Int. J. Engng Ed. Vol. 23, No. 1, pp. 15–23, 2007
Printed in Great Britain.

0949-149X/91 $3.00+0.00

A High-School Programme in Software Engineering*

BRURIA HABERMAN
Computer Science Department, Holon Institute of Technology, and
Department of Science Teaching, The Weizmann Institute of Science,P.O.B 26 Rehovot 76100, Israel.
E-mail: bruria.haberman@weizmann.ac.il

AVI COHEN
Department of Information Science, Bar-Ilan University, Ramat Gan 52900, Israel.
E-mail: Avi@CSIT.org.il

We describe a comprehensive three-year programme in Software Engineering (SE) for high-schools which has been operated in Israel for the last two decades. The aim of the programme is to expose young students to computing, and to motivate them to continue their academic studies in that field. The programme has evolved over the years in accordance with the changes in the discipline of computing. It introduces students to scientific methods, principles of design, implementation of computer systems. Currently it consists of a three-phase modular structure: (a) natural sciences, (b) computer science, (c) advanced specialized topics in computing. During the third year, students are required to develop as a final assignment a comprehensive software project, namely a computer system in a specific, specialized domain.

Keywords: Software engineering, curriculum, project development, system-level perspective, integrative knowledge, software design skills, evaluation.

PROGRAMME IN THE MAKING

EDUCATORS have noted the importance of teaching software designing skills to high-school computer science students [e.g. 13, 15, 28]. During the last two decades, a programme in Software Engineering (SE) especially designed for high-school level has been in operation in Israel. Since 'computing has changed dramatically over time in ways that have a profound effect on curriculum design and pedagogy' [1, p. 1], the programme presented here has evolved, similarly to others, influenced along the way by changes in computing and the development of computer technology.

High-school education in Israel

The education system in Israel is basically centralized and the studies in high-schools have a uniform basic structure. The schools are classified according to one of three tracks: general (academic, theoretical), technological (vocational) and agricultural. Students attend classes 32–36 hours weekly and may study up to 15 subjects at a time. Subjects are taught in instructional units of 90 hours. Upon completion of Grade 12, students may take the matriculation exams. The requirements for obtaining the current Matriculation Certificate are 20 units in a range of 20–25; a minimum of 15 units in compulsory subjects are required. In addition, students who take the technological track have to be examined in 7–15 units in technological subjects in order to get a Technological Certificate in addition to the Matriculation Certificate.

One of the outcomes of the education system's centralization is that programmes of all the technological sub-tracks have always had a common modular structure, which evolved in accordance with the development of technology. Currently, all technological programmes consist of the following 3-phase general structure:

(a) an elective topic in natural sciences/introduction to technology sciences;
(b) a basic mandatory specialized topic;
(c) an elective advanced specialized topic.

In spite of its uniform structure, the programme allows flexibility to some extent. It includes, besides mandatory units, elective units, meaning that teachers in each school are entitled to prepare a programme according to their professional preference that suits their students' background, abilities, interests and needs.

The programme's evolution

Originally, The Software Engineering programme for high-schools was uniform and consisted of three mandatory topics: Introduction to CS, Management Systems Analysis, Design & Programmng (D&P) of Information Management Systems. COBOL was chosen as a suitable programming language to teach the implementa-

* Accepted 18 October 2006.

tion of systems of that type. Although the main learning objectives of the software were to teach the basics of the development and implementation of information management systems (e.g. system modelling and analysis, data organization, and system's life cycle) it mostly emphasized technical aspects of programming at the expense of discussing the basic concepts and principles of the science of computing.

In the early 1990s, as a result of the computing field's expansion, a special committee selected by the Ministry of Education recommended modifying and updating the curriculum. The committee stated that advanced specialized topics should be elective, and suggested syllabuses for three additional specialized topics: Computer Graphics, Operating Systems and Expert Systems [4]. The committee also recommended that students should learn natural sciences as additional enriching topics. However, that recommendation was avoided by many principals who operated the programme in their schools mostly because of:

(a) pedagogical considerations—they were afraid that learning an additional advanced scientific topic might be a burden for the students and might cause a cognitive overload;
(b) budget constraints—based on local institutional priorities and specific problems related to management.

