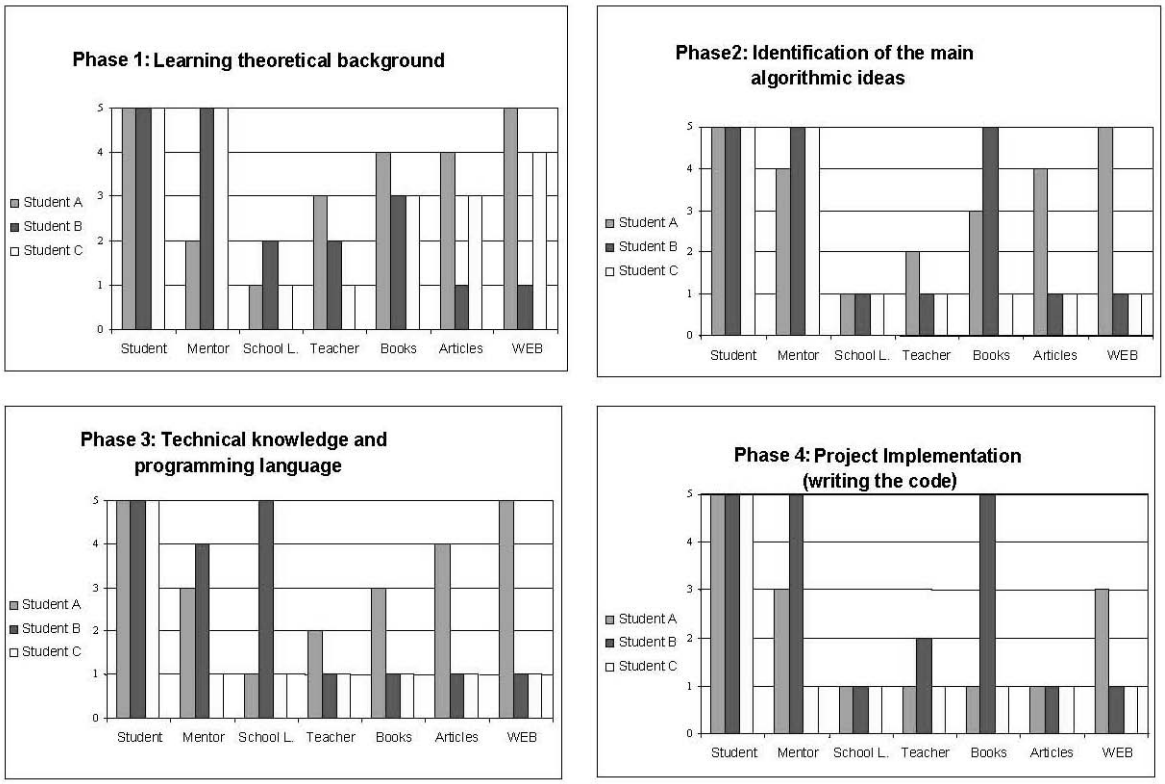


Fig. 4. Three Student assessments of resources used for project development (Student-A, Student-B, and Student-C)

students relied more on their mentors than on bibliographic resources [4].

Figure 4 specifically illustrates the assessment of the resources by Student-A, Student-B, and Student-C in each development phase.

The graphs illustrate that the self-study resource was equally estimated by the three students during all phases (1–4) of the development, receiving the highest grade (5 on a Likert-type scale). For Student-A, the mentor's role was mostly meaningful in phase 2. School-learning was weakly estimated in all four phases; however, the teacher played a significant role during phases 1, 2, and 3 (with respectively decreasing importance). All the bibliographic resources were intensively used during all four phases and were weighed with higher grades than the mentor during phases 1, 2, and 3. The strength of the results lies in the above characterization of Student-A with respect to (a) time-management and (b) frequency of communication with the mentor. Indeed, Student-A could be described as someone who engages in a large amount of independent work and whose success can be attributed to self-investment supported by moderate and consistent mentoring (about two to four times a month), along with the consistent use of bibliographic resources.

