Informatics in Education, 2005, Vol. 4, No. 2, 307–319
© 2005 *Institute of Mathematics and Informatics, Vilnius*

Mini-Projects Development in Computer Science – Students' Use of Organization Tools

Zahava SCHERZ
Department of Science Teaching, Weizmann Institute of Science
Rehovot 76100, Israel
e-mail: zahava.scherz@weizmann.ac.il

Bruria HABERMAN
Department of Computer Science, Holon Academic Institute of Technology
Department of Science Teaching, Weizmann Institute of Science
Rehovot 76100, Israel
e-mail: bruria.haberman@weizmann.ac.il

Received: 09 2005

Abstract. This paper describes a study aimed at identifying different profiles of students' project development processes. Specifically, we assessed the use of abstract data types for the development of knowledge-based projects.

The concept of abstract data types was introduced to high school students who took the course "Computer Science-Logic Programming". During their studies the students learned and practiced various tools and methods of project development, one of which was based on the use of abstract data types as tools for problem solving and knowledge representation.

To this end, a one-day workshop for team development of mini-projects was organized, and the whole development process was audio and video documented, categorized and analyzed. The profiles of team behavior in the project development process were specified. The analysis of the profiles resulted in identifying four types of project development teams, all of which employed some organizing tool in developing their projects. Two types of the developing teams used abstract data types and two used other methods. The findings indicated that the process of project development of those who used abstract data types was more structured and more organized than others.

Key words: abstract data types, mini-projects in computer science, project organization tools, logic programming in education.

1. Introduction

A new curriculum in Computer Science (CS) that combines both conceptual and practical issues has been implemented since 1990 in Israeli high schools. One specific goal of the curriculum is to provide the students tools that will enable them to understand the functionality of computer systems and the process of software design (Gal–Ezer *et al.*, 1995; Gal–Ezer and Harel, 1999). This goal can be achieved by incorporating project work into the curriculum (Gal–Ezer *et al.*, 1995; Fincher *et al.*, 2001). The academic CS community believes that the role of projects in the CS 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 (Fincher *et al.*, 2001; Gal–Ezer *et al.*, 1995; Holcombe *et al.*, 1998).

A 90-hours course "Computer Science-Logic Programming" (the LP course), which is part of the new CS curriculum, introduces logic programming as a declarative programming environment (implemented in Prolog) which is suitable for content formalization, knowledge representation, and problem solving (Sterling and Shapiro, 1994). The detailed syllabus of the course is presented in (Gal–Ezer and Harel, 1999).

Students taking the LP course are required to develop knowledge-based projects as a final assignment. While developing their projects the students have to undergo a preliminary process: they have to choose a subject, to perform a literature search, to interview experts, to learn about the knowledge domain, and finally to formally specify the problem that they are going to solve. Only then, can they start the following recommended six-phase process: (1) definition of goals, (2) choice of problem predicates, (3) abstraction, (4) formalization, (5) programming, and (6) testing and debugging. During their studies the students are introduced to various methods of problem solving and knowledge representation and to various programming techniques, which they may use to solve problems and to develop projects. In this paper we focus mainly on the use of abstract data types (Aho and Ullman, 1992; Dale and Walker, 1996; Parnas, 1972) in the development process. We developed an instructional package aimed at teaching abstract data types as tools for problem solving and knowledge representation that may be used to develop small-scale programs as well as complex computer systems. The package includes an instructional model, learning materials for students, a teacher's guide, and a software package of "ADT black boxes" implemented in Prolog (Scherz and Haberman, 1995).

We found that students tend to apply a variety of strategies and techniques when they use ADTs in problem-solving tasks (Haberman *et al.*, 2002), and they regard ADTs as useful tools for problem solving and knowledge representation (Haberman and Scherz, 2003). Here we describe a study that monitored the development of students' projects. This study focused on students' project development strategies and on the role of ADTs in the process of project development.

2. The Study

2.1. *Main Questions*

The goals of the study were related to the following aspects:

The Structure of the Development Process: The goal was to identify different profiles of students' development processes. Specifically, we asked the following questions regarding students' projects:

- □ What were the project's development stages?
- □ What was the order of the stages, and how are they related?

- □ Did the project development follow the six-phase model?