A significant change aimed at establishing a scientific background for the program and at promoting students' understanding of the scientific method was initiated by the first author, who served (1996–1998) as the head of the computer science and information technology section of the Ministry of Education. To promote the studies of natural sciences by SE students, it was decided to change previously mandatory management systems analysis topic to be an elective topic together with the natural sciences topics (physics, chemistry, biology). Nevertheless, a 90-hour learning unit, which was part of the management systems analysis topic and concentrated on introduction to information systems implemented in a (computing) application builder (e.g. ACCESS), was declared mandatory for all SE students. Consequently, since then, most of the students who had specialized in information management systems chose to learn the original syllabus of the management systems analysis topic, whereas most of the students who specialized in computer graphics, operating systems and expert systems instead chose to study natural sciences. Since the science-oriented high-school studies have always been highly appreciated by the Israeli academic system, this change markedly promoted the status of the SE high-school studies. As a result, there was a significant ongoing growth in the number of schools that decided to operate the software engineering programme and in the number of students who chose to participate in it (Fig. 1).

The next change in the programme occurred in the early 2000s as part of substantial organizational changes that were made in the technological track. The Ministry of Education decided to reduce the gap between the general (academic, theoretical) track and the technological track to enable student movement between tracks. The underlying objective was to motivate science-oriented students in the academic track to specialize in advanced technology topics. The need for an adequate scientific background for taking the technological track was noted, and all the technological sub-track programmes were reorganized in a common modular science-based structure, previously described. Specifically, changes were made in the software engineering programme: the management systems analysis elective topic and the mandatory introduction to the information systems learning unit were removed and the programme consisted of the following components:

(a) an elective topic in natural sciences/introduction to technology sciences;
(b) computer science;
(c) an elective advanced specialized topic in computer science.

As a result, the programme was now based on scientific foundations and can be viewed as an

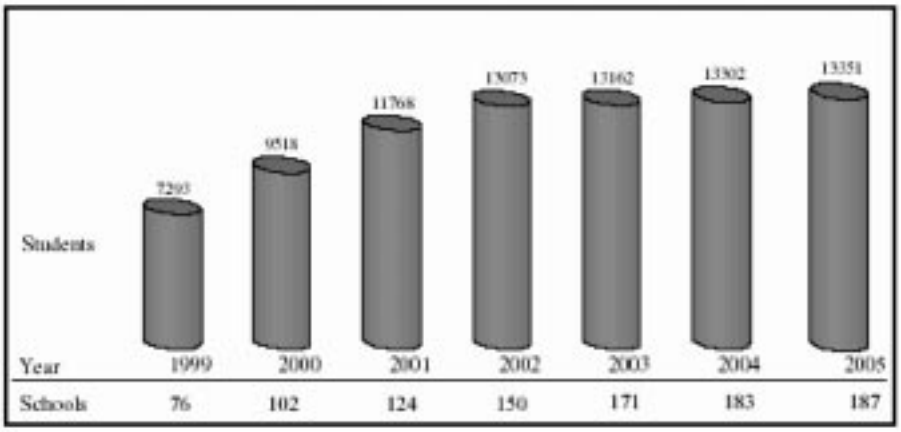

Fig. 1. Number of schools/students attending the SE programme (Grade 10–12)

extension of the computer science programme described in [13, 14]. In addition, the second author, who currently serves as the head of the computer science and information technology section in the Ministry of Education, decided to extend the programme with new elective specialized topics, the aim being to reflect and incorporate the latest developments in computing technology: *XML Web Services* and *D&P of Network Systems*.

Currently, the target population of the software engineering programme consists of:

(1) students from the technological track who decided in advance to specialize in computing;
(2) students from the general-academic track who chose to learn computer science as an elective topic and became interested in expanding their knowledge of the subject.

