ABSTRACT

Importing design patterns from software engineering to the computer science education (CSE) field was followed by defining patterns and pattern languages suitable for CS courses. The main goal of incorporating patterns in CSE was to enhance students’ programming abilities, as well as their design and problem-solving skills. Accordingly, various instructional materials were suggested for using patterns in classroom learning activities, such as collections of patterns and related programming assignments. However, the existing pattern-based materials seem to be insufficient for implementation in the classroom, especially when teaching introductory courses that emphasize syntax and programming language features. Therefore, alternative methods using applicative models for pattern-based instruction, which emphasize problem solving and program design issues rather than specific language features and syntax, should be developed and assimilated within the CS teaching community. We believe that successful implementation of such models should be accompanied by appropriate teacher-training.

In this paper we describe an initial effort to expose CS teachers to the notions of pedagogical patterns and pattern-based instruction, aimed at motivating them to meaningfully adopt and adapt patterns to their concrete pedagogical needs.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – computer science education, curriculum.

General Terms: Algorithms.

Keywords: Computer Science Education, Teacher Training.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION

A pattern-based instruction approach utilizes a course orientation that shifts from emphasis on learning the syntactic details of a specific programming language to the development of general problem-solving and program-design skills [6]. Very often teachers introduce algorithmic problems as part of programming assignments, which basically are designated for practicing practical aspects of programming and becoming acquainted with specific language features. Moving the emphasis from the language features and syntax to problem-solving orientation might not be easy for teachers to implement, especially in the context of teaching introductory CS courses. The difficulty might be problematic especially in the case of a curriculum conceptual change, like the one that encountered in Israeli high schools [8].

Computer science has been taught in Israeli high schools since the mid-1970s. The curriculum focused mostly on using programming languages and on the implementation and application aspects of programming; only a small part was devoted to teaching the underlying theoretical aspects of computer science. In contrast, the new curriculum designated to replace the “old” one, emphasizes conceptual knowledge rather than syntactic and technical knowledge and is oriented towards algorithmic problem solving [8]. The assimilation of a new curriculum should be accompanied by triggering teachers’ professional change. Teachers who were used to teaching the old curriculum must undergo conceptual change as well as a change in attitude; in addition they must adapt their instruction to teaching properly according to the spirit of the new curriculum.

The assimilation of a pattern-based instructional approach may support the shift from programming-based instruction to problem-solving oriented instruction. Exposing the teachers to definitions of patterns and to programming assignments associated with them does not necessarily lead to their implementation in class. Defining and naming patterns should not be the focus of instruction [6]. “Learning how to build good solutions is the goal, not memorizing patterns... Teachers need to learn both the content of the approach as well as how to present it effectively” [13]. Therefore, we believe that an appropriate scaffolding teacher-training is necessary in order to achieve the adoption of the pattern-based instruction approach by the CS teaching community.
These ideas were the basis for an initial effort to train a group of in-service CS teachers during an annual long-term 112-hours “Patterns and Pedagogy” workshop which took place in the Davidson Regional Teacher Center at the Weizmann Institute of Science. During the workshop the teachers were exposed to pedagogical patterns that seemed to be relevant to their pedagogical needs. In particular, the incorporation of algorithmic patterns into class instruction plans was thoroughly discussed. In this paper we present the pedagogical pattern-centered approach that was suggested to the participants of the workshop, and describe their products.

2. PATTERNS AND PEDAGOGY

Patterns and programming knowledge: Psychological studies suggest that programming expertise is partly represented by a knowledge base of pattern-like chunks, which consists of problems, their solutions and associated information. Given a problem, the expert programmer can retrieve an appropriate solution schema from his memory [14]. A wide variety of research shows that good knowledge organization helps students remember and reuse the information that they have learned.