For Student-B, the mentor's role received the highest grade (5) almost in three of four phases. In phase 3, which relates to acquisition of technical knowledge and programming language needed for the development of the project, the mentor's role was assigned a slightly lower grade (4). In this phase, Student-B attributed the highest grade to the role played by the school setting (compared with other phases), which implies that he felt confident with the support of his ''natural environment'', particularly at this stage. Actually, according to the mentor's report, Student-B had a weak performance; hence, the human resources were essential for him during all the phases of the project. In contrast, diverse bibliographical resources (e.g. articles and the web) were slightly used (accept for specific books recommended by the mentor), which objectively implies a low level of autonomous and flexible self-studying. Interestingly, however, from the student's point of view, he highly assessed his self-investment during all phases of the project. Though the behavior of Student-B is far from ideal (from our point of view), the positive consequences of these findings are that this specific student displayed high self-satisfaction and high self-esteem. We believe that his positive experience of developing a project, with additional suitable coaching would improve his self-learning ability.

Although Student-C met his mentor only every two or three months during the whole process, he assessed his role by giving it the highest grade in phases 1 and 2, which relate to the learning of theoretical background and planning the solution. Interestingly, his mentor was a biology researcher with no knowledge in computer science; accordingly, he played the role of a ''real'' client and defined the client's software requirements. In contrast, Student-C's teacher did not play a significant role during the entire process. Instead, the student relied more on bibliographic resources and used them intensively during the three first phases. We can conclude that Student-C exhibited characteristics of an autonomous self-learner and a real developer; however, we found that his interaction with his mentor was below par (in terms of meetings per month), and we assume that they both could benefit from more frequent interactions.

Time management of software projects is the art of discerning meaningful objectives and prudently allocating time investment in several competing tasks. The preliminary findings motivated us to improve the mentoring model in order to improve students' time management and use of resources. Further improvements are needed to better monitor the project development process by improving control and assessment of progress toward the project's completion. Such a model should provide suitable tools to diagnose pitfalls related to various aspects of the development process (e.g. weak communication with the mentor, problem-solving difficulties, and technical problems).

Student perception of the project development experience

Upon submitting their final products, the students were asked to retrospectively relate to various aspects of the development process. Several questions were introduced to the students, in an effort to investigate their perception of the project development experience; specifically, we asked them to refer to:

1) the most difficult phase,
2) the most enriching phase,
3) gaining a new perspective,
4) personal benefits.

What was the most difficult phase of the project development?

Among the twenty-one students who answered this question, one student struggled with the initiation of the self-learning process needed to acquire the knowledge required for developing the project (Phase 1). Two students specified the difficulties they encountered when learning new complex theoretical topics such as Fourier analysis for speech recognition or neural networks for information processing (Phase 1). Two students experienced difficulties in starting the main design of the required system (Phase 0, Phase 2). One student reported that he was bored with the project implementation (Phase 4). Another student reported that he had difficulties when writing the user guide of the application that he had developed. Most of the students had difficulties with the algorithmic formalization of theoretical knowledge and with the implementation of algorithmic ideas (Phases 2–4) (in students' words: the ''implementation of the theory in programming''). For example, one student

reported that he struggled with the algorithmic formulation of ''body collision'' theory; another student encountered difficulties in implementing a ''computer player'' when developing a game (Phase 2, Phase 4). Some students encountered difficulties in some technical issues such as retrieving data from a data base, image format translation, or saving data in files (Phase 3, phase 4). Eight students indicated that they struggled in learning a new programming language or paradigms (e.g. Object-Oriented programming, artificial intelligence). In particular, they found it difficult coping in parallel with theoretical and technical knowledge of a new domain, learning a new language, and above all, integrating the various pieces of knowledge to promote the progress of their products.

What was the most enriching part of the development process?