Further studies

The SE graduate students may continue in a special study track: a one-year programme towards achieving a SE technician diploma, or a two-year programme towards a SE practical engineer diploma. Graduates of the two-year continuation programme may get accreditation in various academic computing studies, depending on their achievements (e.g. B.Sc. in computer science or software engineering).

DESCRIPTION OF THE PROGRAMME

The Joint Task Force on Computing Curricula 2001 stated the following guidelines:

> Computer science, after all, is an ever-expanding field that includes many activities beyond programming. Courses that emphasize only this one aspect fail to let students experience the many other areas and styles of thought that are part of computer science as a whole [1, p. 35].
>
> Because of the importance of computer systems and the wide applicability of computer-based skills, introductory computer science experience should certainly expose students to the design, construction, and application of computer systems and offer them training in skills that have demonstrated utility. [1, p. 26].

The software engineering programme for high-schools presented here was developed following a similar pedagogical approach.

Objectives and underlying principles

The main goal of the programme is to open a window for young students into computing, and to motivate them to continue their academic studies in this field. The programme was not designed to train students to become software professionals; instead, its aim is to expose the students to a fundamental scientific domain whose principles are characteristic of algorithmic thinking as well as system-level perception. It introduces the students to a sort of 'high-level scientific language' for algorithmic problem solving, knowledge representation and formalization of processes. Specifically, its aim is to:

(a) enhance students' problem-solving abilities and reasoning skills;
(b) promote their creativity;
(c) help them gain a system-level perspective and a basic understanding of larger systems and their organization principles;
(d) enable them to construct an integrative and generative knowledge of a variety of computer science topics.

Since educators found a large disparity between the thinking habits and attitudes toward system development processes of young students and those of expert software developers [e.g. 12], it is important that students (even if not trained specifically to become software professionals) should 'acquire correct programming habits, suitable for the development of large complex programmes. . . . [so that they will be able to] cope with developing large software systems in the future' [15, p. 26].

The following underlying principles of the CC2001 curriculum reflect the objectives and the pedagogical approach of the presented programme:

- System-level perspective—Graduates must develop a high-level understanding of systems as a whole: the structure of computer systems and the processes involved in their construction and analysis.
- Understanding the interplay between theory and practice—Graduates must understand not only the theoretical underpinnings of the discipline but also how that theory influences practice. Accordingly, conceptual and experimental issues should be interwoven throughout the programme [13].
- Familiarity with common principles—Graduates must recognize recurring concepts and principles such as recurring themes, e.g. abstraction, complexity, modularity and reusability.
- Integrative knowledge—To ensure that graduates can successfully apply the knowledge they have gained throughout the programme, they must be involved in at least one substantial software project.
- Capabilities and skills—Graduates should develop a wide range of cognitive capabilities and practical skills, some of them related to computer science and others of a general nature, applicable in many other contexts as well.

Components of the programme

The three-phase programme is modular and flexible and can easily be adapted to various student populations (e.g. diverse classes and individual students in heterogeneous classes). Phases A and B consist of 3-unit modules (270 hours) or 5-unit modules (450 hours) each of which is composed of compulsory and elective 90-hour

instructional units. Phase C is a 5-unit complete module of compulsory units. Its flexibility is achieved by final projects which students may develop in different levels (1-unit, 3-units, or 5-units). The flexibility of the programme allows students to take matriculation exams at different levels that are suited to their learning abilities. To address successfully the requirements of the SE track, students must pass three matriculation exams (one of each phase) consisting of 7–15 units.

Phase A—natural sciences

According to the recommendations of Computing Curricula 2001 [1] a successful computer science graduate needs skills beyond the technical ones found in the CS body of knowledge. For example:

- computer science students must have a certain level of mathematical sophistication, familiarity with the methods of science . . .
- Students must develop an understanding of the scientific method and experience this mode of inquiry . . .
- Students may acquire their scientific perspective in a variety of domains, depending on programme objectives and their area of interest [1, pp. 40–41].

