A High School Program in Computer Science

Judith Gal-Ezer
Open University of Israel

Catriel Beeri
Hebrew University

David Harel
Weizmann Institute of Science

Amiram Yehudai
Tel-Aviv University

A team of researchers and educators introduces a computer science curriculum into Israeli high schools. This curriculum combines conceptual and practical issues in a zipper-like fashion.

Computers are undoubtedly the most important invention of the twentieth century, having dramatically and irrevocably changed the way we live and work—mostly for the better. One implication is that educated people should be computer literate. This, in turn, creates the need for introducing computers into high school curricula.

Computing as a scientific discipline, now called computer science (CS), predates the invention of computers. The first decades of this century saw a crystallization of the discipline's fundamental concepts accompanied by surprising discoveries regarding the inherent limitations of computation. More recently, fueled by the invention of computers and their widespread use, the study of computing has bloomed, and CS is now recognized as an autonomous scientific discipline. Its scope includes the study and analysis of algorithmic processes, their power and limitations (sometimes called *algorithmics*[1,2]), and computing systems' design and implementation. CS concepts are influencing work in other scientific disciplines, and CS central notions and thought styles are becoming widely assimilated. Clearly, a modern high school curriculum should reflect this growing importance.

Most high school activity in computing has centered around courses in computer literacy or on the use of computers as teaching aids in other disciplines. When computing has been taught in high school as an autonomous subject, the emphasis has been most often on the technicalities of a programming language; at best, students learned to "code." However, coding is only one side of the coin. Our interest is far more foundational and addresses the need to establish CS as an accepted scientific subject on the high school level of education, to be taught on a par with other sciences, such as biology, physics, and chemistry.

As with any scientific subject, a key criterion for defining the core of the subject is longevity. To a large extent, computer technology changes far more rapidly than the basic ideas of the *science* of computing, which center on the notion of an algorithm and its use in computing systems. Based on the sixty years or so since the pioneering days of CS, these ideas have lasting and fundamental value. Thus, although a proposed high school program should enhance a student's ability to exploit computers beneficially, its backbone must be based on science. The program should provide insight, knowledge, and skills independent of specific computers and programming languages. Moreover, since high school is the only time many students are exposed to CS, a good curriculum must also aim at breadth and versatility.

This article describes such a curriculum. Its emphasis is on the basics of algorithmics, and it teaches programming as a way to get a computer to execute an algorithm. It has been proposed by a committee formed in 1990 by the Israel Ministry of Education.

0018-9162/95/$4.00 © 1995 IEEE

BACKGROUND

CS is a relatively young discipline, and CS education is even younger. But it is not only our field's tender age that causes problems for the educator. The nature of the field itself is a factor as well. On the one hand, with its formal methods and abstract thinking, CS resembles mathematics, but on the other hand, it is very much an engineering discipline, requiring concrete, down-to-earth skills. This has caused lengthy, often tedious controversies.[1,3-5] For example, what *is* CS? What is its relationship with other fields? How do its subfields relate to one another? What characterizes a well-educated computer scientist?

Considerable activity has surrounded CS curricula at all levels. Particularly relevant is the high school curriculum designed by an ACM special task force, whose report contains references to some of this activity.[6] Our work has a similar basic philosophy. However, the goals and scope of the two programs are quite different. ACM's program amounts to a one-year, 120-hour CS-orientation course, whereas ours can reach 450 hours, taught over three years, constituting an extensive study of the subject at the high school level. Moreover, having been appointed by the ministry directly responsible for Israel's educational policy and implementation, we are heavily involved in supervising the many additional activities required to turn a skeleton curriculum into a widely used working program. These include preparing detailed course material, designing teacher-training courses, delegating the task of following an initial field test, and so on.

