On Teaching Abstraction in Computer Science: Secondary-School Teachers' Perceptions vs. Practices

Liat Nakar*
Department of Science Teaching, Weizmann Institute of Science
liat.nakar@weizmann.ac.il

Michal Armoni
Department of Science Teaching, Weizmann Institute of Science
Michal.armoni@weizmann.ac.il

ABSTRACT

Abstraction is a central and fundamental idea of computer science (CS), which is widely used, for example, for simplifying problems and designing algorithms. Many argue that abstraction skills are the most necessary ones for computer scientists. In line with this, abstraction is acknowledged as an essential aspect of CS curricula. However, the literature indicates that CS teachers may lack knowledge about the importance of abstraction and how to teach it. In this qualitative study we closely examined the abstraction-related teaching approaches of eight high-school CS teachers, considering both their reflections on their teaching approach and their actual classroom practice. Our findings indicate that abstraction may not be sufficiently emphasized in class. We also found gaps between the perception of some teachers regarding their abstraction-related teaching approach and their actual way of teaching abstraction in their classrooms.

CCS CONCEPTS

• **Social and professional topics** → K-12 education; Computer science education.

KEYWORDS

K-12, Abstraction, Teaching approach

ACM Reference Format:
Liat Nakar and Michal Armoni. 2023. On Teaching Abstraction in Computer Science: Secondary-School Teachers' Perceptions vs. Practices. In *The United Kingdom and Ireland Computing Education Research (UKICER) conference (UKICER 2023), September 07–08, 2023, Swansea, Wales Uk.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3610969.3611124

1 INTRODUCTION

Abstraction is a central and fundamental idea of computer science (CS) [2, 10, 15, 25], which lies at the heart of CS problem solving, in general, and algorithmic design, in particular. Generally, abstraction is a cognitive means of thinking about an idea, a concept, or any object of thought at different levels of detail, from high levels of abstraction, by ignoring details and focusing on the "big picture", to low levels of abstraction by going into the details of the object under consideration.

When employing abstraction, computer professionals repeatedly move between levels of abstraction, while constantly choosing the appropriate level at which to work [28]. Teaching abstraction is acknowledged as an important aspect of CS curricula at all age levels [3, 7, 18, 27], and it was shown to be rewarding even at the K-12 levels [22, 27]. In particular, it is important to encourage students to be aware of the different levels of abstraction that they use and to acknowledge the benefits of moving between levels of abstraction [3, 16]. However, teachers may lack sufficient knowledge about abstraction [19]. Research that offers instructional methods for teaching abstraction and its different facets, and examines their effects is still scarce.

This study is part of a doctoral research that concerns pattern-oriented instruction (POI) [20] and its connections to the teaching of abstraction and to students' internalization of it [21]. During the research we noticed that a gap may exist between teachers' perceptions regarding the teaching of abstraction, and the way they promote abstraction in practice. Both aspects are essential to effectively teach abstraction. Namely, teachers should be aware of the important role abstraction plays in CS, understand abstraction, and include it in their teaching goals, and they should know how to teach and promote abstraction in practice.

Therefore, we decided to investigate this issue further, and gain deeper insights into the teachers' perceived and actual teaching approaches regarding abstraction. To this end, we closely examined these aspects regarding eight high-school CS teachers. Here we will report on this part of our research.

This paper is organized as follows: Section 2 discusses related work and Section 3 presents our research questions. Sections 4 and 5 present the methodology and our findings, respectively. Finally, Section 6 discusses the results and conclusions.

2 RELATED WORK

Here, we present the background for this research. Section 2.1 elaborates on abstraction, and Section 2.2 deals with learning and teaching abstraction.