This approach is reflected in the SE programme presented here, since students participating in the programme are obliged to study 3–5 units of mathematics and at least one advanced scientific subject (e.g. physics, chemistry, biology) or a basic subject—introduction to technology sciences, following a programme that was specially designed for 'non-scientifically oriented' students. Actually, most SE students choose to study mathematics at advanced level (4–5 units).

Phase B—computer science

During the last decade a new programme in computer science has been taught in Israeli high-schools. The programme emphasizes the foundations of algorithmic thinking, and introduces CS concepts and problem-solving methods independently of specific computers and programming languages, along with the practical implementation of those concepts and methods in programming languages. The 5-unit (450 hours) programme is modular and includes two mandatory modules: Fundamentals of Computer Science (2-units; 180 hours) and Software Design (1-unit; 90 hours), and two elective modules: Second Paradigm/Applications (1-unit; 90 hours) and Theory (1-unit; 90 hours). The underlying principles and pedagogical framework of the programme are presented in [13]; the curriculum and course syllabuses are presented in [14].

The advanced study unit of the curriculum, Software Design, is actually an introductory course in software engineering. It aims at taking the students beyond stand-alone algorithms, and introduces them to various aspects of software systems design. One important goal of the unit is to demonstrate 'full integration of the conceptual material and the actual hands-on experience in designing and constructing a real system' [15]. After studying this unit the students are better prepared to proceed to a higher level of system design and development.

Phase C—Specialization in an advanced topic:

Phase C is designed to enable students to achieve expertise in an advanced computer science topic. The specialization phase is called: Design & Programming (D&P) of Software Systems, since it is designed to teach, in addition to theoretical principles, design methods and implementation tools that are suitable for the specific advanced topic. Currently the programme suggests six alternative topics: Information Management Systems, Computer Graphics, Operating Systems, Expert Systems, Web Services and Network Systems.

Students study the advanced topic for 450 hours during the two last years of high-school. Theoretical principles and practical experimental issues are introduced and practiced in the laboratory. The studies include the learning of a programming language/environment that is suitable for implementing on-topic theoretical material. During the third year, students are required to develop as a final assignment a comprehensive software project.

DEVELOPING SOFTWARE DESIGNING SKILLS

The Software Engineering 2004 Curriculum states that incorporating real-world elements (e.g. case studies, project-based courses and capstone courses) into the curriculum is necessary to enable effective learning of software engineering skills and concepts [3]. Specifically, capstone projects have been recognized as an essential part of SE education [7, 10, 22].

The academic CS community believes that the role of projects in the curriculum is of great importance, since it is a means for effective learning, and also demonstrates the student's mastery of skills appropriate to professional practice [11, 22, 28]. Project development enables students to construct knowledge and to enhance cognitive and reflective skills; it also encourages the student to become a creative and independent learner. In addition, it enables students to encounter real life experience as a project developer [9, 15, 25].

Software design skills and problem solving abilities are gradually developed during the studies of the presented programme:

(1) The Second Paradigm/Applications module of the CS programme (phase B) requires a mini-project in which the student has to utilize specific knowledge acquired studying this module.

(2) When studying the Software Design module of the CS programme, students solve problems that focus on various aspects of design.
(3) During the last year of the specialization studies (phase C) students develop a final project—a software system typical of the learned topic—where they apply, in addition to the design methods and implementation tools that are suitable for an advanced topic, the integrated knowledge that they have acquired during their three years of study.

The projects are developed in the following stages:

(1) choosing a problem;
(2) analyzing and planning;
(3) programming and testing.

In the end, the students have to submit a working system and a written report that describes the problem and documents the outcomes of each stage of the development process; it must include a documented code. The role of the teacher is to guide and to control the students' progress in various stages of developing the project, for example:

(1) checking if the product addresses the initial specification and requirements;
(2) checking if the student progresses according to a planned time table;
(3) assessing the use of design methods, and assessing the quality of the programming [20, 24, 25, 29].