In contrast, novice programmers often utilize a language syntax-oriented organization of programming knowledge. This kind of knowledge organization does not allow a new problem to be matched with a previously learned solution [14,17]. As a result, novices have difficulty in assembling algorithms. An attempt to deal with novices’ difficulties was done in the field of software design through the use of design patterns. The concept of design patterns is introduced in the context of object-oriented software design, as an attempt to capture the best practice of experts in a domain and to transfer the knowledge to other practitioners who face similar software design problems.

Patterns for instruction: Design patterns were the basis for deriving the idea of using elementary patterns, algorithmic patterns, and pedagogical patterns in object technology education [7], and in teaching computer science [4].

Elementary patterns are simple design patterns that can be used by novices in learning programming. They are more structure-based and refer to operations such as selection, repetition, and the manipulation of linearly indexed data structures.

Algorithmic patterns constitute the basis of solving algorithmic problems. They are solutions to basic recurring algorithmic problems and form the building blocks for the development of algorithms. Different from some elementary programming patterns and idioms, which are abstractions of programming constructs and mechanisms, algorithmic patterns refer to a classification of the problems themselves [12]. Examples of algorithmic patterns are as follows: Counting, Accumulation and Extreme Values Computations, Searching, All-equal / All-different Checking, Inversing Order or Cyclic Shifting of elements in a list, Number Assembling and Disassembling, Finding the most frequent element, and others. A set of about 30 algorithmic patterns was developed for a Fundamentals of CS course [9].

Most of the algorithmic problems that students are engaged in may be developed by either modifying one of the patterns or by compounding two or more patterns by their abutting [linear sequencing], nesting, interleaving or merging [18].

Pedagogical patterns are intended to capture the expert knowledge of the practice of teaching and to transfer that knowledge to other practitioners. They do not, however, express new pedagogical ideas, but rather, the tried and proven solutions for pedagogical problems. Specifically, a pedagogical pattern describes an abstract teaching approach from which contextualized teaching strategies can be generated, so that novice teachers are not forced to start over when they design new courses and learning resources. Communities like the Pedagogical Patterns Project [16] have been collecting many types of patterns that can help teachers teach and students learn.

Linn and Clancy [11] discussed pedagogical aspects of using patterns in problem-solving activities. Efforts to develop programming and pedagogical frameworks that give support for using patterns throughout a CS curriculum, in on-line templates and courses organized around selected patterns, were reported [e.g. 1,6,15,19].

Patterns intend to transfer expertise knowledge and therefore serve as means for peer-communication [10,16] Accordingly, patterns have standard compact forms that have evolved over time [16] so that they can be easily communicated to those who need the knowledge [4]. For example, the definition of design patterns usually consists of the pattern’s name, motivation for use (the problem which provides a context in which the pattern is applicable), solution, results and tradeoffs of applying it, sample code, examples of its use, and related patterns.

3. PATTERNS AND PROBLEM-SOLVING INSTRUCTION

Acquiring a pattern denotes the ability to recognize that a pattern can be applied to solve a given problem. Students have difficulties in identifying patterns related to a problem, and in adapting them to form a correct and efficient solution. Educators criticize CS courses that are strictly organized around patterns. As has been experienced and reported, familiarity with patterns does not necessarily lead to the reuse and correct utilization of patterns. For example, Bergin claims that a course that is organized that way “would not be especially effective, any more than a course that is strictly organized around language syntax features” and suggests a pattern for Teaching Patterns [3].

Pattern-based algorithmic problem solving involves introducing a variety of problems dealing with a certain algorithmic pattern, while illuminating a large spectrum of issues that relate to the pattern. Also, general and specific CS problem-solving aspects, like correctness and efficiency of algorithms, are raised and discussed. Patterns need rich connections to examples in order to enhance their reuse in different contexts [5].

3.1 Guidelines for problem solving instruction around a pattern

Our approach for incorporating patterns in class is based on organizing learning settings that emphasize problem solving while utilizing patterns. Here we suggest general guidelines for choosing and sequencing a set of problems, which deal with various aspects of using a certain pattern. The difficulty level of the introduced problems is gradually increased. The difficulty level related to patterns may be estimated by the extent to which the problem at hand varies from the pattern, and grows when
other algorithmic patterns need to be interwoven to form a solution.

