The Principle of Pattern-Oriented Curriculum and its Implementation in a Computer Science Module for High School Students

Dalit Levy and Tamar Paz

Department of Education in Science and Technology
Technion, Israel Institute of Technology, Haifa 32000 Israel
dality.tamarp@tx.technion.ac.il

Abstract

An approach to the teaching and learning of high school computer science (CS) to and by educationally disadvantaged students (EDS) is described. We begin by outlining six pedagogical principles for designing learning environments appropriate for EDS. Then, the implementation of one of these principles is presented. Results from years of using the CS module developed based on these principles in Israeli schools show that at the end of the module, the students master basic programming tools and recognize basic algorithmic patterns. These findings indicate the potential contribution of the approach outlined here for EDS.

1. Principles of a CS Curriculum for the EDS

CS courses designed for the high school population are challenging, since the students may not have elected to study CS and they often exhibit low motivation for learning CS ideas and concepts. In order to deal with such a challenge, a CS curriculum that is tailored to the needs of high school students is a must. A somewhat neglected aspect of the above challenge is a specific portion of the high school population, which can be referred to as ‘educationally disadvantaged students’, or in short, EDS. As we elsewhere described in details [4], one can find in classes of EDS a unique combination of characteristics that are not common in regular high school classes. When designing a CS module for EDS, the main goal is to develop a learning environment that will suit these students’ characteristics and make the basic CS concepts more accessible for them. The list below describes six principles we found useful for designing such a CS module.

1. Decomposition: This principle calls for organizing the learning material in small and focused content units that would help learners cope with the need for immediate satisfaction.
2. Experience-oriented learning environment: This constructivist principle [1] calls for using student activities in the computer laboratory as the basis from which conceptual understanding can emerge (as opposed to using computers for practice per se). An experience-oriented learning environment avoids situations in which students are merely passive readers or listeners. This principle can help students who are capable of working individually to progress at their own rate and be active in their learning, no matter what the others are doing.
3. Project-based learning: The main idea of this principle is that learning should be accompanied by a requirement to design, implement and present some ‘mini’ programming projects. This principle can help elevate students’ low self-estimation and enable students to show initiative by allowing them to choose their own topics for programming projects.
4. Pattern-oriented curriculum: This principle calls for arranging the curriculum around a set of basic programming patterns. In the software design community, patterns are considered a way of documenting experience by capturing successful solutions to recurring problems [5], but even what is regarded in this community as ‘elementary patterns’ can hardly suit the needs of EDS. Thus, our module is arranged around the most basic programming patterns such as Average and Maximum. A pattern-oriented curriculum might help EDS, who find it easier to pull out a ready-made pattern than to develop one from scratch.
5. Spiral learning path: Following this principle, the learning process is organized in gradual stages, through which a specific concept is repeatedly developed. Each stage helps to foster further understanding of the abstract concept and at the same time helps those students who missed a previous stage.
6. Natural programming: This principle leads one to use the students’ natural language instead of, or in addition to, the English-like formal languages usually used for programming.

These six principles were implemented in two versions of the CS module developed for EDS in Israel. One version uses the Pascal programming environment and for the other version a special Hebrew programming language has been developed, based upon Logo [2]. Due to space limitations, the following section focuses only on one principle in the Pascal version; More can be found in [2,4].

2. Designing a Pattern-Oriented Curriculum

At first glance, the Pascal programming environment might seem less accessible to EDS, for several reasons. One major reason is the lack of immediate feedback from...
the computer; another reason is the fact that the environment does not provide detailed error messages. Still, following the required CS curriculum in Israel even the CS module for EDS must include Pascal programming. Our Pascal version for EDS rose to the challenge, while implementing all of the above six principles. As an example, we will here focus on the implementation of the fourth principle -- designing a pattern-oriented curriculum.

In the CS module for EDS, the concept of 'algorithm' becomes important only when learning about conditionals and loops. Although the algorithms in this learning phase are very simple and involve only basic arithmetic calculations, EDS students might need special support in constructing them properly using a formal language. Following the identification of 5 types of basic computing problems: Print, Sum, Count, Average, and Extreme Value, the curricular design proceeded with the development of learning materials designed to help EDS in the gradual construction of the algorithms needed to solve such computing problems (each problem can appear either in a non-conditional version or in a conditional version). Based upon the principle of a pattern-oriented curriculum, we chose to match each type of computing problem (ten in all) with the algorithmic pattern used to solve it. As a result, the learning process is organized around ten patterns, which are general descriptions written both in natural language and in programming language.

The learning process, using any of the ten patterns, develops through three stages. These stages emerged as a result of a field research that described the process by which EDS solve programming problems [3].

1. In the identification stage the students are presented with a set of real life problems, and are requested to classify these problems. The classification is open, especially in the beginning of the learning spiral, when the class deals with the first pattern. The students offer their different classifications, and discuss the criterions for each classification. The discussion finally converges towards an expression of the computational criterion, which is the kind of computing problem implied by each real life problem. Throughout the discussion, an emphasis is placed on the computational differences between problems. Among the real life problems, there are problems in which different settings are represented by the same computational problem, while others share the same setting but represent different computational problems. Throughout the classification process, the students do not engage in solving the problems, and at this identification stage, they do not know what the algorithms look like or what the program components will be; but in the end of this stage they should be able to recognize problems that "fit" the pattern under discussion.

2. In the construction stage the learners construct a description, in the form of a verbal algorithm, in order to solve the recognized computational problem. In this stage, it helps to engage the students in a kind of role-playing exercise, in which they can dramatize the sequence of acts needed to solve the problem. Such activity should be followed by the generalization of the students' concrete acts into a set of consecutive descriptions, using less "natural" language and more abstract description (towards the pure formal language in the next stage).

3. In the implementation stage the students translate the informal algorithm into a programming language. Here, we further decompose the learning process into three sub-stages, with different learning activities in each sub-stage. First, the learners analyze the components of a given Pascal program, written for solving a specific computing problem. For example: "Mark the instruction for initializing the counter in the given program". Second, they continue with learning tasks in which they complete several missing parts in a given Pascal program, written using the same for pattern as in the first sub-stage (most of the missing parts are related to the setting rather than to the pattern). Third, the most advanced learning tasks are those in which students write a complete Pascal program from scratch.

In the overall learning process, patterns are constructed along with the learners, first using their mother tongue and later the programming language, and the concept of algorithmic patterns is repeatedly developed. Thus, although some of the EDS do not possess the ability to develop the required algorithms, they can still successfully use given patterns and implement them in the programming language. Furthermore, other principles like the decomposition principle and the principle of spiral learning are intertwined into the principle of a pattern-oriented curriculum. As a consequence, even the most advanced parts of the CS module can better fit the needs of EDS.

4. References