In order to support project-based learning and the instruction of CS/SE, didactic approaches and appropriate learning materials were developed [13, 21, 24, 27]. Workshops for in-service teachers were conducted to discuss pedagogical aspects of project-based learning, such as students' difficulties, project development, and assessment issues [26].

ASSESSMENT

The software engineering programme for high-schools has been operating in Israel for the last two decades. Figure 1 illustrates how the programme has been disseminated in the last few years. Here we will refer to the following aspects of the programme's implementation:

(a) advantages and disadvantages of the programme;
(b) its outcomes in terms of the students' acquired knowledge.

The assessment of the students' knowledge has been formally conducted by the Ministry of Education in the form of matriculation exams. In addition, there is also a large body of research that has been conducted by education researchers aimed at evaluating learning materials and their influence on students' performance, as well as inquiring about the students' conceptual knowledge [6, 8, 15, 16, 17, 18, 19, 20, 21, 24, 25, 27, 29].

Formal assessment of students' performance

According to instructions of the Ministry of Education in Israel, the formal assessment of the students' performance is based on a combination of traditional (i.e. written exams) and alternative evaluation (i.e. project assignments).

Students' achievements in written exams

Generally, the aim of the written matriculation exams in computer science is to assess the students' problem-solving performance. It includes two types of open questions that require students to provide solutions to given problems or to analyze given solutions. Table 1 illustrates the students' achievements in written matriculation exams in the last three years (2003–2005) in the basic module (fundamentals of computer science) and in the advanced module (software design and theoretical subject). The distribution of the grades for each year is slightly skewed right with a slightly higher average grade in the basic exam than in the corresponding average grade in the advanced exam. The findings indicate that the students acquired the desired knowledge and performed satisfactorily in the problem-solving assignments.

Software design and project development

According to the Ministry of Education in Israel, students who participate in the programme are required to develop individual projects in some learning modules instead of a traditional matriculation exam:

Table 1. Students' achievements in written matriculation exams in computer science

	2003		2004		2005	
Module	Basic	Advanced	Basic	Advanced	Basic	Advanced
No. of Students	12107	8418	11220	7297	10781	6679
Failed (Grade: 0–54)	15%	18%	11%	17%	9%	20%
High Achievers (Grade: 85–100)	40%	27%	52%	31%	55%	39%
Average	75	71	79	72	84	72
Median	80	74	85	76	93	89
STDV	20.1	19.1	19.1	19.4	20.3	23.2

(a) a mini-project in which the student has to utilize the specific knowledge acquired studying the Second Paradigm/Applications module of the computer science programme;
(b) a final comprehensive software project in the learned specialized domain. External examiners evaluate the projects using analytical rubrics especially designed for each specialized topic [5]. For example, the rubric for evaluating projects in the *Web Services* specialization domain is as in the box following:

Rubric for evaluating projects in the *Web Services* specialization domain

- **Project portfolio 30/100 points**
  - Readability of project files: 2 pt.
  - Description of the system and its purposes: 8 pt.
  - Description of database, modules, relations and data flow: 8 pt.
  - Site map and description: 6 pt.
  - Users guide: 6 pt.
- **Programming 55/100 points**
  - Readability of software: 3 pt.
  - Modularity, classes and structure: 5 pt.
  - SQL queries: 10 pt.
  - Using ASP.NET: 10 pt.
  - Using Web Services: 10 pt.
  - Interfaces and their computability to system demands: 9 pt.
  - Visualization and human engineering: 5 pt.
  - Originality, creativity, and special sophistication: 3 pt.
- **Demonstration 15/100 points**
  - At examination time the project must work, otherwise the student will FAIL
  - The student must be familiar with every aspect of the project.

The study findings indicated that students are highly motivated to do this kind of assignment and usually prefer to develop projects rather than to complete a written exam [20, 25].