The use of ADT: The goal was to assess students' use of ADTs in developing their projects. Specifically, we asked the following questions:

- □ In what stage of the development process did students find suitable ADTs to solve the given problem?
- □ Did students use ADT black boxes in the formalization stage?
- □ How did the use of ADTs influence the entire project development process?

The use of practical tools: The goal was to specify the role of the computer versus the pencil-and-paper approach in developing projects. Specifically:

- □ In what stages of the development process did the students use the computer?
- □ What different styles of computer usage can be identified?
- □ In what stages did the students use pencil-and-paper?

2.2. *Methodology*

A workshop for team project development: A one-day (8 hours) workshop was conducted with a group of 35 students who studied a Computer Science – LP course. Thirteen teams of 2–5 students developed knowledge-based projects on a variety of subjects, all of which are related to descriptive topics of a qualitative nature (e.g., Biology, Medicine, Nutrition, and Law). The workshop took place in a computer laboratory. Each team worked independently; therefore interaction between the teams was not possible.

The teams had to decide on a general topic for their project before the workshop and to bring relevant knowledge sources (literature, an interview with an expert, etc.).

A nearby school library provided the students with additional relevant literature during the workshop. At the beginning of the workshop the teams received written instructions describing the project requirements. During the workshop the students had to specify the problem and to formalize it in terms of a Prolog program (about 10 meaningful different rules). At the end of the workshop each team had to present a Prolog program along with a written final report describing the project and the team development process. One or two observers watched each team.

The workshop enabled us to follow closely all the development processes of several teams in parallel in a reasonable period of time. The teamwork required the students to discuss out loud their project development activities and therefore enabled us to record their reasoning, and conflicts and ideas throughout the process. This kind of recording is hard to achieve when students work individually.

2.3. *Data Collection*

Several tools were used to collect data and to document the team's development processes:

Videotapes: Video cameras were used to videotape six randomly chosen teams.

Audiotapes: All thirteen teams were audio taped.

Observers: Each team was watched by one or two observers who took notes on the team's activities. The observers were computer science teachers and senior students from the same school who had completed advanced CS courses and had experience in developing knowledge-based projects. The observers were instructed not to interfere with the team's activities, nor to criticize their project's development. However, they were allowed to help the students in case of technical difficulties. The observers were also responsible for ensuring that the different teams do not share information and do not interact.

Interviews: Each team was interviewed once during the development process and after finishing the workshop obligations.

Final and Intermediate Products: Each team presented a Prolog program at the end of the workshop, and a final report about the team development process. All the intermediate products – handout illustrations and notes that were made were collected.

2.4. *Data Analysis*

Researches have effectively used verbal protocol analysis to identify how designers introduce information or knowledge into design process (Atman and Bursic, 1998). Protocol analysis can be applied to students' verbalization in a design project, coding sentences into categories such as problem definition, information, analysis etc. (Atman and Bursic, 1996). Verbal protocols data can be analyzed and represented schematically (Chi, 1997; Schoenfeld, 1985).

A graphic tool that we termed an **activity-chart** was devised to enable a concise depiction of a project's development process (see Fig. 1). This graphical illustration displays a concise description of the development process for each project on a macro level, thus offering a general impression of its nature.

The work of each team was represented by an activity-chart. The chart was prepared from the transcriptions of the audiotapes, the observers' reports, the videotapes (of those teams that were filmed), the teams' handout illustrations and notes, and their final products. The X axis of the chart presents the time duration of the workshop. Eight activities are presented on the Y axis of the chart: (1) choice of subject, (2) definition of goals, (3) use of abstract data types, (4) choice of problem predicates, (5) formalization, (6) testing and debugging, (7) use of the computer, and (8) writing the final report.

Activity No 3 refers to various aspects of the use of abstract data types such as adapting ADTs to the problem, use of ADTs' graphical illustration to present relations between problem-predicates, and the use of ADT black boxes to write the program.

We made an effort to find a statistical tool that would enable us to compare and categorize the teams, but no appropriate method was found. Accordingly, the analysis was carried out using the following method: each of the activity charts was drawn on the same scale and reproduced onto transparencies, which facilitated the comparison of the development processes of the different teams. The transparencies were placed one on top of the other and were compared and grouped in an effort to identify common (or similar) patterns.

