Reduction in CS: A (Mostly) Quantitative Analysis of Reductive Solutions to Algorithmic Problems

MICHAL ARMONI
Weizmann Institute of Science

Reduction is a problem-solving strategy, relevant to various areas of computer science, and strongly connected to abstraction: a reductive solution necessitates establishing a connection among problems that may seem totally disconnected at first sight, and abstracts the solution to the reduced-to problem by encapsulating it as a black box. The study described in this article continues a previous, qualitative study that examined the ways undergraduate computer science students perceive, experience, and use reduction as a problem-solving strategy. The current study examines the same issue, but in the context of a larger population, using also quantitative analysis, and focusing on algorithmic problems. The findings indicate difficulties students have with the abstract characteristics of reduction and with acknowledging reduction as a general problem-solving strategy.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms: Algorithms

Additional Key Words and Phrases: Reduction, reductive thinking, black box

ACM Reference Format:
Armoni, M. 2009. Reduction in CS: A (mostly) quantitative analysis of reductive solutions to algorithmic problems. ACM J. Educ. Resour. Comput. 8, 4, Article 11 (January 2009), 30 pages. DOI = 10.1145/1482348.1482350. http://doi.acm.org/10.1145/1482348.1482350.

1. INTRODUCTION

Reduction is a problem-solving heuristic that is relevant to various areas of computer science (CS). Essentially, solving a problem by reduction means transforming it into a simpler problem (or problems) for which a solution is already known, and constructing or deducing the solution to the original

A summarized version of this article was presented in ITICSE'08.
Author's address: M. Armoni, Department of Science Teaching, Weizmann Institute of Science, Israel; email: Michal.armoni@weizmann.ac.il.

problem based on the solution to the reduced-to problem. This strategy can be useful for solving design problems (for example, software design, algorithm design, or automata design) or proof problems (for example, nonpossibility results such as nondecidability).

Reductive strategy relies on known building blocks—the solutions to the reduced-to problems—that are to be used as black boxes. Therefore the resulting solutions are usually clean and simple-structured. That is, a reductive solution should not be concerned with, or relate to, the details of the solution hidden inside the black box, and need only be certain of its existence. The correctness of that solution can also be taken for granted. Because of the black box characteristic, reduction is closely related to abstraction, a connection on which we elaborate in the next section.

We conducted a series of studies focusing on reductive thinking which varied regarding the population (high school or undergraduate CS students), the context (algorithmic or computational model problems), and research methods (qualitative or quantitative). Each of these studies (on which we elaborate in Section 3) taught us more about the ways students experience and use reduction, and each study laid the foundation for the one that followed. The current study is mostly based on the one preceding it in this series. The previous study [Armoni et al. 2006] was qualitative, mostly interview-based, focusing on a small population. Through that study, we gained some insight into undergraduate students' perceptions of reduction and specifically of black boxes. The current study examines a larger population and uses quantitative methods to analyze a large quantity of students' solutions to algorithmic problems. We believe that the findings of these studies can serve to improve the ways reductive thinking is taught to CS students.

This article is organized as follows: in Section 2 we elaborate on reduction in CS, and in Section 3 we refer to relevant literature. The methodology of the study is described in Section 4, and the findings are described in Section 5. We discuss our results in Section 6.

2. REDUCTION IN CS AND IN CS EDUCATION

Reduction is a problem-solving approach relevant in various disciplines. In other words, it is one of the many habits of minds [Cuoco et al. 1996] that mathematicians and scientists naturally use. Reduction has been long recognized in the area of mathematical problem solving. In his classical book on this topic, Polya [1957] recommended asking students, after presenting them with a new problem, what other previously solved problem the given problem reminds them of.

Although not unique to CS, reduction is widely used in CS problem solving in more than one CS field. A given problem can be reduced to one problem, in which case the reduction involves a *translation* of the given problem to the reduced-to problem. Such reductions are used in computability theory for undecidability proofs, in complexity theory for establishing membership in complexity classes (for example, the class of NP-complete problems) and in algorithmics to obtain solutions for various algorithmic problems. In other

cases, a given problem can be reduced to a few problems, in which case the reduction involves a *decomposition* of the given problem to the reduced-to problems. Such reductions are useful, for example, in software design (though the principle guiding the decomposition may vary among different programming paradigms) and in automata and formal language theory, for solving classification problems (either design problems or proof problems) where the reductive solutions utilize closure properties.