The average grades of the mini-projects as well as the final projects are usually very high compared with the average grades of the written matriculation exams. For example, in 2005 the average grades were 94.8 for the mini-projects and 92.9 for the final projects—much higher than the corresponding average grades of the written exams (basic modules—84; advanced modules—72). The differences in the achievements in the traditional exam vs. the projects are due to the different characteristics of the traditional and the alternative evaluation:

(a) the projects are gradually developed in a non-stressed environment (mostly at home) and under the mentoring of the teachers;
(b) students who attend a traditional written exam cannot consider dropping the course if they do not pass the exam. In contrast, students who fail, for any reason, to produce a final working project may decide to drop the course and therefore do not take the exam.

Academic assessment of students' conceptual knowledge

Qualitative research studies have been conducted aimed at identifying students' difficulties and assessing their CS conceptual knowledge. Students' perceptions of basic computing concepts (e.g. variables [19], and correctness [8]) as well as advanced concepts were investigated (e.g. recursion [18], abstraction [15, 21, 29], efficiency [16, 17] and non-determinism [6]). The study findings served as the basis for didactic recommendations and curricular enhancements. For example, Haberman and Scherz (2002) investigated students' conception of abstract data types (ADTs). The findings indicated that the students demonstrated an integrative knowledge of ADT boxes as programming tools, and employed unique autonomous problem-solving strategies when using ADTs in programming [21]. Ragonis, Shapiro, Ben-Ari and Scherz (1998) evaluated students' conceptual knowledge of expert systems (one specialized topic of the SE programme). Evaluation of the course showed that it was successfully implemented and found suitable for the intended student population. The students' achievements were high, and the main concepts were successfully acquired [27].

Qualitative research studies have also been conducted aimed at (a) evaluating the software designing skills developed by students, and (b) identifying the students' project development strategies. For example, Gal-Ezer and Zeldes (2000) found that high-school students who studied the software design module exhibited difficulties in designing general top-down solutions for a given problem; instead, they preferred to deal with specific examples. The students, however, were able to reuse general structures to distinguish complex tools from basic tools [15]. Scherz and Haberman (2005) found that students who used problem-solving organizing tools in developing their projects were more likely to use abstract data types that resulted in a structured and well-organized development process [29]. Pollack and Scherz (2003) investigated the influence of supportive learning materials on high-school students' motivation, performance and final products regarding the projects in computer science that they developed. The study findings indicated that students who tended to perceive projects as a school activity were mainly motivated by outside rewards such as the projects' external assessment. Moreover, students who did not use supporting materials for project-based learning tended to modify the original problem according to their ability to develop a proper project [24, 25].

General evaluation of the programme

The Computing Curricula 2004 Overview Report discusses the similarities and differences of CS and SE academic curricula:

> Both CS and SE curricula begin by requiring a foundation in programming fundamentals basic CS theory. They diverge in what they focus on beyond those core elements. CS programmes tend to keep the core small and then expect students to specialize selectively in one or two of areas of CS concentration (such as systems, networking, database, artificial intelligence, theory, etc.). In contrast, SE programmes generally provide less freedom of choice about advanced computer science topics, and instead expect students to focus on a range of topics that are essential to the SE agenda (problem modelling and analysis, software design, software verification and validation, software quality, software process, software management, and so on [2, p. 38].

According to this distinction, the programme presented here has the characteristics of an academic CS programme since it is composed of a fundamental core and offers the students the possibility of specializing in one specific advanced CS topic. Moreover, the SE programme for high school does not include formal topics that are essential for the academic SE programmes (e.g. software verification and validation, software quality).

We believe that the programme manages to expose high-school students to the field of computing and enables them to experience software design and development processes. However, the settings of the project development assignment have several shortcomings:

- Usually the projects' performances provide evidence of students' high programming skills and their in-depth investment in developing the project; however, the development processes do not resemble actual research and development industrial processes, and the products are rarely applicable to real-world situations.
- Usually, the specifications of the product are not provided by a real external client.
- The teachers are not members of the SE community of practice, and they usually lack practical industrial experience.
- The school labs are far too inadequate to provide the infrastructures characteristic of high-tech industry.
- The students develop individual projects (team-projects are not approved for formal external evaluation by the Ministry of Education). However, as long as this policy continues, teachers may apply alternative solutions to initiate group-work. For example, they may encourage students to perform peer-assessment of their intermediate products and the development of the final projects. Moreover, the teachers may incorporate project-based classes into their teaching [12, 13].