About half of the students stated that the most enriching part of the project was the learning of a theoretical subject or the identification of the main algorithmic ideas needed for solving problems (Phase 1, Phase 2). The remaining students found that the most enriching part was associated with phases connected with acquiring technical knowledge or a programming language (Phase 3), or implementation issues (Phase 4); for example, learning a new programming language, acquaintance with a new paradigm or an unfamiliar working environment, or building hardware (which is considered new for most CS students). One student reported that the process of seeking a solution to fix errors enabled him to reach a better understanding of his own work.

Some students did not refer to specific phases of the development; they instead noted that the most enriching experience was to work together with the mentor. For example, a student wrote in his final report that: ''Working with a mentor who is a professional was for me as a unique experience, with both pleasing and educational aspects. His contribution was a meaningful contribution to the project's achievement.''

The perception of the most enriching part of the project is very individual. Our program is characterized by great diversity related both to the student population, mentors, and the types of projects offered by the mentors (or suggested by the students). This implies a great diversity and the uniqueness of the students' view in relation to his perception of the significance of each phase of the development process. Interestingly enough, usually there has been a direct connection between the sources of difficulties reported by each student and the phase he found most interesting in his project. It seems that overcoming difficulties was perceived as a challenging and enriching experience that provided satisfaction.

Did the project provide you with a new perspective on the school's learning materials?

Most of the students stated that the project development activity enabled them to significantly broaden the knowledge that they acquired at school. One student specifically indicated that the activity increased his awareness of the need to learn the basics at school as a basis for developing a project even with an advanced topic. Some students described the new perspectives in terms of specific knowledge or new comprehension gained, such as improving their understanding of the mechanism underlying running a program, providing creativity in programming, learning new concepts in Object-oriented programming, solving complex problems, or improving achievements in software design. Six students (out of twenty-one) stated that they did not gain any new perspectives but learned totally new domains not connected to the topics learned at school.

What was the personal benefit that you thought the project development provided?

Almost all the students claimed that they acquired new knowledge and professional experience. Only four of them specified that they learned time management. Interestingly, some students specified other meaningful affective benefits from their experience such as being able to manage a project satisfactorily; enhancing self-confidence; sharpening their sense of responsibility; and accomplishing a final, self-made working product satisfactorily.

CONCLUDING REMARKS

Our program, aimed at exposing the students to state-of-the-art computing research and development, provides them with the opportunity to be mentored by representatives of the computing community of practice. We believe that it is imperative that academia and the high-tech industry take an active part and contribute to making the professional domain of computing more attractive, especially in view of occasional high-tech crises and the continual decline of enrollment in computer science and software engineering studies.

Students in our program are given the opportunity to experience various aspects of developing projects in the ''real-world'', such as coping with ''real'' problems in the sense of developing software for a real client instead of doing assignments that reflect the school setting alone (i.e. defined by the teacher, or entirely motivated by the students). Another important opportunity that the students have is to be exposed to the authority and expertise of a professional; to acquire new theoretical and technical knowledge needed for developing the project; to experience planning, designing, and implementing according to professional standards and norms.

For the last four years the program was accompanied by an ongoing evaluation the aim of which was to assess students' attitudes towards the differ-

ent style of learning and towards various aspects of the experience of developing long-term extracurricular projects [3, 4]. Part of the evaluation's findings is presented in this paper. Importantly, these findings relate to the group of students who succeeded in accomplishing their projects. Since these students consist of only ~60% of the candidates for the project development assignment, we felt that it was important to develop diagnostic tools suitable for detecting pitfalls related to various aspects of the development process (e.g. weak communication with the mentor, problems in time-management, problem-solving difficulties, and technical problems). These problems might especially affect highly motivated students who did not acquire enough skills that characterize an independent learner, and might cause them to quit before completing their project.

In-depth analyses of students' time-management profiles and interviews with students, mentors, and teachers motivated us to specifically diagnose further those difficulties encountered by student-mentor interactions. For example, the interviews revealed the following difficulties:

- Gaps between the students' expectations from the mentors (perceived as teachers) and the mentors' expectations from the students (perceived as independent adults self-motivated enough to learn professional skills).
- Mismatch in the student–mentor pairs.
- Communication difficulties (related to the mentor's professional constraints and travels, technical problems in email communication, the student's school–study constraints).
- Weak engagement of mentors/students in face-to-face meetings.
- Failures related to the authority of the mentor (e.g. small age gap, weak tools to enforce authority).
- Pitfalls in the mentors' management strategies that might result in students quitting before completing the assignment.
- Pitfalls in the follow-up of the mentoring process by the management team of the program regarding each mentor–student pair.