2.1 Abstraction

As a fundamental idea of CS, abstraction is used in different CS contexts and is manifested in different ways [22]. Essentially, abstraction is about focusing on the "big picture", seeing the forest rather than the trees. Namely, it is about ignoring details, which enables one to focus only on the relevant details, where relevancy is determined by context. In the context of algorithmic problem solving, at each phase of the solution process other details may

*This research was part of a Ph.D. research conducted at the Weizmann Institute of Science. This author's current affiliation is Ono Academic College. liat.nakar@Ono.ac.il

UKICER 2023, September 07–08, 2023, Swansea, Wales Uk

ACM ISBN 979-8-4007-0876-3/23/09.
https://doi.org/10.1145/3610969.3611124

be ignored, depending on the purpose of the phase. For example, ignoring irrelevant details in the problem's description simplifies the problem and enables the solver to focus on its computational core. Ignoring details can also be expressed as generalization or as distinguishing between "what" and "how". Generalization is about the extraction of the essence and the common properties of several instances [18], while ignoring the details that distinguish between them, for example, defining a class for a set of objects. Distinguishing between "what" and "how" allows one to decide when to focus on the properties of a certain entity, rather than its internal representation. For example, in the context of algorithmic problem solving, one may decide at a certain phase of the solution process to focus on what a piece of an algorithm does (i.e., its interface) rather than on how it is implemented (ignoring the implementation details). All manifestations of abstraction in CS involve thinking at different levels of abstraction and moving between them while constantly choosing the appropriate abstraction level at which to work [11, 25].

Dale and Walker [8] distinguished between procedural abstraction and data abstraction. Procedural abstraction inherently involves distinguishing between "what" and "how" when solving algorithmic problems, as mentioned above (using a top-down strategy for algorithm design employs procedural abstraction, although not to its full extent, since it only involves moving down through the levels of abstraction). Data abstraction is used for abstracting the details of representing different sorts of data. An abstract data structure describes the properties of possible sets of data and includes a collection of operations that can be performed on such sets, without revealing the implementation details to those who use it.

Procedural abstraction can be modeled using the hierarchy suggested by Perrenet, Groote, and Kaasenbrood [24] (hereafter denoted as the PGK hierarchy), who originally used it to capture the concept of an algorithm at different levels of abstraction:

- *Level 4, the problem level:* The highest level of the hierarchy refers to a problem and its characteristics, which are independent of specific solutions.
- *Level 3, the algorithm (object) level*: This level refers to an algorithm as an object that solves a problem and is independent of a specific programming language.
- *Level 2, the program level:* At this level, an algorithm is an executable program in a specific programming language.
- *Level 1, the execution level:* This is the lowest level of the hierarchy; it refers to a specific run of a program, on a specific input, and on a specific machine.

Perrenet et al. [24] used the hierarchy to assess undergraduate students' perception of the concept of an algorithm. Namely, as a cognitive hierarchy that represents the abstraction levels of the perception of this concept. At the same time, it also captures the idea of algorithmic abstraction during algorithmic problem solving, where one needs to constantly move between levels of abstraction to obtain a solution for a given algorithmic problem.

2.2 Learning and teaching abstraction

Many studies, concerning different age levels and different CS contexts, indicated that students' understanding and employment of abstraction are insufficient (e.g., in the context of object-oriented programming [23], data structures [1], algorithmic design [14], and the use of reduction [5]). Consequently, different approaches have been suggested for teaching abstraction, either implicit [e.g., 6] or explicit ones [e.g., 16].

Armoni [3] suggested a framework for the explicit teaching of abstraction in the context of algorithmic problem-solving, which uses the PGK hierarchy for explicitly distinguishing between the solution phases. Among its principles are to work throughout all the levels of abstraction, starting with high levels, going down to lower levels when necessary, and climbing up again when appropriate. Statter and Armoni [27] showed that an implementation of this framework in middle schools was very effective and that the teachers could implement it following only a short preparation.

Waite et al. [29] also used the PGK hierarchy (whose levels they renamed *problem*, *design*, *code*, and *running the code*) when they studied the use of abstraction levels by elementary school teachers, using reflective interviews. They focused mainly on the design level and showed that the teachers used designs in several didactic roles; however, their separation between algorithm and code was often obscured. Liebe and Camp [19] also used reflective interviews to investigate the teaching and assessing of abstraction by K-12 CS teachers. Teachers' answers were based on their own views of abstraction, which were often limited. Their findings indicated that the teachers lacked professional training regarding abstraction, were unsure what abstraction abilities the students need to acquire, and tended not to assess abstraction. Only a few of them referred to abstraction levels. Both studies did not observe and examine teachers' in-class practice.