Wing [2006] mentions reduction as a skill characterizing computational thinking and Schwill [1994] includes reduction in his suggested list of CS fundamental ideas. According to Schwill's definition of a fundamental idea this means that Schwill considers reduction as widely applicable in many contexts of computer science (the horizontal criterion) and as a concept that can be taught at many levels of understanding (the vertical criterion). As a fundamental idea, reduction satisfies two other criteria presented by Schwill: the criterion of time and criterion of sense. That is, reduction can be clearly observed in the historical development of CS and will be relevant in the future, and the idea of reduction can be related to everyday language and thinking.

In the context of algorithmic problems, a reductive solution for an algorithmic problem, in its basic, simplest form, can be characterized as follows: identifying a connection between the problem at hand (A) and another, already-solved algorithmic problem (B), a connection that allows translating an input to A to an input to B, and then an output of B to an output of A. The solution to A is thus obtained by translating a given input to A, to an input for B, then activating B's solution, and translating the resulting output to obtain an output for A.

Therefore, there is a strong connection between reduction and abstraction, another central CS habit of mind. This is a twofold connection, expressed by ignoring irrelevant details in two aspects: first, identifying a connection between two problems, which may be of totally different domains, requires ignoring irrelevant characteristics of these problems while focusing on similarities or joint characteristics. Second, reductive solutions rely on the concept of a black box, ignoring the details of the solution to the reduced-to problem. When constructing a reduction-based solution, one only needs to know that a solution to the reduced-to problem exists, can be obtained if necessary, its correctness has already been proved, and its complexity is known. The inner details of that solution can be ignored; thus one works on a higher level of abstraction.

The first aspect, of bridging between problems, is in line with one of the definitions of abstraction, the definition given by Hershkowitz, Schwarz, and Dreyfus [2001]: "An activity of vertically reorganizing previously constructed mathematics into a new mathematical structure." The adverb "vertically" in this definition means that the reorganized structures do not replace the original structures, but come on top of them, thus enabling work in various levels of abstraction. In previous work [Armoni and Gal-Ezer 2006; Armoni et al. 2006] this definition was generalized in the following manner: "An activity of vertically reorganizing structures into a new structure or structures." Reduction falls nicely into this definition since identifying a connection between problems necessitates looking at the given problem through some prism, enabling one to

identify structures in it, structures that are not necessarily clear at first sight, yet coincide with or resemble those of another problem. This aspect of bridging between problems induces a notion that we referred to [Armoni and Gal-Ezer 2006; Armoni et al. 2006] as the level of reductive thinking, namely, the conceptual gap between the original problem and the reduced-to problem. This notion is therefore strongly related to the level of abstraction.

The second aspect, the black box characteristic, differentiates reduction from analogy, another powerful problem solving strategy. Like solution by reduction, solution by analogy necessitates a connection between two problems, but then alters, or takes pieces of, a solution to one problem, to obtain a solution to the given problem, rather then using the known solution as is.

Although reduction is a fundamental CS idea, perhaps because of that, it is not an easy concept to teach. Being applicable in different domains and with respect to different kinds of problems, a fundamental idea is always a *general* idea. Demonstrating it for a specific problem and in a specific context is necessary in order to explain it, but may not be sufficient to achieve *nonspecific transfer* [Schwill 1994], that is, to induce significant learning that will enable students to use reduction for solving other problems in other contexts. This difficulty is common to all CS fundamental ideas, and is already recognized and reported in the literature for several concepts, usually referred to as *soft concepts* [Corder 1990], such as abstraction [e.g., Or-Bach and Lavy 2004; Sanders and Thomas 2007; Turner et al. 2008], recursion [e.g., DiCheva and Close 1996; Kahney 1983; Levi 2001], encapsulation [e.g., Turner et al. 2008] and programming paradigm [Stolin and Hazzan 2007]. The appropriate, effective way to teach fundamental ideas, according to Bruner [1960], is in a spiral manner, which touches many contexts at various levels.