Our program was designed to fit Israel's particular educational needs but can be applied elsewhere. To understand the context of our work, see the sidebar, "Israel's education system." Computer science has been a subject in Israel's high school curricula since the mid-1970s but has not yet become a fully accepted scientific subject like physics, biology, or chemistry. Instead of the usual 3- and 5-unit programs geared toward matriculation exams, CS was taught in 2- and 4-unit programs (where each unit consists of 90 hours of study). Moreover, many high schools in Israel didn't even offer CS or offered only the 2-unit program. Partly because of this, Israel's university administrators have never taken CS as seriously as other sciences that students may have taken in high school. In the overall evaluation of a university candidate, high school units in CS are not worth as much as units in the other sciences.

The curriculum developed by a Ministry of Education committee in the 1970s included a solid, detailed course in Basic programming. In addition, it called for several electives. If this curriculum had been fully implemented, by now, we might have only needed to update it to reflect changes in the field and a better understanding of the relevant educational issues. However, course material (study books, teacher's guides, and so on) was not always available for the electives in the 1970s program. So, although students learned to program in Basic, they were not always exposed to the entire planned curriculum. That committee did consider algorithms, but the emphasis was on using a programming language. In addition, some teachers did not know enough about the subject matter and taught the material as they saw fit.

Indeed, the "teacher issue," visible from the start, has been very problematic. While well-defined requirements exist to qualify teachers in most high school subjects, the situation in CS is quite different. Unlike teachers of physics or mathematics, CS teachers in many countries are self-taught or have only a high school education. It was only in 1992 that the US organization NCATE (National Council for Accreditation for Teacher Education) adopted standards for US teacher preparation programs, which were to take effect in 1994.[7]

Getting back to the development of curricula in Israel, various new units based on Logo and an electronic spreadsheet language were developed in the 1980s (among other things, to meet the growing demand for teaching com-

Israel's education system

The Israeli education system is basically centralized. The Ministry of Education sets educational policy on all levels and implements it with help from specialized committees, work teams, and professional supervisors.

Students go through ten years of mandatory education, usually divided into elementary school (grades 1 through 6), mid-level school (grades 7 through 9), and one year in high school (grade 10). Two additional high school years (grades 11 and 12) are optional. These three high school years culminate in an extensive set of matriculation exams (called *bagrut* in Hebrew), which are crucial for admission into Israeli universities. The exams are based on a core of required subjects and several additional electives. Subjects are taught in "study units," each denoting three hours per week for a year, or approximately 90 hours. To get past the matriculation hurdle, a student must successfully pass the exams in at least six subjects, accumulating at least 20 total study units, although most students earn more.

Many subjects can be studied at various levels, the most common being 3-unit and 5-unit programs that differ significantly in material quantity and conceptual depth. A 5-unit program would typically require studying the subject for 5 weekly hours throughout the three years of high school.

Required courses include Hebrew (language and literature), English, Bible studies, mathematics, and history. Electives include Talmud studies, geography, the sciences (physics, biology, and chemistry), and several other courses. In addition, some schools have a technological track, where students study some technical subject intensively to prepare for specialization during their future studies.

Obviously, no two educational systems are quite the same. But high school studies in most countries contain a scientific component that is comparable to—although in some places not as extensive as—that of the Israeli system. Our work can therefore be applied to other countries as well.

puter literacy). Some schools adopted these in place of part or all of the curriculum, which sometimes entailed moving even further away from true computer science.

Our committee was formed in 1990. It includes a researcher specializing in the educational aspects of mathematics and CS (Gal-Ezer), three computer scientists interested in educational issues (Beeri, Harel, and Yehudai), two experienced high school teachers of computer science, and four education professionals from the Ministry of Education, including the head of the computers and computer science section.

We started out by reviewing the existing situation, and concluded that the whole issue must be readdressed and a new and carefully thought-out CS program must be developed for high school (grades 10 through 12). (In Israel, grades 1 through 10 are mandatory, while grades 11 and 12 are optional.) We were convinced that the committee should not only decide on the general topics and principles, but should also prepare detailed and rigorous syllabi for all units in the program; it should help form and supervise the teams that prepare the course material, providing them with continuous technical feedback; it should be involved in teacher training activities; and it should guide and follow a small-scale initial implementation of its recommendations.