3 RESEARCH QUESTIONS

We aimed to study the differences between the teachers' perceived and actual teaching approaches, with respect to employing and emphasizing CS abstraction when teaching. Note that teachers may not have an accurate perception of their own teaching.

To this end, we posed two research questions:

- What are the teachers' perceived abstraction-related teaching approaches?
- What are the teachers' actual abstraction-related teaching approaches?

This study was approved by the IRB of the Weizmann Institute of Science and by the Israeli Ministry of Education.

4 METHODOLOGY

We now present the research methodology. Section 4.1 presents the research sample and setting, and Section 4.2 describes the research tools.

4.1 Research sample and setting

We followed several classes for two full school years, from 09/2018 until 07/2020. The research sample in this study comprised the teachers of these classes. The classes included 10^{th}- and 11^{th}-grade CS students (15-17 years old) in an introductory CS course of 180 hours (taught over one or two years). During these two school years (before, during, and after the learning process), we collected rich data from 11 classes and 8 teachers (T1-T8).

This study did not involve an intervention. It was a snapshot study, i.e., the teachers were asked to teach as usual, without any further instructions. Moreover, they were unaware of the research focus, namely, abstraction.

All the teachers who participated in the study were experienced and confident teachers, whose worldview regarding the practice of CS education was rather stable (since we conjectured that new teachers can still perfect their teaching of abstraction in their first years of work). As mentioned above, this study was part of a large research project that concerned POI. In line with this, the teachers were chosen based on preliminary conversations in which we tried to reveal their POI-related teaching approaches. Hence, the sample included some teachers who used POI, some who did not use it, and some who partially used it.

4.2 Research tools

Two research tools were used for data collection to reveal the teachers' abstraction-related perceived and actual teaching approaches, respectively.

Interviews: We used interviews with the teachers to learn about their perceived teaching approach with respect to abstraction. The interviews were clinical [13], half structured, and were conducted via Zoom for about 45 minutes. They took place towards the end of the data collection. The interview protocol was content validated [9] by the second author, an expert in CS education research.

Class observations: Non-interfering observations were conducted in all 11 classes (with varying frequencies) by the first author, throughout the learning process. The observations were audio-recorded. In addition, during the observations, notes as well as photos of the class board were taken. We used the observations to learn how the teachers taught in practice, with respect to abstraction. For that purpose, we chose for each teacher observations that contained rich problem-solving episodes that could potentially lend themselves to the employment of procedural and data abstraction.

As part of the large research project, we also examined the reflection of abstraction in the questions that the teachers included in their exams. The results indicated that in their assessment the teachers hardly expected the students to exhibit abstraction skills. Owing to space limitations, we will not report on this aspect here.

5 ANALYSIS AND FINDINGS

In this section we will focus on diagnosing the teaching approaches with respect to promoting and emphasizing abstraction in teaching. We examined both how the teachers described their teaching (Section 5.1) and how they implemented it in class (Section 5.2), respectively. In each of these sections, we will first describe the analysis method and then present the findings.

In general, for both the perceived and actual teaching approaches, separately, we aimed at classifying the teachers into three abstraction-related classes, as follows:

- Promote-Abs teachers who understand and acknowledge the importance of abstraction, emphasize, demonstrate, and encourage the use of abstraction, for example, by referring to procedural or data abstraction, and by distinguishing between levels of abstraction. These teachers tend to devote considerable time to the higher levels of the PGK hierarchy (the problem and the algorithm levels), in addition to the lower levels (the program and execution levels).
- Partial-Abs teachers who use abstraction partially and inconsistently. They devote some time to the higher levels of the PGK hierarchy, but probably not enough.
- Low-Abs teachers who usually do not consider abstraction. For example, they do not distinguish between levels of abstraction, or are even unaware of them. These teachers usually devote considerable time to the lower levels of the PGK hierarchy, the program level and the execution level, and tend to neglect the higher levels.

The Promote-abs teachers are those who are in accordance with the recommendations for teaching abstraction mentioned in Section 2.2.