CONCLUDING REMARKS

In this paper we presented a three-year software engineering programme for high-school students. The main goal of the programme is to open a window of opportunity for young students in computing, and to motivate them to continue their academic studies in this discipline.

The study findings indicated that the programme succeeds in exposing the students to a fundamental scientific domain whose principles are characteristic of algorithmic thinking as well as system-level perception. Moreover, the final project assignment, which is part of the presented programme, enables high-school students to experience software design and the processes of development, as well as to acquire a system-based perception of software engineering.

However, although the programme was not designed to train students to become software professionals, it is recommended that the students be exposed, as part of their studies, to 'real-world' research and industrial development processes that are related to the software engineering projects of the high-tech industries. This can be partially achieved, for example, by informal enrichment meetings in which CS/SE scientists and practitioners will give appropriate lectures to the students [30]. In addition, 'visiting the industry' tours should be encouraged.

Moreover, we believe that it is most important that the high-tech industry will take an active part in educating potential newcomers, and will contribute to making the software engineering professional domain more attractive, especially in view of the last high-tech crises.

REFERENCES

1. *ACM/IEEE Joint Task Force on Computing Curricula*, Final Report, December, 2001.
2. *ACM/IEEE Joint Task Force on Computing Curricula*, Overview Report, June, 2004.
3. *ACM/IEEE Joint Task Force on Computing Curricula, Software Engineering 2004* Curriculum Guidelines for Undergraduate Degree Programmes in Software Engineering, A Volume of the Computing Curricula Series, August, 2004.
4. The Ministry of Education, Israel, *A high-school Information Technology programme,* (1997). (in Hebrew)
5. The Ministry of Education, Israel, *A high-school Software Engineering programme*, (2004). Available: http://csit.org.il (in Hebrew)