THE NEW PROGRAM

Underlying principles

Before getting into a more detailed description of the new program, it is helpful to pinpoint the principles we have used to guide our work, some of which recapitulate issues discussed above. In reading them, it helps to keep in mind that the program must introduce a new subject; students are not exposed to any computer science before embarking on this program. This is one of the main differences between computer science and other high school subjects.

COMPUTER SCIENCE IS A FULL-FLEDGED SCIENTIFIC SUBJECT. It should be taught in high school on a par with other scientific subjects.

THE PROGRAM SHOULD CONCENTRATE ON THE KEY CONCEPTS AND FOUNDATIONS OF THE FIELD. The main notion to be emphasized throughout the program is that of an algorithmic problem and an algorithm as a solution thereof. To some extent, the more general notion of a system and the accompanying principles of modularization and abstraction should also be discussed. Other topics are to be viewed as building upon these.

TWO DIFFERENT PROGRAMS ARE NEEDED, ONE FOR 3 UNITS AND ONE FOR 5. The first is for students with only a general interest in CS, and the second, which should be deeper and broader, is for those with more specific interest in CS. However, the design of the 3-unit program should take into account that for many students this might be the only exposure to computer science, so some attempt at comprehensive coverage should be made.

EACH OF THE TWO PROGRAMS SHOULD HAVE REQUIRED UNITS AND ELECTIVES. While the entire program should consist of central and important topics, some of these are less crucial than others and can be made elective. Moreover, variance and flexibility are important for their own sake.

CONCEPTUAL AND EXPERIMENTAL ISSUES SHOULD BE INTERWOVEN THROUGHOUT THE PROGRAM. The word "conceptual" here does not mean "impractical." It refers to subjects that are taught in the classroom rather than in the laboratory. This two-track approach, which we dubbed the zipper principle, is one of the salient points of our program (see the sidebar "The zipper principle").

TWO QUITE DIFFERENT PROGRAMMING PARADIGMS SHOULD IDEALLY BE TAUGHT. It is highly recommended

The zipper principle

A major principle in our program's design, and a crucial guideline for teaching it, concerns the interweaving of conceptual and experimental issues. We call this the zipper approach—a little of this followed by a little of that, combining to form a unified whole.

This "zippering" is most visible in the Fundamentals module and the Software Design module. In fact, these two modules constitute a two-track effort; one is conceptual and the other recasts concepts and ideas in practical implementations with a real programming language.

Progress along the two tracks is made in parallel. Instructors first discuss each new concept in the classroom. Then, if needed, they introduce relevant parts of the programming language, which the student gets to practice in the computer laboratory. Manual problem solving is done before the implementation segment—an important matter, as we want students to understand that the concepts are more fundamental than their specific realization in a particular language. Students will become acutely aware of this when exposed to an additional programming paradigm, but we want to drive the idea home from the start.

For example, we recommend that the notion of repetition in algorithms be illustrated with informal, natural language descriptions (such as "capitalize all words in the input list") before any programming language renditions of this concept are introduced. Instructors present a specific programming construct (for repetition, the *while* or *for* statement), and students apply and practice their knowledge by using it. Thus, we don't teach the *while* statement as an entity in its own right, but rather, as one of many possible forms that a repetitive construct can assume in a programming language. Nevertheless, we don't spend too much time discussing abstract notions, since most high school students need the concrete to fully understand the abstract.

The zipper principle is also reflected in the final matriculation exam, with part given in the computer laboratory.

that a student learn a "mother tongue" first, but then, on a more humble scale, be introduced to another language, of a radically different nature, that suggests alternative ways of algorithmic thinking. This emphasizes the fact that algorithmics is the central subject of study.

A WELL-EQUIPPED AND WELL-MAINTAINED COMPUTER LABORATORY IS MANDATORY. This is the responsibility of the school system and entails setting things up to support laboratory sessions and adequate individual "screen time" for students.