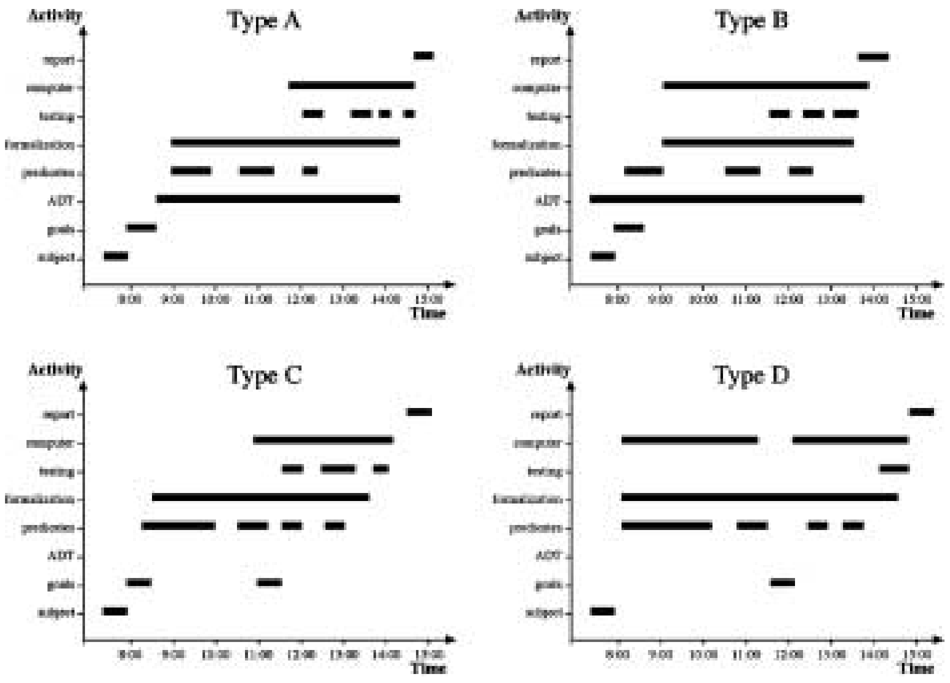

Fig. 1. The activity-charts of the four identified types of project development teams.

Additional information, drawn from our observations, recordings, interviews, and students' reports was used besides the activity charts to establish criteria for defining teams' work profiles.

3. Results and Discussion

The analysis of the activity-charts resulted in identifying four types of project development teams. The classification of the teams appears in Table 1, and the typical activity-charts are presented in Fig. 1.

3.1. *Criteria for Identifying Types of Project-Developing Teams*

Based on our research questions, we established three criteria according to which the profiles of the four types were specified: (1) the structure of the development process; (2) the use of ADTs; and (3) the use of practical tools.

The following section further describes these criteria.

3.1.1. *The structure of the development process*

The structure of the development process is determined by the following parameters:

Problem analysis: The first step in any project development involves defining the problem to be solved and its analysis. This activity involves: (a) Breaking up a prob-

Table 1

Types of behavior during project development

Activity	***Type***			
	Type A (3 teams)	**Type B (3 teams)**	**Type C (4 teams)**	**Type D (3 teams)**
Definition of goals	*At the beginning (all the teams)	* At the beginning (all the teams) * In the continuation (2 teams)	* At the beginning (all the teams) * In the continuation (1 team)	* At the beginning (all the teams) * In the continuation (1 team) * Toward the end (1 team)
Use of ADTs	List (all teams)	Tree (2 teams) Graph (1 team)	No use of ADTs	No use of ADTs
Choice of problem-predicates	Immediately after the initial choice of ADTs (all teams)	Immediately after the initial choice of ADTs (2 teams) Gradual – after the initial choice of ADTs (1 team)	Gradual – parallel to formalization (all teams)	Gradual – beginning at the start of development, parallel to formalization (all teams)
The start of the formalization	Use of predefined “black box”, after the initial choice of predicates	Use of a self-defined “black box” (2 teams)	* Immediately after the initial choice of predicates * Formalization alternately combined with the choice of problem predicates	* Immediately at the start of development * Formalization alternately combined with the choice of problem predicates
Use of computer	* From notes to computer 3 hours after the start (3 teams)	* Formalization on the computer (2 teams) * From notes to computer (1 team)	* Formalization on the computer (1 team) * From notes to computer (3 teams)	* From notes to computer at an early stage (3 teams)
Testing and debugging	After the program was keyed-in (1 team)	No uniformity among the teams	* After program was keyed-in (3 teams) * During formalization (1 team)	* After program keyed-in (1 team) * During formalization (3 teams)

lem into sub-problems in a systematic manner. Each of the sub-problems is then dealt with separately, and the relations between the sub-problems are well defined. (b) Identifying the entities involved in each sub-problem and determining how they are related. An organized and systematic analysis of the problem should result in a structured development process; this serves as a basis for the initial stages of project development, such as defining the project's goals, the decision regarding problem-predicates, and knowledge representation.