Imagining the three approaches positioned on an ordered range, we consider the Promote-Abs and Low-Abs approaches as relatively narrow end-intervals, whereas the Partial-Abs approach represents a wide portion of the range, between these end-intervals.

For both analyses we used qualitative content analysis [26], which relies on a category system used for coding the data. Although the analyses were different, they had common characteristics. First, owing to the deep and wide nature of our analyses, both in the wealth of evidence analyzed and the deep interpretation of abstraction they referred to, the analyses were complex, and consisted of several smaller analyses each. In some of them we first used a top-down approach to construct an initial set of categories based on the relevant literature, whereas in others we started with an empty set. Then, in all the analyses we employed an iterative bottom-up approach, where in every cycle we coded portions of the data, and extended or refined the category set when the need to do so emerged from the data. This process was completed when we reached a stable set. The resulting set was validated by the second author, who repeatedly coded a portion of the data using this set. If the need to modify the category set emerged from the data, it was discussed by the first and the second authors until it was resolved.

The first author then began the analysis by coding a significant portion of the data using the category set. To validate the coding process, a fellow researcher coded about 10% of the data, which were chosen randomly. There were several rounds of discussion in order to come to an agreement, followed by an adaptation of the coding guidelines according to the resolved disagreements. This process ended with an agreement of 95% and 96.6% for the perceived and actual approaches, respectively. Finally, the entire data were coded again using the final guidelines.

5.1 The Perceived Teaching Approach

In this section, we focus on the teachers' perceived teaching approaches, with respect to abstraction. To this end, we used the interviews that we had conducted with the teachers, in which they reflected on their teaching. The teachers were not asked directly regarding their approach to abstraction. Rather, we used the perspective of abstraction to analyze their answers to general questions regarding their teaching (e.g., how they guided their students to solve a new problem and what guided them when they organized a new topic into modular teaching components).

5.1.1 The Analysis Method. The interviews were analyzed based on the general method described above. The analysis units were sentences or a part of a sentence that reflected a coherent meaning or idea. In an initial bottom-up coding, two types of abstraction expressions emerged:

- Semantic expressions – evidence in the teachers' answers that indicated the extent to which they considered abstraction in their teaching.
- Syntactic expressions – language-related evidence of the attention each teacher devoted to each level of abstraction, as expressed in the terminology used throughout their discourse.

For the **semantic expressions** (Type A), we grouped the pieces of evidence according to their content. For example, two groups included evidence regarding "criteria for identifying similarity between problems" and "designing an explicit algorithm before programming", respectively. Then we created super-groups by identifying common characteristics between groups. The following super-groups emerged, reflecting the strong connection between teaching abstraction and the PGK hierarchy:

1. The problem-related super-group included groups of evidence that concerned problems such as "using a representative problem for introducing a new topic" and "the abstraction levels used in the problem descriptions".
2. The algorithm-related super-group included groups of evidence that concerned algorithms, such as "focusing on algorithms" and "focusing on designing algorithms together with the students".
3. The program-related super-group included groups of evidence that concerned programming, for example, "references to language control-flow structures" and "coding order: by blocks or line-by-line".
4. The abstraction-types super-group included two groups of evidence: "procedural abstraction" and "data abstraction".
5. The abstraction-related didactics super-group included groups of evidence that concerned the didactics of teaching abstraction, such as "explicit distinction between abstraction levels", "preference for teaching at a high level of abstraction", and "demonstration of moving up and down between levels of abstraction".

Each piece of evidence was coded as either promoting abstraction or ignoring abstraction. We then quantified the qualitative results as follows: For each teacher and for each group, we assigned weights according to the combined nature of the evidence in that group. Namely, if the evidence in a group was consistent with promoting abstraction, we assigned to this group a weight of 1; if it was consistent with ignoring abstraction, we assigned a weight of -1; if it included contradictory evidence (e.g., organizing the learning based on both algorithmic ideas and low-level language structures), we assigned a weight of 0.5. For each super-group, we computed a score by summing the weights of all its groups. Finally, we computed a total score for semantic expressions for each teacher by summing the super-group scores. Higher scores represent a higher extent of abstraction expressed in the teachers' reflections on their teaching.