However, in a standard undergraduate CS program, reduction is first mentioned explicitly when learning topics in computability and complexity theories. These topics are usually taught in advanced courses. In addition, as argued by Armoni and Ginat [2008], use of reduction in these contexts is not simple: the direction of reduction is opposite to the natural direction. Instead of reducing an unsolved problem to a solved one (going from unknown to known), as done in reductive solutions to algorithmic problems, a standard reductive proof in computability and complexity theories reduces an already classified problem to an unclassified problem (going from known to unknown). For example, when proving that problem A is NP-complete, one needs to show that another problem, already proved to be NP-complete, can be reduced to A. In addition, in computability theory a proof by reduction is usually a proof by contradiction, known to be a difficult and problematic concept [e.g., Lin et al. 2003; Thompson 1996]. Thus, a first introduction to reduction in these contexts may involve additional cognitive difficulties and may not lend itself to generalizability of the idea of reduction.

Although reduction is a powerful tool for solving algorithmic problems, most standard textbooks that discuss algorithm design and analysis [e.g., Cormen et al. 2001] do not mention reduction explicitly (except in the context of complexity classes). For example, in the textbook by Kleinberg and Tardos [2006], which emphasizes problem solving, heuristics, and techniques, the first explicit

mentioning of reduction, as a tool for solving an algorithmic problem, is only in Chapter 7 (out of 13). This first mentioning of reduction is not accompanied by any explanation, as if it is assumed that students are already familiar with this problem-solving approach. Thus, for most undergraduate CS students, exposure to reduction in the context of algorithmic design is at best limited and hardly ever occurs (if at all) in introductory courses.

3. RELEVANT WORK

One of the important characteristics of reduction is that of a black box encapsulated within the solution. Several studies relate to the concept of a black box (as opposed to a white box or a glass box). Some of these studies examine black boxes in the context of abstraction, but we could not find work that discusses black boxes in the context of reduction as a general problem-solving strategy.

Haberman et al. [2002] described a strategy for teaching abstract data types (ADTs) as black boxes and analyzed students' achievements and the contribution of this strategy to their understanding of the concept of ADT.

Turkle and Pappert [1990], and other references therein] interpreted the ability of working with black boxes as an indicator for the ability for abstraction. They correlated gender and students' need to open a black box. A more recent work [McKenna 2004] presented findings contradicting this correlation.

Black boxes are sometimes introduced as a means for scaffolding, when learning a new topic: Buchberger [1990] recommends hiding previously learned mathematical procedures, which the students already know and understand well, inside black boxes. This encapsulation can assist students in focusing on a currently taught mathematical concept. Haberman and Ben-David Kolikant [2001] also discuss black boxes as a means for scaffolding, for teaching programming concepts, this time hiding new procedures (such as working with variables, input, output, and assignments) inside black boxes. This way novice students would be able to understand the "behavioral interface" of these concepts (for example, the effect of an output statement on other program entities) before coping with details of their implementation.

We conducted a series of studies on reductive thinking. The first study examined reductive thinking of high school students in the context of computational model problems [Armoni et al. 2005]. The findings showed that many students preferred direct, nonreductive solutions, even in cases in which reductive solutions could have significantly decreased the complexity of the solution. It was also found that most of the students who constructed reductive solutions chose straightforward reductions, for which a lower level of reductive thinking is required, even if other, less straightforward reductive solutions could have been more rewarding in terms of design complexity. The second study [Armoni and Gal-Ezer 2006] dealt with reductive thinking among undergraduate CS students in the context of a course on computational models. The findings indicated difficulties students encountered in applying reduction in the context of this course.

The next study [Armoni et al. 2006] was a qualitative, mostly interview-based study that examined the tendency of undergraduate CS students, at

different stages of their studies, to use reductive solutions when solving problems taken from a variety of CS areas. The problems used in that study were in the scope of the courses CS1, data structures, algorithms and computational models. The findings indicated an evolving development of reductive thinking: the first-year students hardly ever used reductive solutions, whereas the more mature students exhibited higher levels of awareness of the concept of reduction as well as its potential use in different problem-solving situations. Yet even the advanced students did not fully utilize the power of reduction, using it only in specific contexts for some kinds of algorithmic problems, not tending to use it for other kinds of problems and demonstrating no transfer of reductive thinking from the domain of algorithms to the domain of computational models.