NEW COURSE MATERIAL MUST BE WRITTEN FOR ALL PARTS OF THE PROGRAM. The teams that are to prepare the course material must have "real" computer scientists on board, as well as CS high school teachers and researchers in computer science education.

TEACHERS CERTIFIED TO TEACH THE SUBJECT MUST HAVE ADEQUATE FORMAL CS EDUCATION. An undergraduate degree in computer science is a mandatory requirement, as is formal teacher training.

In summary, the program should focus on the most basic, lasting concepts of CS. It must challenge students by relating these concepts to the practical side of computing. And it should train students to handle intellectually demanding tasks.

Structure and content

The 3-unit version of the program consists of 270 hours of study; the 5-unit version, 450. All hours are absolute. The way they are distributed over days and weeks is determined by the schools. The programs are constructed from the following five modules:

- *Fundamentals 1 and 2* (2 units, 180 hours). This 2-unit module provides the foundation for the entire program. It introduces most of the central concepts and teaches how to apply them in a procedural programming language.
- *Software Design* (1 unit, 90 hours). This module is actually a continuation of Fundamentals. It concentrates on data structures, introducing abstract data types in the process. It also takes a step beyond stand-alone algorithms and discusses the design of complete systems.
- *Second Paradigm* (1 unit, 90 hours). In this module, the student is introduced to a second programming paradigm that is conceptually different from the procedural approach adopted in the first two modules. Logic programming and system-level programming units are currently approved; other possibilities include object-oriented, functional, or concurrent programming.
- *Applications* (1 unit, 90 hours). This module concentrates on one particular kind of application, teaching both principles and practice. Currently approved alternatives include computer graphics and management information systems.
- *Theory* (1 unit, 90 hours). This module is intended to expose the student to selected topics in theoretical computer science. Two alternatives are currently approved: a full unit on models of computation (mainly automata) and a two-part unit consisting of models of computation and numerical analysis.

The two Fundamentals units are mandatory for both program versions. The third unit in the 3-unit version can be satisfied by either the Second Paradigm or Applications. In the 5-unit version, Software Design is mandatory and the fourth and fifth units are chosen from among Second Paradigm, Applications, and Theory. Since Second Paradigm and Applications can also be taken as part of the 3-unit alternative, we expect somewhat deeper versions of these modules to eventually be developed for the 5-unit alternative. The modules are now described in more detail.

FUNDAMENTALS 1. This unit is taught in 10th grade and can also serve as a stand-alone minicourse for students who will not study any more computer science. It covers the basic concepts of an algorithmic problem and its solution, the algorithm. It also discusses functions as a refinement mechanism and introduces the notions of algorithmic correctness and efficiency. (See sidebar, "The Fundamentals 1 module.")

The program itself does not impose a specific language. Most of the curriculum is language independent.

The sidebar on the zipper principle explains the underlying pedagogical approach taken in this and other modules. Basically, each subject is introduced first on a conceptual level, including manual exercising, and is then recast in practical form and implemented in a programming language.

Much has been said about the significance of the first programming language, the "mother tongue."[8-10] Many good arguments have been made for adopting nonprocedural styles of programming in the first course—notably, functional languages such as Scheme.[11] Since this is still a controversial issue, we have decided that a high school curriculum should remain in the mainstream; hence, we have adopted a "vanilla" procedural (imperative) style of programming. As of now, we support development of a Pascal version of the material. However, the program itself does not impose a specific language, and teams can develop course material that uses other languages. In fact, most of the curriculum is language independent.

FUNDAMENTALS 2. This second part of the basic material is taught in 11th grade. Some of the topics covered in the 10th grade are revisited and expanded upon in order to deepen the student's understanding of that material. A number of new facets of algorithmic analysis and design are emphasized, such as stepwise refinement and top-down and bottom-up techniques. In addition, the following topics are taught: recursion (only for 5-unit learners), procedures, and two-dimensional arrays. Time efficiency is treated in more detail, and a special section is devoted to more advanced problems, such as searching and sorting.