Next, we classified the teachers regarding Type A into the three abstraction-related teaching approaches by the quartile of the score. The Promote-Abs and the Low-Abs teachers were those teachers whose scores were in the upper quartile and the lower quartile, respectively. The rest of the teachers were considered as Partial-Abs teachers.

For the **syntactic expressions** (Type B), we examined each teacher's statements throughout the interview, focusing on the terminology used by each teacher and its abstraction level. We were interested in the way the teachers divided their attention between the different levels of abstraction, as reflected by the relative volume that each level occupied in their discourse. Therefore, we coded the data using these four levels as categories, obtaining the distribution over all levels.

We determined the teachers' classification according to Type B as follows: if the most dominant level of the teacher was the program or the execution level, then the teacher was considered as a Low-Abs teacher. If the most dominant level was the problem level, then the teacher was considered as a Promote-Abs teacher. If the most dominant level was the algorithm level, we examined the second most dominant level. If it was the problem level, then we considered the teacher as a Promote-Abs teacher; otherwise, we considered the teacher as a Partial-Abs teacher.

5.1.2 Findings. Table 1 presents the findings for Type A. It specifies the score for each teacher and each super-group. The right-most column presents the total score for each teacher.
Table 2 shows the distribution of the teachers' discourse among the different levels (Type B).
Table 3 presents the classification of the teachers based on Type A and Type B separately, as well as a combined final classification of the teachers regarding their perceived teaching approaches (the right-most column). In the combined classification, a teacher was considered as perceived Promote-Abs or perceived Low-Abs if the teacher's classifications regarding Type A and Type B were both Promote-Abs or Low-Abs, respectively. Otherwise, we classified this teacher as perceived Partial-Abs. This reflects our above-mentioned view of the Promote-Abs and Low-Abs approaches as narrow end-intervals.

5.2 The actual teaching approach

In this section, we focus on the actual teaching approaches of the teachers with respect to abstraction. To this end, we analyzed the class observations, while uncovering information regarding the use of abstraction in practice, throughout their teaching and the problem-solving processes. Owing to space limitations and the depth of the analysis, as well as the wealth of the findings, we will leave out some of the details[1].

5.2.1 The Analysis Method. The classification of the teachers was mostly based on qualitative analyses, in line with the general process described above. The analysis units were discourse elements that reflected a coherent meaning or idea. We were looking for indications of emphasizing and using abstraction in class. We conducted

[1]An elaborated description of this analysis is available as an online appendix in https://www.weizmann.ac.il/ScienceTeaching/research-and-development/computer-science/projects/684.

Table 1: Scores for semantic expressions (Type A)

Super-groupTeacher ID	Problem-related	Algorithm-related	Program-related	Abstraction types	Abstraction-related didactics	Total score
T1	4.5	1	1	1	6	13.5
T2	4.5	2	1	0.5	4	12
T3	2	1	0	1	2.5	6.5
T4	1.5	1.5	1	2	4.5	10.5
T5	3.5	1.5	0	1	0	6
T6	1.5	3	0	1.5	1	7
T7	0.5	1	0	1	0.5	3
T8	2	0.5	0	0.5	-0.5	2.5

Table 2: Distribution of syntactic expressions (Type B)

Teacher ID	Problem	Algorithm	Program	Execution
T1	21	9	5	1
T2	13	2	3	0
T3	6	1	2	0
T4	3	7	6	3
T5	22	8	8	1
T6	9	6	5	2
T7	5	4	7	2
T8	10	2	15	1

Table 3: Final classification by the perceived teaching approach

Teacher ID	Type A	Type B	Perceived abstraction-related teaching approach
T1	Promote-Abs	Promote-Abs	Promote-Abs
T2	Promote-Abs	Promote-Abs	Promote-Abs
T3	Partial-Abs	Promote-Abs	Partial-Abs
T4	Partial-Abs	Partial-Abs	Partial-Abs
T5	Partial-Abs	Promote-Abs	Partial-Abs
T6	Partial-Abs	Promote-Abs	Partial-Abs
T7	Low-Abs	Low-Abs	Low-Abs
T8	Low-Abs	Low-Abs	Low-Abs

two analyses, corresponding to two different aspects, respectively. The first (Aspect I) was performed only on episodes of problem-solving processes, in which we examined the abstraction levels used throughout the process. The second (Aspect II) was performed on the whole teaching practice. Extending our view beyond problem-solving episodes enabled us to examine additional teaching events that could lend themselves to teaching abstraction.