In that qualitative study, essentially within the framework of grounded theory [Glaser and Strauss 1975], emerged a theory which we found surprising. First, our findings indicated that those students did not conceive of reduction as a rewarding problem-solving heuristic that reflects high problem-solving skills. Second, we became aware of a significant problem that students (even advanced students) had regarding the abstract nature of reduction, specifically with the notion of a black box. A number of students felt that the use of black boxes was illegitimate, and some even referred to it as cheating. Some students had difficulties in working with black boxes, tending to open the black box before or during a reductive solution: referring to the details of that solution, sometimes even relying on its inner details, and on some occasions, reducing the given problem not to another problem but rather to a specific solution to that problem or even to a specific implementation.

We suggested a connection between this emerged theory and the framework of reducing levels of abstraction [Hazzan 2003], according to which students tend to work at lower levels of abstraction when dealing with a relatively new concept (in this case, reduction): an algorithm (solution) is considered to be at an abstraction level lower than that of a problem [Perrenet et al. 2005], and obviously working with an "opened" black box is at an abstraction level lower than that of being able to work while keeping a black box closed.

Since this theory emerged from qualitative analysis, conducted over a small population, it could not be directly generalized, to be applied to other populations. This was the motivation for the study described in this article, where we examined the applicability of this theory to a larger population using quantitative analysis as described in the next section. This is a common combination of qualitative and quantitative research [Hazzan et al. 2006], when a theory that has emerged from a qualitative study is generalized through a quantitative study.

4. RESEARCH METHODOLGY

As mentioned in the previous section, our goal in this study was to reexamine the theory that emerged in the previous, qualitative study. That is, we wanted to investigate whether students appreciate and acknowledge reduction, and whether they can work with black boxes in the context of reduction on a

satisfying level of abstraction. The research strategy induced by this goal is not intervention-based, but rather taking a snapshot of an existing situation. For this examination we decided to focus again on one CS area, this time algorithm design and analysis.

We did not take this snapshot at the end of the undergraduate program, since we wanted to collect the data in a natural setting, and we felt that a natural setting to collect data regarding solution strategies of algorithmic problems is the algorithms course. Many of these students would not learn algorithm design and analysis in a methodological manner in their undergraduate program beyond this course, and therefore, if reductive skills are expected to be developed by graduation, we can expect students to reach a certain level of appreciation of reduction (even if not fully matured) at the algorithms course checkpoint. Therefore we decided to use students' solutions to algorithmic problems given in an algorithms course.

Our data was collected at two institutions. From one institution (A), we collected students' home assignments during one semester. From the other institution (B), we collected students' final exams for two semesters, two final exams from one semester, and three from another. (In this institution there are three final exams each semester and students can choose to take one of six exams: the three exams given in the semester they are registered to the course or the three exams given in the following semester. A student who fails the first exam can take another one out of the exams remaining on this pool of six exams). From these assignments and exams, we analyzed all students' full solutions to all questions that could be solved by reduction: six questions given on homework assignments and 15 questions from final exams. In institution A students were not required to submit all home assignments, and thus some questions given in later assignments were solved by fewer students. In institution B, each exam consisted of five questions, from which students had to choose four, thus the exam questions also varied in the numbers of submitted solutions.

The syllabi of the algorithms course in these two institutions was quite similar, though the textbook used at institution A was Kleinberg and Tardos's textbook [Kleinberg and Tardos 2006], whereas institution B used the textbook by Cormen, Leiserson, and Rivest [Cormen et al. 1990]. In both institutions the algorithm course is mandatory, taken after the data structures course, and before a course on computability theory, and does not involve programming.

We already justified the point at which the snapshot was taken. Our decision to take this snapshot in these two institutions should also be justified, to allow for generalizability of our results. Both institutions are research universities. Institution B is an open admission institute, but with high academic demands throughout the CS program. Institution A is a top-ranked, world-renowned academic institution in our country. The computer science department is considered one of the more popular ones in this institution, has very high entrance requirements, and is comparable to leading CS departments in the U.S. As just mentioned, both institutions use standard and appreciated textbooks for the algorithm design and analysis course. We therefore feel that if a snapshot taken at such institutions indicates problems with

students' reductive skills, such problems may be documented in other schools as well.