Stages and order of execution: A project can be developed similarly to the six-phase model, following a linear order. An iterative development is also possible by means of backtracking and the stepwise refinement of the different stages. Other kinds of development processes may include different stages or disregarding some (or most) of the six-phase model stages.

Modularity and stage dependencies: A modular process calls for a clear distinction between the different stages of development, with special attention given to the dependencies between them.

A close examination of the above components should enable defining a project development process as structured or unstructured.

3.1.2. *The use of ADTs*

ADTs may be used at various levels and stages throughout the project's development. We chose to examine the use of ADTs according to the following parameters:

Abstraction: Adapting suitable ADTs that represent a specific problem. This involves two aspects: (a) Identifying the general problem that represents a specific concrete problem, and (b) Mapping the general problem to the appropriate abstract data type model.

Use of ADT black boxes: Mapping the general problem-predicates to the appropriate ADT interface operations.

3.1.3. *The use of practical tools*

Use of the computer: The computer can be used at various stages of the development process and for different tasks. It can be used as an instrument of trial-and-error, as a word processor to key-in a program that was first formalized on paper, and as a tool for testing and debugging. It is interesting to note that when do students start to use the computer, and do they employ it continuously.

Pencil-and-paper: Another aspect of the process relates to the use of pencil-and-paper in developing the program, specifically how much this method is used, at what stages, and whether the formalization is done first on paper and then on the computer.

3.2. *Types of Project-Developing Teams*

We found four types of project-developing teams:

3.2.1. *Type A*

Type A includes three teams that used predefined ADT black boxes.

All the teams performed problem analysis and defined their goals at the very beginning of the development process. They analyzed their problems at an early stage and distinguished between the general problem and the specific data of the concrete problem. The teams decided on ADTs and chose problem-predicates at the beginning of the development process. Only then did they begin the formalization in terms of Prolog facts and rules. All three teams chose the *list* abstract data type to depict their problems, and used transparently the predefined *list black box* to formalize problem-predicates.

The formalization of the initially chosen predicates was performed entirely by means of pencil-and-paper, without use of the computer. Two of the three teams completed the entire formalization before starting to work on the computer, whereas the third team performed most of the formalization using pencil-and-paper, and continued to formalize additional problem-predicates on the computer.

Only at later stage, about three hours after beginning the workshop, did the teams begin to use the computer, first to type-in and edit the already written facts and rules. All the teams performed run-and-test activities as soon as the whole program was keyed-in the computer. Changes in the formalization of the problem-predicates were made as a result of the debugging process.

Fig. 1 shows that ADTs served as meaningful development tools for Type A teams. They used ADTs throughout most of the development process.

All the teams displayed a structured linear process in developing the project, exactly in accordance with the six-phase model suggested above. All the teams followed the same order in executing the stages of project development, and all made a clear distinction between the different stages.

We can conclude that for type A teams, abstract data types served as a means for organizing the entire development process. The use of *list* ADT for representing knowledge and the use of predefined *list black box* for formalization encouraged the development of a systematic structured process.

3.2.2. *Type B*

Type B includes three teams who used ADTs with no predefined black boxes, and tried to construct the suitable black boxes during the project development process.

All the Type B teams analyzed their problems at the beginning of the development process. They started to define their goals immediately as a result of the problem analysis, and two of the three continued to do so as their work progressed. All three teams generalized the problem, distinguishing between the specific and the general problems involved. All of them began their choice of problem-predicates immediately after determining the abstract data type.

The three teams chose suitable abstract data types for the knowledge representation of their problems. However, unlike the Type A teams that opted for the *list* ADT, the Type B teams selected the *tree* or *graph*. One of the teams chose the *tree* ADT at the beginning of the process, and then decided on a subject that could properly be represented by it.