Software Design. At universities, the course that follows introductory computer science is usually devoted to data structures and data types. However, we had more general goals in mind—namely, to endow each student with a basic understanding of larger systems and their organization principles. Instructors introduce new data structures, such as stacks and binary trees. In the Fundamentals module, procedures and functions were the main structuring tools; in this module, abstract data types are added to help students handle larger systems. This module also touches on dynamic memory allocation and involves the implementation of one or two small systems.

Second Paradigm. A logic programming version is currently available, and another version based on assembly language is in preparation. The former introduces basic logic notions and discusses knowledge representation through facts and clausal rules. Programming is done in Prolog, with recursion, lists, and trees taking a prominent place in the material. The assembly language version presents a computer system's conceptual structure and introduces programming in assembly language. We hope to see additional alternatives developed for this unit—for example, units based on functional or object-oriented programming or concurrency.

Applications. Although this unit can be taken by any student, its content caters to the needs of students on the technological track. The unit should help them apply computers in the profession they pursue. The most relevant non-CS specializations in this track revolve around information systems (for example, in hotel administration) or graphical aspects of computation (for example, in architectural design). Accordingly, two alternatives for the module are being developed: One introduces management information systems, discussing logical file and data organization, a system's life cycle, and basic systems modeling and analysis. The other introduces computer graphics, presenting the basics of representing and manipulating graphic objects and discussing their use in problem solving. In both versions, the student uses a ready-made software package, such as a database system or a computer-aided design (CAD) package, and must complete a final project. Here, too, we hope to see additional alternatives developed.

Theory. This unit exposes the student to topics in theoretical computer science. Two alternatives for the module are under development: one in models of computation, the other in numerical analysis. The first introduces finite automata, pushdown automata, and Turing machines, elaborating on their relative power. It also presents the Church-Turing thesis and briefly discusses computer limitations. The second alternative concentrates on two main topics: iterations for root extraction and solutions of linear systems of equations. Issues discussed include round-off errors, absolute and relative errors, approximate solutions with error control, and ill-conditioned problems. Two versions of models of computation are being developed: one of 90 hours, which covers the full unit, and the other an abridged version of 45 hours, which is taken together with the 45-hour numerical analysis module.

GETTING THE PROGRAM UNDER WAY

Successful program implementation involves two steps: developing the material and teaching it correctly.

Developing the material

When we had the first version of the curriculum planned out, we proceeded by appointing professional teams to prepare detailed syllabi. After the committee's approval

The Fundamentals 1 module

The Fundamentals 1 module is taught in the 10th grade. Besides constituting the basis of both the 3-unit and 5-unit programs, it is intended as a stand-alone minicourse for students who choose not to continue with computer science.

The following list of its main topics reflects three years of experimentation and might still undergo some minor changes. We list them in linear order, with their recommended hours, to show when and how extensively they are first introduced, but most of the topics are not simply taught and then set aside. They are actually "wall to wall" topics, consistently accompanying the material of the entire module (as well as that of Fundamentals 2 and Software Design) with varying intensity:

- Introductory notions, such as algorithms, algorithmic problems, and the execution process (5 hours).
- Introduction to the basic computation model of data, variables, and input/output; emphasis is on viewing a simple program as a sequence of value-changing instructions (15 hours).
- Modularization: constructing an algorithm from simpler ones (3 hours).
- Conditional execution; Boolean conditions with And and Or connectives (9 hours).
- An initial discussion of algorithm correctness, including valid inputs, correctness with respect to an algorithmic problem, and testing the algorithm with sample inputs (4 hours).
- Repetitive execution of various kinds: counters, accumulators, exit conditions, nontermination (12 to 15 hours).
- An initial discussion of algorithms' time efficiency, including running time as a function of input, running time comparisons, and worst-case time behavior (3 hours).
- Functions, emphasizing the use of a function to solve a subproblem and viewing a function call as a new basic instruction (8 hours).
- One-dimensional arrays (12 hours).
- A section that concludes the 10th grade material and includes more complex examples of algorithms and programs, the nesting of control structures, and more (16 to 19 hours).

of the syllabi, course material was to be developed, possibly by different teams.