A guideline for the construction of a set of examples may raise the teacher’s consciousness to the notion of covering many ideas related to the pattern, which otherwise might be ignored, or mentioned without underscoring their importance.

The following guidelines concern central aspects of CS problem solving and pattern-centered instruction that should be taken into account when planning a setting for learning.

G1. Representative Example: A simple problem serves as an example for the pattern. The choice of the first presented problem has a special importance, since a concrete example in a specific context is more comprehensive and accessible to students than the abstract pattern.

G2. Pattern Definition: A pattern is abstracted from one or more analogical problems, or by a generalization of solutions for smaller, previously solved problems. The definition of a pattern includes description of its components.

G3. Pattern Name: The pattern’s name captures and illustrates its essence. Referring to a pattern by its name while analyzing problems and solutions leads to high-level discussions among problem solvers.

G4. Similar Patterns and Similar Problems: The instruction tries to tie links between similar problems and similar patterns. While pointing to links, attention is set to differences and similarities between related patterns and some anticipated confusion is confronted.

G5. Comparison of Solutions: Alternative solutions to a given problem, including those that relate to different patterns, should be compared. Especially, the differences in efficiency of algorithms should be discussed. “Generating alternatives has a long-term effect. It helps one to better understand the particular problem under consideration as well as the class of similar problems” [18, p.138]. Furthermore, comparing alternative solutions enhances the ability of solution evaluation.

G6. Typical Uses: Representative contexts in which the pattern is commonly utilized should be introduced.

G7. Common Mistakes and Difficulties: Discussing incorrect solutions related to the pattern points to its common misuses, and to common difficulties.

G8. Pattern Composing: Problems whose solutions may be composed of several patterns or of a multiple use of the same pattern should be introduced and discussed.

G9. Entry and Turning Point: Practicing the modification of pattern-related solutions for similar problems in a variety of ways, in order to satisfy different constraints, may lead to the construction of an algorithm suitable for a new general problem and to a corresponding new pattern. These processes are typical “transitions” from one pattern to another when identifying the most suitable pattern/s for a given problem.

Introducing a pattern through a set of problems constructed according to the suggested guidelines in a variety of contexts may enhance problem-solving activities. Recognizing the resemblance as well as the diversity between problems may encourage the reuse of ideas gained from previously solved problems to be used for a new one. To this end, when students learn to look for common characteristics in problems, understanding the behavior of the solution of a specific problem can simplify the analysis of other problems based on the same pattern [13].

3.3 Demonstration of pattern-based instruction

In this section we present a collection of sample problems intended to teach the Maximum Value Pattern (Figure 1), which was developed according to the pattern-centered approach guidelines. The pattern is abstracted and generalized out of simple cases of finding maximum value of two/three given values, and is defined according to guidelines [G2, G3]. The problems are presented in a general abstract fashion: each problem is associated with a name that describes its essence; its definition actually represents a set of instances of corresponding concrete problems; for each problem the significant, related guidelines are indicated in square brackets.

<table>
<thead>
<tr>
<th>Name: Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state: collection of values.</td>
</tr>
<tr>
<td>Goal: maximal value in the collection</td>
</tr>
<tr>
<td>Algorithm:</td>
</tr>
<tr>
<td>Initialize Max to First_value</td>
</tr>
<tr>
<td>While there are more items do</td>
</tr>
<tr>
<td>Assign next element to Next_Element</td>
</tr>
<tr>
<td>If Next_Element &gt; Max then</td>
</tr>
<tr>
<td>Assign Next_Element to Max</td>
</tr>
<tr>
<td>Remarks: Here important highlights concerning the pattern and its use are indicated, as well as its related patterns.</td>
</tr>
</tbody>
</table>

Figure 1: The Maximum Value Pattern

P0. Simple pattern use: Finding the maximum of a collection of values as a result of applying the pattern as it is (i.e. by a simple mapping of the problem to the basic pattern). [G1, G6].