Since the abstract data type here was not the *list* (for which a predefined black box file was available), but instead the *tree* or *the graph* (for which no such predefined black box file was available), the formalization stage was not trivial for the students, in fact it was also very challenging. The students had to write a program to implement the *tree* or *graph* ADT that they had decided to use.

One of the teams performed a mapping between problem-predicates and tree-predicates that could assist them in their formalization. They started by developing a black box to represent the *tree* ADT, copying the formalization of selected tree-predicates from the textbook. This was a relatively short process, and since the team relied on the formalization of tree-predicates in the book, they did not query the program to test the formalization. Once the black box had been constructed, they used it to formalize the problem predicates.

Another team attempted to construct their own black box to represent the *tree* without consulting the textbook. This was a lengthy process, during which the formalized tree-predicates were tested and debugged. Like the previous team, they too used the constructed black box to formalize problem-predicates.

The third team, which had chosen the *graph*, utilized it only at the level of graphical illustration to present data. Although they stated explicitly that the suitable abstract data type for their problem was the *graph*, they did not show any intention of using a black box representing the *graph*. They made no effort to check if there existed a suitable, predefined black box, nor did they attempt to construct one. This team formalized problem-predicates by means of Prolog if-then rules, while referring to the graphic depiction of the abstract data type.

The two teams that constructed a *tree black box* began their formalization directly on the computer at an early stage of the development process. The team that relied on the *graph* to depict the relations between the problem-predicates worked on their formalization at the level of pencil-and-paper, and only later copied their notes into the computer.

The teams did not display uniformity in the way in which they tested the program. The team that had constructed a *tree black box* on its own, with no use of the textbook, entered queries at the very beginning of the formalization process so as to test the formalization of the tree-predicates. The team that had relied on the textbook to construct the *tree black box* did not query the formalization of the tree-predicates, but rather periodically tested the problem-predicates as new ones were added. The team that employed the *graph* at the level of graphic depiction alone performed a run-and-test only after entering the entire program.

Generally speaking, all three Type B teams closely followed the six-stage model in developing their projects. Like the Type A teams, they displayed a systematic structured process. We can conclude that for Type B teams, like those in Type A, ADTs served a means of organizing the development process. Here, however, the lack of a predefined "black box" complicated this process.

3.2.3. *Type C*

Type C included four teams. All the teams defined their goals at the beginning of the process, and one continued to do so during the course of their work. All four teams started to analyze the problem at the very beginning of the workshop. They identified the entities involved in each sub-problem and the relations between them. They generalized the problem, distinguishing between the specific and the general problem involved.

All of the teams chose their problem-predicates by means of a stepwise refinement in the course of formalization, performing a hierarchical mapping of the various predicates. They used illustrations to aid in their mapping.

None of the teams in this group attempted to identify an ADT suitable for the problem. Consequently, the solution was developed at the level of the problem alone, without reference to any formal model to represent it and without the use of general predicates for formalization.

Formalization began immediately after the initial choice of problem-predicates. The most prominent feature of this group was the alternating manner of their work, combining the choice of problem-predicates with their formalization.

Interestingly enough, three of the four teams began their formalization by means of pencil-and-paper, and only later keyed it into the computer. The fourth team began working with pencil-and-paper, and then alternated this activity with work on the computer.

Three teams ran and tested their programs only at the end of the formalization process, and did not add any new problem-predicates according to the results. The fourth team began to query the program during the choice and formalization of the problem-predicates.

The Type C teams only partially followed the six-stage model in developing their projects and displayed a fairly structured process, although less organized than the Type A and Type B teams. We have concluded that while for teams of Type A and Type B, ADTs were used as a project development organizer, for Type C teams, who did not use ADTs, the definition of goals and the choice of problem-predicates served as the organizer for their work.

3.2.4. *Type D*

Type D consisted of three teams. None of the teams in this group performed problem analysis, nor defined goals at the beginning of the development process. In fact, one of the teams never did this at any stage. Another team took a break in the middle of the development, when the members realized that the process was not proceeding well and began again, this time defining their goals. The third team only defined the goals at the end of the development when they were writing up their report and discovered in the instructions that the description of projects' goals was to be included in their report.

The teams did not distinguish between specific instances and the general problem at the start of development. The decision as to which problem-predicates would be formalized through data and which through rules was taken throughout the course of the process.