Unlike the previous studies on this series, in this study we did not focus on the issue of the tendency to use reduction or the issue of transfer. Actually, as can be seen in Appendix A, some of the questions used in this study directed students toward reduction (explicitly or implicitly), whereas other questions had clear similarities to questions solved by reduction in class. Thus, such questions could not contribute to examination of the tendency to reduction. Instead, we decided to focus on the manner in which reduction was performed and acknowledged. Each solution was analyzed in order to extract from it data regarding the acknowledgement and appreciation of reduction, and the level of abstraction encapsulated in it. For that purpose we defined a set of analysis categories. In order to explain these categories, we should first look at the pattern of a reductive algorithmic solution and identify its major components. Let us take, for example, the following problem:

> Construct an algorithm that given a directed graph $G = (V, E)$, with a weight function on its edges w: $E \rightarrow \{1, 2\}$, and two nodes $s, t \in V$, finds the length of a minimum-weighted path from s to t.

A reductive solution follows:

We solve this problem by reduction to the problem of finding the length of a shortest path in a directed, nonweighted graph.

Given a graph as an input to the original problem, we transform it to a nonweighted graph in the following manner: each two-weighted edge $e = (u, v)$ is replaced by a chain of two new edges $(u, node\_e)$, $(node\_e, v)$, where $node\_e$ is a new node.

We now find the length of a shortest path from s to t in the resulting graph, ignoring its weights, and return this value as output.

The correctness of this solution is proved using a bidirectional argument, proving a one-to-one and onto correlation between paths in the original graph and paths in the new graph, where an x-weighted path in the original graph can be mapped to an x-lengthed path in the new file, and vice versa.

The correctness proof does not refer to the solution of the reduced-to problem; it only assumes that such a solution, proved to be correct, is known. Similarly, the complexity analysis of this solution considers the cost of the graph translation, and of a solution to the reduced-to problem, without relating to its inner details.

The major components of such a reductive solution are as follows: declaring a solution by reduction, identifying the reduced-to problem, performing a correct input/output translation, proving correctness using a bidirectional argument, and analyzing complexity. The solution is characterized by a high level of abstraction, working at the problem level, and therefore we do not want to refer to the inner details of a solution to the reduced-to problem.

This rationale induced a first set of categories:

1. Method of solution (direct or reductive).
2. Acknowledgement of the solution as reductive (made or not).

3. Reference to the reduced-to problem (exists or does not exist).
4. Reference to inner details of the solution to the reduced-to problem (made or not).
5. Proof structure (around a bidirectional argument or not).

A first analysis of the data yielded a refinement of these categories. For example, we saw that an explicit mentioning of the term reduction can be made in various forms, most of which do not express a true understanding of the nature of reduction, and that the reduced-to problem can be mentioned in various stages of the solution. The final set of categories follows. All categories, except for the first, were applied during the analysis only for reductive solutions:

1. Was the problem solved by reduction or solved directly? This category has a subcategory: what was the level of reductive thinking involved in the solution? This subcategory may be relevant for some questions that have a few possible reductive solutions, but not for others.
2. Was the solution declared as a reductive solution, or was the word reduction at all mentioned? This category has a few subcategories: We hope that reduction will be mentioned in terms of problems, as in the example given above. However, in our data, reduction was also mentioned in terms of solutions (e.g., "I use reduction to BFS"), in terms of inputs (e.g., "I reduce the given graph to a new graph"), unspecified (e.g., "I use reduction"), or in terms of "a problem", when the reduced-to problem was identified with its solution (e.g., "I use reduction to the BFS problem").
3. Was the reduced-to problem at all mentioned, or was only a reference to a solution to it made? If it was mentioned, at what phase of the answer (before a reference to its solution, right after mentioning its solution, or at a later stage of the answer)?
4. Were inner details of the solution to the reduced-to problem referred to?
5. Was the algorithm proved formally, using bidirectional mapping, based on a bidirectional argument?

As mentioned above, the first category is not at all a central category in this study. The second category, an explicit reference to reduction, expresses students' acknowledgement of their solutions as reductive. We believe that once students treat reduction as a heuristic, in Schoenfeld's terms [1985], as a tool in their toolbox, the term reduction might carry along with it a list of characteristics, helping to achieve a complete, correct solution. For example, they will remember to identify the reduced-to problem and to construct a correctness proof around a bidirectional argument. Declaring a solution as reductive may serve as an indicator (obviously not decisive) of students' awareness of the fact that their solution falls into the general category of a solution by reduction.