P1. Bounded collection: Finding the maximum of a collection of values that has a computable lower bound (e.g. collection of positive numbers). This case can be solved by a simple mapping to the basic pattern. Alternative solution is the initialization of Max to the lower bound. A typical misuse of the pattern is wrong initialization of Max where no lower bound exists (e.g. setting initial value of Max to zero for the case of temperatures in Alaska). [G1, G6, G7]

P2. Nesting patterns: Finding the maximum among values generated from processing each number of a given collection (example of processing: computing the sum of digits of each given number; in this case the nested applied pattern is Digits-Assembling). [G7, G8]

P3. Relating to a subset: Finding the maximum among values with a required attribute (e.g. Maximal odd number among a collection of integers). The initial max value has to be found. For that purpose the “Searching an Element in a Collection” pattern is needed. A typical error is ignoring the empty subset case. [G5, G7, G8]

P4. Multiple use of a pattern: Finding the minimum among maximum values (e.g. where to locate fireworks station? Given a distance map, find a city whose furthest distance from any other city is minimal). Here, mathematical aspects of problem analysis might be difficult for the students. [G7, G8]
P5. **Dynamically generated set:** Finding the maximum among values, which are created during a cumulative process (e.g. the maximum number of passengers in a bus during a drive from Tel-Aviv to Haifa with N intermediate stations). In this case, it might be difficult for students to realize how the considered values are dynamically generated through accumulation by compensation. [G7, G8]

P6. **Additional related value:** In addition to the maximum, a related value is to be returned (e.g. place of maximum in a list of elements; the number of times the maximum appeared). Here, whenever a temporary maximum is found, we must update and keep additional related information. Construction of solution dealing with both goals simultaneously might cause difficulties. [G4, G5, G7, G8]

P7. **Insight and Revelation:** Sometimes the most suitable pattern for a given problem, which may lead to an elegant and efficient solution, might not be easily identified by students because the definition (description) of the problem might lead to another pattern (e.g. A group of people is considered to be homogenous if the age difference between any two people is less than five years). Efficient solution requires finding both maximum and minimum. [G4, G5, G9]

4. **THE WORKSHOP**

Fifteen CS teachers participated in the “Patterns and Pedagogy” workshop. The group was very diverse with respect to the teachers’ formal education and teaching experience. About half the participants had a formal academic background in CS and CS education. The participants’ teaching experience ranged between 1 and 20 years.

The 112-hours workshop consisted of 21 bi-weekly, 4-hour sessions in the teacher center and 28 hours for guided development of final assignment.

The workshop was based on: (a) Introducing algorithmic patterns for teaching the fundamentals of CS, which were developed by the Patterns Team at the School of Education in Tel Aviv University [9], and (b) Introducing pedagogical patterns for teaching CS [2]. One specific goal was to simulate the evolution of the pedagogical patterns from the design patterns, and to discuss the relations and the resemblance between both types of patterns.

The instructional methods included lectures, plenums, and small group discussions, recommended reading of printed and web-based instructional materials, analysis of instructional events, and individual guidance and counseling.

For their final assignment the teachers were asked to submit: (a) a sequenced set of problems, intended to incorporate a certain algorithmic pattern in class instruction; and (b) an original pedagogical pattern for teaching CS. They were asked to justify the submitted pattern’s originality, to indicate to which of Bergin’s patterns [2] it is related, and to suggest learning activities based on utilizing the pattern. In order to accomplish the assignment, the teachers had to collect ideas and to identify new pedagogical patterns based on their personal teaching experience.

Preparing the teachers to carry out their final assignments included the following workshop activities: Authentic examples of predefined pedagogical patterns were presented and discussed. Teachers were asked to identify patterns they have used repeatedly in their own teaching. They practiced developing learning activities and learning materials related to specific given patterns.