A typical development team comes from a science teaching department of an Israeli university and has three to four members. We insisted that the team contain at least one computer scientist, one academic researcher with experience in CS education, and one high school CS teacher. We tried to distribute the choice of teams so that as many academic institutions as possible would take part in the effort. This helps ensure program versatility, as different scientific and didactic approaches are represented.

Preparing the syllabi for the various modules was quite a lengthy process, despite the fact that many of the topics are taught at the university level, where we could draw upon accumulated experience. Indeed, several syllabi versions were often needed before we gave final approval. In addition, syllabi were often changed further during the period of course material preparation and even during the field test. Nevertheless, preparing the syllabi was straightforward compared with preparing the course material itself.

Students who don't study CS beyond the 10th grade will still have a well-rounded, albeit simple, view of the subject's important aspects.

One of the first difficulties that we faced in developing the program, which became more acute in writing the course material, involved student population. Ideally, we would have preferred to develop separate course material for the 3- and 5-unit versions, reflecting the significant difference in required breadth and depth. In reality, Israel's educational structure does not encourage this. Students are not required to make decisions on program alternatives in *any* subject until just before the 11th grade. In fact, by the 10th grade, when our program starts, many students aren't sure whether they will even be taking computer science for matriculation. In the 10th grade, a typical science-oriented study group takes all the main scientific subjects available in high school—that is, physics, chemistry, biology, and in some cases computer science. Because students don't make their actual choice of matriculation subjects until just before the 11th grade, we had to plan so that part of the material in our program would be accessible to a heterogeneous collection of students, from future 5-unit learners to those who will not even complete the 3-unit program.

The 10th-grade computer science studies typically involve 3 weekly hours, yielding 90 hours for the year. Therefore, our program's first 90 hours of study, the Fundamentals 1 module, had to not only be sufficiently elementary but also capable of standing alone. In this way, students who don't study CS beyond the 10th grade will still have a well-rounded, albeit simple, view of the subject's important aspects.

Another problematic aspect of the development of course material was our own rather severe underestimation of the effort. On the one hand, as a collection of individuals, our committee has extensive experience both in high school and university teaching and in writing CS textbooks and course material. On the other hand, we should have known that projects like this always take longer, and are more painful and tedious, than one plans. Anyway, we decided to start the development of a core program, consisting of the mandatory modules and a small number of electives. Our rationale was that the availability of good core material would motivate the Ministry of Education to adopt our program as the official high school CS program, which many schools would implement. This would generate sufficient support to drive the development of additional modules as needed.

In retrospect, the decision to develop only a core program first was fully justified—among other reasons, because our optimism was greatly exaggerated. Now, four years later, we have satisfactory syllabi for most of these modules. Improvements are still needed, of course, since we cannot claim to have final versions until the material is thoroughly class-tested and possibly rewritten. But the course material for several modules is satisfactory or at least good enough to get us through the initial period.

Course material preparation on such a large scale requires a significant budget. However, in Israel, committees such as ours do not have operating budgets but can only make recommendations to the Ministry of Education. This means that the success of the proposed program largely depends on our ability to continually convince Ministry personnel of the value of their investment.

Teaching the material

In the fall of 1991, we started a limited field test for parts of the new program. It initially involved eight study groups in five schools, and by the 1994/1995 academic year had grown to around 40 groups in 9 schools. These small numbers reflect the difficulties both of convincing schools to participate in a new, embryonic program and of finding qualified teachers.

In fact, the teacher issue is one of the main difficulties in implementing the program. The anomaly is that most CS teachers do not have a university degree in computer science. This is unacceptable, and one of the committee's decisions has been to require such a degree from any teacher seeking a permit to teach CS in high school. The Ministry of Education's official regulations, rarely adhered to, are that a teacher is required to have a second degree—MSc or equivalent—to teach a full scientific program in the 11th and 12th grades of high school. However, in view of the job market situation, we have tried to be more realistic. Specifically, we require a BSc or its equivalent in CS, or an appropriate BEd in CS education.