The third and fourth categories examine the level of abstraction in students' solutions. When working in a lower level of abstraction (in the algorithm level rather than in the problem level), one can see references to the solution of the reduced-to problem rather than just to the problem itself, for example, phrasing the answer in terms of BFS, rather than in terms of the nonweighted

shortest path problem. In such cases, the problem itself might not be mentioned at all, or that it might be mentioned late (even very late) in the answer, after the reference to its solution, and sometimes mentioned only implicitly.

A reference to the details of the solution of the reduced-to problem (fourth category), is another indicator of a lower level of abstraction. Instead of assuming only an interface, otherwise treating the solution to the reduced-to problem as a black box, a lower-level answer is phrased in terms of the implementation of the solution to the reduced-to problem. For example, instead of referring only to the shortest path (or its length), obtained by applying a nonweighted shortest path algorithm, one describes it in terms of the variable names in some coding of BFS, explicitly refers to the actions of BFS, its inner data structures, or even its code.

The fifth category examines the completeness of the solution. As noted above, students who are aware of the reductive nature of their answer are more likely to remember the kind of proof that such a strategy necessitates.

These categories do not refer to correctness. For example, a solution can be both reductive and incorrect, just like a proof of correctness can be based on a bidirectional argument and yet be mistaken or incomplete.

We analyzed 737 solutions to 21 questions, classified according to the reduced-to problems, as follows (the questions are given in the appendix):

Group I – Shortest Path Problems: Nine algorithmic problems involved shortest paths in graphs. Three questions (HW1, HW2, and HW3) were given in the first homework assignment at institution A, and we received 59 solutions for each of these questions. Six questions (E1, E2, E3, E4, E5, and E6) were taken from final exams in institution B, and we received 28, 28, 46, 32, 19, and 15 solutions for these questions, respectively.

Group II – Path Counting Problems: Four questions (E7, E8, E9, and E10), included in the final exams of institution B, could be solved by reduction to path counting, that is, the problem of counting (not necessarily simple) paths between two nodes in a directed graph. In the context of reduction, these questions are interesting since they can be viewed at multiple levels of reduction or abstraction: a problem can be solved by reduction to path counting, which in turn can be solved by reduction to matrix multiplication, which can be solved by a specific algorithm such as Strassen's algorithm. Therefore, for these questions we added another category, relevant to reductive solutions, indicating the level of abstraction to which a solution arrived (after possibly passing through higher levels). We received 30, 32, 30, and 31 solutions for these four questions, respectively.

Group III – Maximum Flow Problems: One question (HW4) in the sixth homework assignment in institution A, and two questions (E11 and E12) taken from final exams in institution B, could be solved by reduction to the problem of finding maximum flow in a network. We received 13, 18, and 36 solutions for these questions.

Group IV – Minimum Spanning Tree Problems: Two questions (HW5 and HW6), given in homework assignments at institution A, and one question (E13), taken from a final exam at institution B, could be solved by reduction to

the minimum spanning tree problem. We received 58, 59, and 30 solutions for these questions, respectively.

Group V and VI – Other Problems: One question (E14) from a final exam at institution B, to which we received 32 solutions, could be solved by reduction to the topological sort problem. Another question (E15), taken from a final exam in institution B, and to which we received 23 solutions, required students to use a black-box algorithm that computes a certain matrix operation in order to perform another matrix operation, meeting a given efficiency criterion. Each of these two questions was considered to be in a different (singleton) group.

Note that this classification has some relevance to the teaching process of the courses in the two institutions. Each of the problem classes listed above includes problems relevant to the material covered in a different part of the course.