Regarding the **problem-solving episodes** (Aspect I), for each teacher, we identified in the observations several episodes of problem-solving processes (except for T8, for which no episodes were found); we chose three of them (the most detailed ones, in which the process was explicit and there was a large volume of communication between the students and the teacher). We started with the four levels of the PGK hierarchy as an initial category system. Then, the need to refine the set emerged from the data. For example, in some cases, when discussing the general idea of a solution to an algorithmic problem, teachers used programming-related terminology. Hence, we split the algorithm level category into two new sub-categories (i.e., sub-levels) of "pure algorithmic discussions" and "programming-related algorithmic discussions". Consequently, an episode coding resulted in a sequence representing the problem-solving process in terms of abstraction levels and the transitions between them. The classification of the teachers according to this aspect was based on the quality of their problem-solving process, as reflected by the sequences. The quality was determined by its accordance with recommendations for teaching abstraction in problem solving from the literature [3]. For example, describing the problem at a high level of abstraction or moving gradually down between levels of abstraction and climbing up when appropriate were both considered as evidence of promoting abstraction.

Regarding **the whole teaching practice** (Aspect II), a bottom-up phase revealed groups similar to those found in the analysis of the semantic expressions (in the interviews, see Type A in Section 5.1) as well as new ones. For example, in the observations there were several cases in which a teacher was satisfied with an algorithm alone as a solution and did not move on to program it. Hence, a corresponding group was created. Super-grouping by common characteristics, as was done in the Type-A analysis, yielded almost the same super-groups, reflecting the common content world of the introductory CS course. Only the abstraction-types super-group, which emerged from the interviews, was not reflected in the observations, because most of the observed lessons focused on algorithmic problem-solving and did not reach relatively complex data structures. Thus, here the set of super-groups included the problem-related, algorithm-related, program-related, and abstraction-related didactics super-groups. As can be expected from the in-class nature of the observations, here the last super-group was richer; for example, it included a group that contained evidence of events in

Table 4: The perceived and actual abstraction-related teaching approaches

Teacher ID	Perceived	Actual
T1	Promote-Abs	Promote-Abs
T2	Promote-Abs	Low-Abs
T3	Partial-Abs	Low-Abs
T4	Partial-Abs	Partial-Abs
T5	Partial-Abs	Partial-Abs
T6	Partial-Abs	Promote-Abs
T7	Low-Abs	Partial-Abs
T8	Low-Abs	Low-Abs

which the teacher encouraged or prevented multiple solutions for the same problem (thus reflecting the idea that a problem is a generalization of several solutions). Owing to the high volume of the evidence, here we classified the teachers into three classes regarding each super-group separately. These classifications were based on the level of accordance of the evidence of each teacher with the recommendations for teaching abstraction in problem-solving from the literature [3]. We quantified each classification for each teacher as follows: Promote-Abs – 3, Partial-Abs – 2, Low-Abs – 1.

For the final classification of the teachers regarding their actual teaching approaches, we averaged for each teacher all the different classifications related to Aspects I and II, yielding a value for each teacher. Teachers whose values were in the first and last quartiles were classified as Low-Abs and Promote-Abs teachers, respectively. The others were classified as Partial-Abs teachers.

5.2.2 Findings. For the final classification of the teachers according to their actual teaching approach, see the rightmost column in Table 4.

6 DISCUSSION AND CONCLUSIONS

Table 4 depicts the teachers' perceived and actual abstraction-related teaching approaches side by side.

Apparently, for some of the teachers, their actual abstraction-related teaching approach was consistent with their perceived one. This was the case for T1, who was classified as Promote-Abs, for both her perceived and actual teaching approaches. Her interview included statements such as: "I see the technical writing of a loop as something on which one should not spend too much time; however, the loops enable the introduction of algorithmic patterns". Her actual approach was reflected in the class observations when she demonstrated a generalized view of a certain problem, or when she discussed algorithmic solutions without involving implementation details.