While we are confident that this policy can gradually improve the overall quality of CS teaching, it does not solve our present problems. Many teachers have been teaching programming and CS in high schools for years without such formal training. Some of these might quit when the new program becomes fully operational, but the majority will probably want to continue. Hence, we have outlined a special crash course, consisting of about six basic subjects that are taught in university CS departments. Current teachers without CS degrees will be required to complete these courses to teach the new program.

Nevertheless, even the best teachers will need some training to teach the new program's modules. For example, not every CS graduate is familiar with logic programming or information systems, and even those who are can use help in the didactic aspects of teaching such topics in high school. The few teachers chosen for the field test, although better trained than others, still had to spend considerable time in ad hoc teacher training, which the course material developers usually provided in accordance with the committee's guidelines. This was necessary, in part, to dispel some beliefs and habits of the participating teachers. In the field test's initial phases, many teachers were more confident teaching what they knew well—for example, the technicalities of the programming language. Moreover, some teachers did not approve of the emphasis changes. For example, we de-emphasized traditional flowcharts, and they felt deprived of an important technical device; we emphasized pseudocode in algorithm design (as opposed to direct coding), and they thought this new medium was too vague.

Although the ad hoc training greatly helped the initial group of teachers, within a few years we will likely need a massive instructional program for teachers interested in the new program. This is one of the most challenging problems that the committee faces.

The schools have initiated a three-fold increase in the number of study groups joining the field test since its inception.

Other difficulties besides the teacher problem have surfaced in this initial implementation. One is the incredibly diverse student backgrounds. On the one hand, the program must cater to students with no familiarity with computers (except possibly computer games). On the other hand, many students have extensive programming experience. What makes the problem particularly acute is that most of these are self-taught and have developed habits that may severely hinder the orderly study of algorithmics. Another problem is that some schools are short of computers and cannot provide students with the necessary laboratory time; in fact, many labs are inadequately maintained. There is still a long way to go before schools treat their computer labs with the same respect they confer on their physics and chemistry labs. Finally, another objective difficulty arises because Israeli universities do not yet give incoming students the same bonus points for high school CS as they do for other sciences. Until this happens, many students will hesitate to choose CS.

As to the field test itself, the jury is not yet in. Thus far, though, we think it has been quite successful. Judging from teachers' reports and the student exam results, the main goals are apparently being met. In fact, the schools have initiated a three-fold increase in the number of study groups joining the field test since its inception.

WE HOPE THAT WITHIN TWO YEARS OUR PROGRAM will be adopted throughout the Israeli high school system. But much work remains. Some parts of the course material are not yet ready, and even those parts that are will need to be rewritten in a few years to reflect experience gained in wide-scale application. In addition, we've mentioned the need to develop two versions of Fundamentals and new alternatives for the Second Paradigm and Applications modules. We must also devise effective teacher-training courses.

Our work provides a basic, no-frills program for CS study in high schools. But we envision more specialized variants, perhaps based on different didactic approaches, developed to handle heterogeneous students or project-oriented study. Finally, we strongly believe that the program should evolve to use specialized educational software (see Barwise[12] for an example) and state-of-the-art technology such as interactive TV. ∎

Acknowledgments

This work reflects the efforts of many people. First and foremost are the members of the committee itself, which is chaired by Amiram Yehudai. Besides ourselves, it includes Ben-Zion Barta (who heads the computer and CS section in the Ministry of Education), Roni Dayan, Ephraim Engel, Meir Komar, David Levant, and David Sela. We also owe a great debt to the members of the various development teams, too numerous to list individually, whose work and dedication permeate the entire effort.