All the teams continued to choose their problem-predicates throughout most of the process of development, in parallel with their formalization.

None of the teams made any attempt to identify a suitable abstract data type for the problem. Consequently, the solution was formulated at the level of the problem itself, without reference to any formal model to represent it and without the use of general predicates in the formalization. Only while writing up their final report did one of the teams retrospectively define the ADT in order to fulfill the requirements of the report.

Formalization began immediately at the start of development, and was done directly on the computer, without any use of pencil-and-paper. All Type D teams started to work on the computer at a very early stage.

Only one team performed a run-and-test after entering the entire program. The other two teams did so in the course of the formalization.

The Type D teams followed the six-stage model only to a very minor extent. They displayed a development process that was highly unstructured and based primarily on trial and error. The teams displayed no organizing principles throughout much of the development process. Instead, the computer served as their primary tool in developing and organizing the program.

4. Conclusion

Table 2 illustrates the categorization of the four types of project development teams.

We found that each team employed some organizing tool in developing their project. Half of the teams (types A and B) employed abstract data types in one form or another. The use of abstract data types clearly simplified the development and contributed to a systematic structured process. Type A teams who used the predefined *list* ADT black box to formalize problem-predicates, developed their projects in an organized and structured manner. These teams postponed the use of the computer until the final stages of the project's development.

Type B teams that utilized ADTs to present their problems, but did not use predefined ADT black boxes and instead created black boxes of their own, also worked in a structured and organized manner. Some teams had difficulties in creating the black boxes, but this did not interfere with the systematic development process. These teams started to use the computer at an earlier stage than the Type A teams; since they wanted to test the black boxes they created.

Half of the teams (Type C and D) did not employ abstract data types in any form, and worked in a less structured way than the teams that used ADTs. Those teams seemed to use another tool to organize their development processes. Type C teams that defined the project's goals, and chose problem-predicates in the early stages of the process, worked in a more structured way than the Type D teams that did not bother to define goals at all. Eventually Type C teams used the definition of the goals and the relationship between

Table 2

Categorization of the project development processes

Use of Abstract Data Types		**No Use of Abstract Data Types**	
Type A (3 teams)	**Type B** (3 teams)	**Type C** (4 teams)	**Type D** (3 teams)
Predefined ADT black boxes	Self-defined ADT black boxes	Definition of goals	Trial and error
Use of the computer at a very late stage	Use of the computer at an early stage	Use of the computer at a late stage	Intensive use of the computer from the beginning
Structured process	Structured process	Mostly structured process	Unstructured process

problem-predicates as a means of organizing the project development, whereas Type D teams employed trial and error techniques to develop their projects, and used the computer intensively through the entire development process to test their formalization.

We have concluded that project development requires a solid means of organizing the construction and direction of the project. Our instructional approach was to introduce to students a variety of problem-solving tools. Students seemed to use some of these tools as project organizers. Students' employment of project organizers influenced the nature of the entire development process. We found that the choice of ADTs, which are advanced CS problem-solving tools, resulted in a structured and well-organized development process.

References

Aho, A.V., and J.D. Ullman (1992). *Foundations of Computer Science*. W.H., Freeman and Company.

Atman, C.J., K.M. Bursic and S.L. Lozito (1996). An application of protocol analysis to the engineering design process. In *Proceedings of the American Society for Engineering Education Annual Conference*, June, Washington, DC.

Atman, C.J., and K.M. Bursic (1998). Documenting a process: the use of verbal protocol analysis to study engineering student design. *Journal of Engineering Education Special Issue on Assessment*, **87**(2), 121–132.

Chi, M.T.H. (1997). Quantifying qualitative analysis of verbal data: a practical guide. *The Journal of the Learning Sciences*, **6**(3), 271–315.

Dale, N., and H.M. Walker (1996). *Abstract Data Types – Specifications, Implementations, and Applications*. D.C. Heath and Company.

Fincher, S., M. Petre and M. Clark (Eds.) (2001). *Computer Science Project Work Principles and Pragmatics*. Springer-Verlag, London.

Gal–Ezer, J., C. Beeri, D. Harel and A. Yehudai (1995). A high school program in computer science. *IEEE Computer*, **28**(10), 73–80.