6. M. Armoni and J. Gal-Ezer, Non-determinism in CS high school curricula. *Proceedings of the FIE 2003 Conference*, Boulder, CO, F2C–18—F2C–23 (2003).
7. L. Adams, A. Goold, K. Lynch, M. Daniels, O. Hazzan and I. Newman, Challenges in teaching capstone courses. *Proceedings of ITiCSE'03*, Thessaloniki, Greece, 219–220, (2003).
8. Y. Ben-David Kolikant, Students' alternative standards for correctness. *Proceedings of the First International Computing Education Research Workshop*, University of Washington, Seattle, WA, 37–43 (2005).
9. B. Bracken, Progressing from student to professional: the importance and challenges of teaching software engineering. *JCSC*, 19(2), 2003, pp. 358–368.
10. A. T. Chamillard and K. A. Braun, The software engineering capstone: structure and tradeoffs. Proceedings *of SIGCSE'02*, Covington, Kentucky, USA, 227–231, (2003).
11. S. Fincher, M. Petre and M.Clark, (Eds.), *Computer Science Project Work Principles and Pragmatics*, Springer-Verlag, London, (2001).
12. A. E. Fluery, Students' beliefs about Pascal programming. *Journal of Educational Computing Research*, 9(3), 1993, pp. 355–371.
13. J. Gal-Ezer, C. Beeri, D. Harel and A. Yehudai, A high-school programme in computer science. *Computer*, 28(10), 1995, pp. 73–80.
14. J. Gal-Ezer, and D. Harel, Curriculum and course syllabi for high school CS programme. *Computer Science Education*, 9(2), 1999, pp. 114–147.
15. J. Gal-Ezer and A. Zeldes, Teaching software designing skills. *Computer Science Education*, 10(1), 2000, pp. 25–38.
16. J. Gal-Ezer and E. Zur, The concept of 'algorithm efficiency' in the high school CS curriculum. *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*, Boston, MA, T2C1–T2C6 (2002).
17. D. Ginat, Efficiency of algorithms for programmeming beginners. *Proceedings of SIGCSE'96*. Philadelphia, PA, 256–260 (1996).
18. B. Haberman and H. Averbuch, The case of base cases: Why are they so difficult to recognize? Student difficulties with recursion. *Proceedings of ITiCSE'02*, Aarhus, Denmark, 84–88 (2002).
19. B. Haberman and Y. Ben-David Kolikant, Activating 'black boxes' instead of opening 'zippers'—A method of teaching novices basic CS concepts. *Proceedings of ITiCSE'01*, Dublin, Ireland, 41–44 (2001).
20. B. Haberman and Z. Scherz, Abstract data types as tools for project development- High school students' views. *Journal of Computer Science Education online*, January 2003. Available: http://iste.org/sigcs/community/jcseonline/
21. B. Haberman and Z. Scherz, Evolving boxes as flexible tools for teaching high-school students declarative and procedural aspects of logic programming. In Mittermeir, R. (ed.), From Computer Literacy to Informatics Fundamentals, International Conference on Informatics in Secondary Schools-Evolution and Perspectives. *Lecture Notes in Computer Science*, 3422, 156–165 (2005).
22. 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).
23. D. N. Perkins, Smart schools-from training memories to training minds, New York: The Free Press, (2000).
24. S. Pollack and Z. Scherz, Organization of learning by computer science projects. *Proceedings of The 10th PEG conference*, Tampere, Finland, 143–148, (2001).
25. S. Pollack and Z. Scherz, Supporting project development in CS—the effect on intrinsic and extrinsic motivation. *Journal of Computer Assisted Learning*, special issue: Present into future: ICT for learning and teaching in the 21st century (accepted for publication).
26. N. Ragonis and B. Haberman, A multi-level distance learning-based course for high school computer science leading-teacher. *Proceedings of ITiCSE'03*, Thessaloniki, Greece, 224 (2003).
27. N. Ragonis, E. Shapiro, M. Ben-Ari, and Z. Scherz, Development, implementation and evaluation of a course in expert systems for high-school students. *Proceedings of ITiCSE'98*, Dublin, Ireland, 300 (1998).
28. 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., (1993).
29. Z. Scherz and B. Haberman, Mini-projects development in computer science- Students' use of organization tools. *Informatics in Education*, 4, 2005, pp. 307–319.
30. C. Yehezkel and B. Haberman, 'Computer science, academia, and industry' educational project. *Proceedings of ITiCSE'05*, Monte de Caparica, Portugal, 364 (2005).

Bruria Haberman received her Ph.D. degree in Science Teaching from the Weizmann Institute of Science in 1999. Presently, she is an instructor in the Department of Computer Science at Holon Institute of Technology. She is also a member of the computer science team in the Department of Science Teaching in the Weizmann Institute of Science, and a leading member of Machshava—the Israeli National Centre for high school computer science teachers. She has developed learning materials for high school level in the areas of logic programming, artificial intelligence and algorithmic patterns. She has developed academic programmes for undergraduate level in computer science. Her primary research interests are computer science educational research— student conceptualization of computer science, as well as in-service teacher education and distance learning.

Avi Cohen has a Ph.D. from Anglia Polytechnic University at the Ultarlab UK. Since 2001 he has specialized in instructional technology and Internet studies. Currently, he is the director and superintendent-in-chief of Computer Science and Information Technology at the Ministry of Education in Israel. In this capacity he is responsible for Computer Science studies, curriculum, matriculation exams and authorization of CS teachers. For the last 23 years he has been a high-school teacher in Computer Science and Electronics Engineering. He wrote three textbooks for high-school students (programming—CS0, 1, 2). Recently he has been teaching in the Department of Information Science at Bar-Ilan University. His primary research interests are XML Web Services and Web development, as well as Internet stateless programming.