References

1. D.E. Knuth, "Computer Science and its Relation to Mathematics," *American Mathematical Monthly*, Vol. 81, No. 4, Apr. 1974, pp. 323-343.
2. D. Harel, *Algorithmics: The Spirit of Computing*, 2nd ed., Addison-Wesley, Reading, Mass., 1992.
3. E.W. Dijkstra, "On the Cruelty of Really Teaching Computing Science," *Comm. ACM*, Vol. 32, 1989, pp. 1,398-1,414.
4. D.L. Parnas, "Education for Computer Professionals," *Computer*, Vol. 23, No. 1, Jan. 1990, pp. 17-22.
5. A.W. Biermann, "Computer Science for the Many," *Computer*, Vol. 27, No. 2, Feb. 1994, pp. 62-73.
6. S. Merrit et. al, *ACM Model High School Computer Science Curriculum*, ACM, New York, 1994.
7. H.G. Taylor, L.G. Thomas, and D.G. Kneze, "The Development and Validation of NCATE-Approved Standards for Computer Science Teacher Preparation Programs," *J. Technology and Teacher Education*, Vol. 1, No. 4, Dec. 1993, pp. 319-333.
8. R.I. Wexelblat, "The Consequences of One's First Programming Language," *Software—Practice and Experience*, Vol. 14, No. 7, July 1981, pp. 733-740.
9. M.C.C. Baranauskas, "Observational Studies About Novices' Interactions in a Prolog Environment Based on Tools," *Proc. Seventh Int'l PEG Conf.*, Moray House Inst. of Education, Edinburgh, Scotland, 1993, pp. 537-549.
10. A. Lee and N. Pennington, "The Effects of Paradigm on Cognitive Activities in Design," *Int'l J. Human-Computer Studies*, Vol. 40, No. 4, Apr. 1994, pp. 577-601.
11. H. Abelson and G.J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Mass., 1985.
12. J. Barwise and J. Etchemendy, *Turing's World*, CSLI Publication, Stanford, Calif., 1993.

***Judith Gal-Ezer** is a computer science faculty member at the Open University of Israel, Tel-Aviv, and served as its director of development, responsible for course development in all*

subjects. She has also designed undergraduate and graduate study programs in computer science. Her research interests include the educational aspects of mathematics and computer science. Gal-Ezer received BSc, MSc, and PhD degrees in applied mathematics from Tel-Aviv University in 1968, 1971, and 1978, respectively. She is a member of the ACM and the IEEE Computer Society.

***Catriel Beeri** is a computer science professor at the Hebrew University, Jerusalem, where he served as chair of the Department of Computer Science. His research interests include database management systems, with emphasis on theoretical aspects. Beeri received BSc, MSc, and PhD degrees in mathematics from the Hebrew University in 1967, 1969, and 1975, respectively. He has spent two years doing postdoctoral work at the University of Toronto and Princeton University. He has served on the program committees of numerous conferences and workshops related to databases. He is a member of the ACM.*

***David Harel** is the William Sussman professor of mathematics at the Weizmann Institute of Science, Rehovot, Israel, and served as chair of its Department of Applied Mathematics and Computer Science. He is also a cofounder of i-Logix Inc. His research interests include computability and complexity theory, logics of program, database theory, systems engineering, and visual languages. Harel received a BSc degree from Bar-Ilan University in 1974, an MSc degree from Tel-Aviv University in 1976, and a PhD degree from MIT in 1978. He is the recipient of ACM's 1992 Karlstrom Outstanding Educator Award and is a fellow of the ACM and the IEEE.*

***Amiram Yehudai** is an associate professor of computer science at Tel-Aviv University and served as chair of its Department of Computer Science. His research interests include software engineering and programming languages. Yehudai received a BSc degree in electrical engineering from the Technion in 1973 and MSc and PhD degrees in computer science from the University of California, Berkeley, in 1974 and 1977, respectively. He is a member of the ACM and the IEEE Computer Society.*

Readers can contact Gal-Ezer at the Department of Computer Science, Open University of Israel, 16 Klausner St., Tel-Aviv, Israel 61392; e-mail galezer@cs.openu.ac.il.

Kathleen Swigger, formerly Computer's *Cybersquare area editor, coordinated the review of this article and recommended it for publication. Her e-mail address is kathy@ponder.csci.unt.edu.*