Interestingly, for some teachers, the abstraction-related perceived teaching approach differed from their actual one. For example, T2 exhibited the widest gap between the perceived and the actual teaching approach (Promote-Abs and Low-Abs, respectively). Her reflections on her teaching expressed an awareness of the importance of abstraction. For example, she mentioned the merit of designing an algorithm with the students before moving into programming. However, in practice, she focused on the lower levels. For example, she asked her students for an algorithm for a given problem, but immediately moved to writing code.

Therefore, our findings indicate possible gaps between the perceived and actual teaching approaches. These gaps demonstrate an inaccurate perception that some teachers may have regarding their teaching of abstraction. Apparently, even teachers who seem to be aware of abstraction and its importance may neglect to promote it in their actual class practice. Possible explanations may be a partial understanding of abstraction, unacknowledged persistent habits, unwillingness to deal with its instruction, or the effect of students' a-priori preferences towards programming [4].

To gain deeper insights into this phenomenon, we conducted a case study (that will be reported elsewhere) that closely examined T1 and T2, whose class practice exhibited a wide gap regarding the teaching of abstraction, although both reflected on their teaching as promoting abstraction.

Additionally, some teachers seemed to be unaware of abstraction whatsoever. For example, T8 was classified as a Low-Abs teacher (both perceived and actual). This was reflected in his interview, in which we could not find any indication of the design of an algorithm or a separation between levels of abstraction. This was also reflected in class, where he did not use the top-down strategy and did not devote time to algorithm design (he did say once, "I recommend that you not go immediately to programming, first plan your solution"; however, he did not demonstrate it).

Our findings illuminate important phenomena, which other teachers may share. First, that despite the importance of abstraction, teachers may lack knowledge about it and its important role in CS, may not be sufficiently aware of the importance of teaching abstraction, lack knowledge on how to implement instruction that promotes abstraction and how to evaluate it, and may teach it unsatisfactorily. This supports and extends the results of Liebe and Camp [19], which were based only on teachers' reflections. Second, we saw that teachers' reflections about their teaching in general, and specifically about the teaching of abstraction, may be unreliable.

The methods used in this study to assess the teaching approaches of these eight teachers were obtained by using a wide and deep operationalization of abstraction that considers multiple forms of manifestation. These methods can be generalized to a diagnostic framework for assessing teachers' abstraction-related teaching approaches, which will be described in a future publication.

As is usually the case in a qualitative study, our findings were obtained using a very small sample of eight teachers; hence, their generalization is limited. In addition, the number of observations varied among the teachers. Nevertheless, out findings were based on rich data and deep qualitative analysis. This strengthens the validity of the existence of the phenomena described above.

The importance of abstraction calls for further investigation. First, an appropriate course for teachers' professional development can be designed and assessed. Such a course can be based on the teacher preparation used by Statter and Armoni [27], extending it to enhance its effectiveness. Second, our operationalization of an abstraction-related teaching approach may be extended in several ways, for example, to also include the aspect of explicitness in teaching as recommended by Armoni [3] (e.g., a teacher may distinguish between levels of abstraction without explicitly making the

students aware of the distinction). In the larger research project, we examined the connection between the teachers' teaching approach and the students' abstraction performance.

REFERENCES

[1] Dan Aharoni, 2000. Cogito, Ergo sum! cognitive processes of students dealing with data structures. In *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education*. (pp. 26-30).

[2] Alfred V. Aho and Ullman D. Jeffrey, 1992. *Foundations of Computer Science*. Computer Science Press, Inc.

[3] Michal Armoni, 2013. On teaching abstraction in CS to novices. *Journal of Computers in Mathematics and Science Teaching* 32(3), 265-284.

[4] Michal Armoni and Judith Gal-Ezer, 2023. High-school computer science – its effect on the choice of higher education. *Informatics in Education*, 22(2), 183-206.

[5] Michal Armoni, Judith Gal-Ezer and Orit Hazzan, 2006. Reductive thinking in computer science. *Computer Science Education* 16(4), 281-301.