The findings motivated us to plan additional qualitative and quantitative assessments, as well as to devote our efforts to improving the mentoring model with the aim of achieving a balance between diversity and maintaining standards. For example, one specific question relates to the freedom students should be given in respect of following specifications [20]. Some scholars (e.g. [13]) claim that students must be given complete freedom to express their creativity, and for that to be realized, they must not be forced to stick with the initial plan of their projects. Other scholars maintain [21] that students must fulfill their projects in an organized, methodological way, which includes following the plan that was set at the beginning of their projects. In Stage-B of our program we will instruct the students and their coaches to allow the plan to be changed at will, up to five months before the scheduled end of the project. Based on Boehm's diagram [22], which indicates that the cost of fixing errors, as the project phases advance, proceeds according to a logarithmic growth pattern, we predict that until this point in time, even dramatic changes in what the student will choose to work on will have only a marginal price tag.

To this end, we decided to perform the following actions:

- Organize more preliminary meetings between the management team, the students, the teachers, and the mentors in order to provide more information about the process and to reach an agreement regarding the expectations of the students, mentors, and teachers.
- Refine the student-mentor matching by involving the teachers in finalizing the matching process.
- Organize continuous meetings throughout the project development activity related to time management and other aspects of the project development process.
- Reinforce the mentors' authority regarding timely delivery and to provide them with appropriate assessment tools.
- Reinforce the teachers' authority in encouraging communication between students and mentors.
- Refine the follow-up of the program's management team after the mentoring process for each mentor–student pair.

Further improvements are needed in the future for better monitoring of the project development process by improving assessments of the progress of the projects toward their completion. More lectures and more scaffolding activities should be conducted. For example, in the buffer-management activity presented above, we taught time-management that fits to the closed-system nature of continuous working hours at a workplace. In contrast, as Figure 3 (Assessment of invested time per month) illustrates, students work on an individual school project only for short bursts. Buffer-management assumes that the main competition for an employee's time is between different tasks. Students have no fixed "working hours" for their homework projects; therefore, the list of competing tasks includes much more than just the project tasks. The buffer-management workshop exposed the students to yet another difference between the academic setting and industry. The students were instructed to apply this technique of time management only to the bursts of more or less continuous work on their projects, but not during the time between.

In addition, scaffolding learning-based tools for project development should be integrated, e.g. the PBL Organizer, which was developed by Polack and Scherz (1999) [15, 21] for gradually supporting both students and teachers (mentors) in project development processes.