Gal–Ezer, J., and D. Harel (1999). Curriculum and course syllabi for a high school CS program. *Computer Science Education*, **9**(2), 114–147.

Haberman, B., E. Shapiro and Z. Scherz (2002). Are black boxes transparent? – High school students' strategies of using abstract data types. *Journal of Educational Computing Research*, **27**(4), 411–436.

Haberman, B., and Z. Scherz (2003). Abstract data types as tools for project development – high school students' views. *Journal of Computer Science Education Online*, January 2003. Available: `www.iste.org/sigcs/community/jcseonline/2003/1/haberman.cfm`

Holcombe, M., A. Stratton, S. Fincher and G. Griffiths (Eds.) (1998). Projects in the computing curriculum. In *Proceedings of the Project 98 Workshop*. Springer-Verlag, London.

Parnas, D.L. (1972). On the criteria to be used in decomposing systems into modules. *Communication of the ACM*, **15**(12), 1053–1058.

Scherz, Z., and B. Haberman (1995). Logic programming based curriculum for high school students: the use of abstract data types. *SIGCSE Bulletin*, **27**(1), 331–335.

Schoenfeld, A.H. (1985). *Mathematical Problem Solving*. Academic Press, Orlando, FL.

Sterling, L., and E. Shapiro (1994). *The Art of Prolog*, 2nd ed. MIT Press, Cambridge, MA.

Z. Scherz has a MSc in biophysics, and a PhD in science education. Her postdoctoral research was performed at the University of Washington's College of Education. In 1984, she joined the staff of the Department of Science Teaching at the Weizmann Institute of Science, where she has led the Logic Programming in Education Group, and currently heads the chemistry and the scientific communication teams at the junior high level. She has written many learning materials for the junior high and high school levels in the areas of logic programming, artificial intelligence, science and technology and high order skills. Her research has focused on student conceptualization of computer science and scientific principles, on their learning of high order skills, as well as on the professional development of leading teachers.

B. Haberman received her PhD degree in science teaching from the Weizmann Institute of Science in 1999. She is currently an instructor in the Department of Computer Science in the Holon Academic 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 Center for high school computer science teachers. She has developed learning materials for high school level in the areas of logic programming and artificial intelligence, and algorithmic patterns. She has developed academic programs for undergraduate level in computer science. Her primary research interests are computer science educational research, students' conceptualization of computer science, as well as in-service teacher education and distance learning.

Nedidelių projektų vystymas informatikoje: moksleivių gebėjimas naudotis organizacinėmis priemonėmis

Zahava SCHERZ, Bruria HABERMAN

Straipsnyje aprašoma studija, kurią atliekant nagrinėti skirtingi studentų projektų vystymo profiliai. Ypatingas dėmesys kreipiamas į gebėjimą naudoti abstrakčiuosius duomenų tipus, skirtus vystyti žiniomis paremtus projektus.

Abstrakčiųjų duomenų tipo koncepcija buvo pristatyta vidurinių mokyklų moksleiviams, lankiusiems „Informatikos – loginio programavimo“ kursą. Pamokų metu moksleiviai buvo supažindinti su skirtingomis projektų vystymo priemonėmis bei metodais, kurių vienas rėmėsi abstrakčiųjų duomenų tipų panaudojimu sprendžiant uždavinius bei reprezentuojant žinias.

Kurso pabaigoje, buvo suorganizuotas vieną dieną trukęs seminaras mini projektų vystymui grupėse, visas vystimo procesas buvo fiksuojamas garso bei vaizdo aparatūra, o vėliau sustruktūruotas bei išanalizuotas. Buvo išskirti atskiri grupės elgsenos atliekant projektą profiliai. Išanalizavus šiuos profilius buvo išskirti keturi projekto vystymo grupių, – vykdant projektą kiekvienoje iš jų pasitelktos vienokios ar kitokios organizacinės priemonės, – tipai. Dviejuose projekto vystymo grupių tipuose buvo pasitektas abstrakčiųjų duomenų tipų metodas, o kitose dviejose – naudoti kiti metodai. Tyrimas atskleidė, kad tiems, kurie projekto vystymo eigoje pasitelkė abstrakčiuosius duomenų tipus, sekėsi kur kas geriau ir jų darbas buvo labiau struktūruotas bei organizuotas, nei tų, kurie vadovavosi kitais metodais.