[6] Paolo Bucci, Timothy J. Long and Bruce W. Weide, 2001. Do we really teach abstraction? In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education* (pp. 26-30).

[7] Lillian Cassel, Alan Clements, Gordon Davies, Mark Guzdial, Renée McCauley, Andrew McGettrick, Bob Sloan, Larry Snyder, Paul Tymann and Bruce W. Weide, 2008. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. ACM.

[8] Nell Dale and Henry M. Walker, 1996. *Abstract Data Types: Specifications, Implementations, and Applications*. Jones & Bartlett Learning.

[9] Elena Delgado-Rico, Hugo Carretero-Dios and Willibald Ruch, 2012. Content validity evidences in test development: An applied perspective. *International Journal of Clinical and Health Psychology* 12(3), 449-460.

[10] Edsger W. Dijkstra, 1972. The humble programmer. *Communications of the ACM* 15(10), 859-866.

[11] Edsger W. Dijkstra, 1975. About robustness and the like. EWD 452. *The Archive of Dijkstra's Manuscripts*. Available in https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD452.html.

[12] N. Elangovan and E. Sundaravel, 2021. Method of preparing a document for survey instrument validation by experts. *MethodsX* 8:101326.

[13] Herbert Ginsburg, 1981. The clinical interview in psychological research on mathematical thinking: Aims, rationales, techniques. *For the learning of mathematics* 1(3), 4-11.

[14] Bruria Haberman, 2004. High-school students' attitudes regarding procedural abstraction. *Education and Information Technologies* 9(2), 131-145.

[15] Juris Hartmanis, 1994. Turing award lecture on computational complexity and the nature of computer science. *Communications of the ACM* 37(10), 37-43.

[16] Orit Hazzan, 2008. Reflections on teaching abstraction and other soft ideas. *ACM SIGCSE Bulletin* 40(2), 40-43.

[17] Jeff Kramer, 2003. Abstraction – is it teachable? The devil is in the detail'. In *Proceedings of the 16th Conference on Software Engineering Education and Training*. IEEE Computer Society.

[18] Jeff Kramer, 2007. Is abstraction the key to computing?. *Communications of the ACM* 50(4), 36-42.

[19] Christine Liebe and Tracy Camp, 2019. An examination of abstraction in K-12 computer science education. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 9:1-9).

[20] Orna Muller, Bruria Haberman, and Haim Averbuch. 2004. (An almost) pedagogical pattern for pattern-based problem-solving instruction. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 102-106).

[21] Liat Nakar and Michal Armoni, 2022. Pattern-oriented instruction and students' abstraction skills. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2* (pp. 613-613).

[22] Jacqueline Nijenhuis-Voogt, Durdane Bayram-Jacobs, Paulien C. Meijer and Erik Barendsen, 2021. Teaching algorithms in upper secondary education: a study of teachers' pedagogical content knowledge. *Computer Science Education* 33(1), 61-93.

[23] Rachel Or-Bach and Ilana Lavy, 2004. Cognitive activities of abstraction in object orientation: an empirical study. *ACM SIGCSE Bulletin* 36(2), 82-86.

[24] Jacob Perrenet, Jan Friso Groote and Eric Kaasenbrood, 2005. Exploring students' understanding of the concept of algorithm: levels of abstraction. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 64-68).

[25] Andreas Schwill, 1994. Fundamental ideas of computer science. *Bulletin-European Association for Theoretical Computer Science* 53, 274-274.

[26] Asher Shkedi, 2005. *Multiple Case Narrative: A Qualitative Approach to Studying Multiple Populations*. Vol. 7. John Benjamins Publishing.

[27] David Statter and Michal Armoni, 2020. Teaching abstraction in computer science to 7th grade students. *ACM Transactions on Computing Education (TOCE)* 20(1), 8: 1-37.

[28] Jeannette M. Wing, 2006. Computational thinking. *Communications of the ACM* 49(3), 33-35.

[29] Jane Lisa Waite, Paul Curzon, William Marsh, Sue Sentance and Alex Hadwen-Bennett, 2018. Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools* 2(1), 14-40.