REFERENCES

1. P. D. Long and S. C. Ehrmann, *Future of the learning space: Breaking out of the box,* Educause, (2005), pp. 42–58.
2. D. Passig, Taxonomy of IT future thinking skills, In Tailor, H., & Hogenbirk, P. (Eds), *Information and Communication Technologies in Education: The School of the Future*, Kluwer Academic Publishers, Boston, (2001), pp. 152–166.
3. C. Yehezkel and B. Haberman, *Bridging the gap between school computing and the ''real world'',* Lecture Notes in Computer Science, 4226, (2006), pp. 38–47.
4. B. Haberman and C. Yehezkel, Computer science, academia, and industry—An educational program For establishing an entry point to the computing community of practice, *J. Info. Tech. Educ*., **7**, 2008, pp. 81–100.
5. J. Gal-Ezer, C. Beeri, D. Harel, and A. Yehudai, A high-school program in computer science, *Computer*, **28**(10), 1995, pp. 73–80.
6. J. Gal-Ezer and A. Zeldes, Teaching software designing skills. *Computer Science Education,* **10**(1), 2000, pp. 25–38.
7. J. E. Sims-Knight and R. L. Upchurch, *Teaching software design: a new approach to high school computer science,* Annual Meeting of the American Education Research Association, Atlanta, GA, (April 1993).
8. ACM/IEEE Joint Task Force on Computing Curricula, Software Engineering 2004 *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, A Volume of the Computing Curricula Series, August, 2004. Retrieved June 25, 2007, from http://sites.computer.-org/ccse/SE2004Volume.pdf
9. S. Papert, and I. Harel, Costructionism, Ablex Publishing Corporation, (1991).
10. E. Fleury, Students' beliefs about Pascal programming*, J. Educ.Computing Research*, **9**(3), 1993, pp. 355–371.
11. B. Bracken, Progressing from student to professional: the importance and challenges of teaching software engineering, *JCSC*, **19**(2), 2003, pp. 358–368.
12. S. Fincher, M. Petre and M. Clark, Computer Science Project Work Principles and Pragmatics, Springer-Verlag (Eds.), London, (2001).
13. P. J. Denning and A. McGettrick, Reentering computer science, *Communication of the ACM*, **48**(11), 2005, pp. 15–19.
14. M. Holcombe, A. Stratton, S. Fincher, and G. Griffiths, (Eds.), *Projects in the computing curriculum*, Proceedings of the Project 98 Workshop, Springer-Verlag, London, (1998).
15. Z. Scherz, and S. Pollack, *An organizer for project-based learning and instruction in computer science*, Proceedings of ITiCSE'99 Conference, Cracow, Poland, 1999, pp. 88–90.
16. R. J. Wieringa, Requirements researchers: are we really doing research? *Requirements Engineering*, **10**(4), 2005, pp. 304–306.
17. D. L. Parnas, Software engineering programs are not computer science programs, *Annals of Software Engineering*, **6**, 1998, pp. 19–37.
18. H. Salzer, and I. Levin, The core-science of software engineering, Proceedings of the 5th annual European Computing and Philosophy Conference (ECAP) June 21-23, 2007. University of Twente, The Netherlands (2007).
19. E. M. Goldratt, Critical Chain, The North River Press Publishing Corporation, MA, USA (1997).
20. L. B. Sherrell and S. G. Shiva, Will earlier projects plus a disciplined process enforce SE principles throughout the CS curriculum? ICSE, St. Louis, Missouri, USA, (2005), pp. 619–620.
21. S. Pollack, and Z. Scherz, Supporting project development in CS—the effect on intrinsic and extrinsic motivation. Proceedings of the 10th PEG Conference, Tampere, Finland, (2005), pp. 143–148.
22. B. W. Boehm, Software Engineering Economics, Prentice-Hall (1981).

Bruria Haberman received her Ph.D. degree in Science Teaching from the Weizmann Institute of Science. She is currently a senior teacher in the Department of Computer Science in the Holon Institute of Technology, teaching Logic Programming, Data Base Systems and Expert Systems. She leads the Computer Science, Academia & Industry educational program for talented high school students and their teachers in the Davidson Institute of Science Education in the Weizmann Institute of Science. Her research interests focus on computer science educational research, problem-solving utilizing abstract data types and algorithmic patterns, and in-service teacher education.

Cecile Yehezkel received her Ph.D. degree in Science Teaching from the Weizmann Institute of Science. She holds a M.Sc. in Bio-Medical Engineering from Tel Aviv University and a B.Sc. in Electrical Engineering from the Technion, Haifa. She is currently teaching courses related to microcomputers and low level language at the School of Engineering at Bar-Ilan University. She leads the Computer Science, Academia & Industry educational program for talented high school students and their teachers in the Davidson Institute of Science Education in the Weizmann Institute of Science. Her research interests focus on human-computer interaction, modeling and simulation design and evaluation, computer architecture and engineering education.

Hanania Salzer is a Ph.D. student at the Tel-Aviv University, School of Education. For the last 25 years he worked in the software industry (currently at EDS Israel). He earned his M.Sc. in zoology and B.Sc. in biology at the Tel-Aviv University.