4.1 **Teachers’ Products**

4.1.1 **Submitted Pedagogical Patterns**

Six pedagogical patterns were submitted by the workshop participants; all of them referred to actual pedagogical problems, each of which was identified by the teachers as very important and relevant to their field work. Some of the patterns are related to essential disciplinary goals such as “how to introduce the notions of CS meaningfully to students”; other patterns are related to typical recurring students’ de-contextualized difficulties and misconceptions, independent of specific CS subjects. The patterns differed in their levels of abstraction, and reflected the writers’ teaching experience and their formal education. Actually, some of them were proto-patterns [16] and some were abstracted too much. Next, we briefly present the submitted patterns:

**Data Flow** - This pattern suggests organizing learning around problem decomposition, using active simulation of developing compound programs, emphasizing the importance of design-first.

**To Know it All** - This pattern suggests that in order to get “the whole picture” of a specific concept, students should be acquainted right from the beginning, with all of its essential components and with related tools.

**Text Comprehension** - This pattern suggests how to deal with guided reading activities, with the common difficulty of comprehending verbal descriptions of given problems.

**Membership in a List** - This pattern deals with students’ difficulties in manipulating compound dynamic data structures.

**The Function Pattern** - This pattern suggests how to organize learning around functions using related Bergin’s pedagogical patterns for teaching CS.

**This Small One Will Become Big** - This pattern suggests organizing learning around central projects that motivate students “to find answers”.

4.1.2 **Problems around specific Algorithmic Patterns**

Four sets of problems were developed, which were intended to incorporate specific algorithmic patterns into class instruction. All the sets included a representing example problem [G1] and were sequenced in a gradually increasing level of difficulty. The problems were mostly composed according to the guidelines and referred to typical logical mistakes that are related with the pattern in question [G7]. These findings reinforced the results of the analysis of the submitted pedagogical patterns, which indicated that teachers are aware of students’ difficulties and common mistakes, and that they are motivated to suggest remedial activities.

As an example, we refer to the work of one team that suggested a set of patterns intended to emphasize both the similarity and the diversity between two closely related algorithmic patterns: (1) **Check if Exists an Element with a Required Attribute**; (2) **Check if All Elements Have a Required Attribute**. The patterns have the same initial state and share a common general structure of the algorithm. However, they present different logical statements about a given list of elements. In order to emphasize the similarity and the diversity between the patterns, the teachers
developed almost identical problems like the following, which deals with decomposing a number to its digits:

<table>
<thead>
<tr>
<th>Given an integer number and a digit,</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Check if all the number’s digits are equal to the given one;</td>
</tr>
<tr>
<td>(b) Check if the given digit is one of the number’s digits.</td>
</tr>
</tbody>
</table>

5. CONCLUSION

When a CS curriculum is mostly focused on conceptual knowledge and is oriented towards algorithmic problem solving, teachers should emphasize problem-solving and design orientation rather than syntactic details and specific language features. However, teachers find it difficult to focus on problem-solving, especially when teaching programming in introductory CS courses. In order to achieve a significant change course orientation from syntax to problem-solving, a conceptual and pedagogical change in instruction should be implemented. We believe that the pattern-based problem-solving instructional approach is suitable for triggering this change. However, instruction strictly organized around defining patterns, without interweaving effective problem-solving activities does not necessarily foster the desired change in orientation. Guidelines regarding the construction of problem-solving class activities, may raise the teachers’ consciousness for the need to cover many ideas related to patterns which otherwise might be ignored or mentioned without underscoring their importance.

In this paper we described a teachers’ workshop whose aim was to present the notions of pedagogical patterns and pattern-based instruction. The analysis of the teachers’ final products indicated that they indeed successfully utilized pedagogical patterns to plan classroom activities relevant to their concrete needs. Moreover, the exposure to the pattern-based instruction approach triggered the shift from programming-based instruction to problem-solving-oriented instruction. According to their products, the teachers seemed to adopt the general idea of interweaving patterns and pedagogy.

6. REFERENCES