5. FINDINGS

The analysis described above was conducted for each of the 21 questions. Since the number of questions prevents us from presenting the findings in one table, we chose to divide it into a few tables with some correspondence with the classes listed above. Tables I and II present the findings regarding Group 1 problems for the three homework problems and the six exam problems, respectively. Tables III, IV, and V present the findings regarding the second, third, and fourth groups, respectively. Table VI presents the findings for the fifth and sixth groups. The structure of all the tables is identical, except that Table III contains also data regarding the level of abstraction in students' solutions, as described above, and Tables II and III also contain data regarding the level of reductive thinking involved in students' solutions (a higher level number indicates a higher level of reductive thinking), for questions that have a few possible reductive solutions (E2, E3, E7, E8, E9, and E10). Table VII summarizes the analysis by the groups of questions, as listed above. This last table includes only categories that are relevant to all groups (that is, it does not include rows for level of reductive thinking or for level of abstraction).

Note that percentages in the second and third row, direct solution and reductive solution, respectively, are calculated in each column with respect to the total number of solutions (N, first row). In Tables II and III this is so also for the rows describing the level of reductive thinking. Other percentages are calculated with respect to the number of reductive solutions only (presented for each column in the third row).

Additional comments, relevant to specific questions, rows, columns, or even entries, are given in the tables by means of footnotes.

Of the 738 solutions, 84 (about 11%) were direct, nonreductive solutions. For most questions the portion of reductive solutions was high, with the exception of question E4. As noted above, this is not surprising, since some of the questions directed students towards reduction (explicitly or implicitly), whereas other questions had clear similarities to questions solved by reduction in class, thus using reduction for such questions does not point at a nonspecific

Table I. Questions from Homework Assignments Relating to Shortest Paths in Graphs

Question	HW1	HW2	HW3
N	59	59	60[1]
Direct solution	9 (15.3%)	7 (11.9%)	2 (3.3%)
Reductive solution	50 (84.7%)	52 (88.1%)	58 (96.7%)
Explicit mentioning of the word reduction			
in terms of problems	-	-	-
unspecified	-	1 (1.9%)	2 (3.4%)
in terms of "a problem"	-	1 (1.9%)	1 (1.7%)
in terms of solutions	-	1 (1.9%)	1 (1.7%)
in terms of inputs	-	-	-
none	50 (100%)	49 (94.2%)	54 (93.1%)
Explicit reference to the reduced-to problem			
before mentioning its solution	2 (4%)	3 (5.8%)	9 (15.5%)
right after mentioning its solution	19 (38%)	4 (7.7%)	14 (24.1%)
at a later phase	9 (18%)	21 (40.4%)	31 (53.4%)
no reference	20 (40%)	24 (46.2%)	4 (6.9%)
Reference to inside-details of the solution to the reduced-to problem	16 (32%)	19 (36.5%)	19 (32.8%)
Proof of correctness			
bidirectional	40 (80%)	34 (65.4%)	9 (15.5%)
only one direction proved	3 (6%)	7 (13.5)	11 (19.0%)
nonformal, narrative	-	1 (1.9%)	3 (5.2%)
no proof	7 (14%)	10 (19.2%)	35 (60.3%)

[1]One solution was composed of two phases, each of which could be solved separately by reduction, but only one of them was. These two phases were counted as two independent solutions, one reductive and one direct.

transfer. However, very few of the 654 reductive solutions included all the major components of a reductive solution that were described above.

Looking first at students' explicit acknowledgement of reductive solutions as such, we can see that only about 14% of the reductive solutions were explicitly identified as reductive. This portion was low for most of the questions, ranging between 45% and 40% for two questions (one of them explicitly mentioned reduction), between 40% and 30% for three questions, between 30% and 20% for four questions (one of them explicitly mentioned reduction), and fewer than 14% for the other 12 questions (in six of these 12 questions reduction was not mentioned at all).

Of those solutions that did use the term reduction, it seems that most (63 out of 89, which are almost 71%) did not fully understand the true meaning of reduction since they did not describe the reduction in terms of problems. About 42% of those that used the term reduction (37 out of 89), referred to reduction in terms of inputs, reducing a given graph to another.

There seem to be some differences regarding explicit reference to reduction, and the nature of those references, when examining it over the various groups of questions. For the four questions that could be solved by reduction to path counting (Group II), only in nine of the 119 reductive solutions (less than 8%) was the term reduction mentioned, and only in one of them, was it mentioned

Table II. Questions from Exams Relating to Shortest Paths in Graphs

Question	**E1**[2]	**E2**[3]	**E3**	**E4**	**E5**	**E6**
N	28	28	46	32	19	15
Direct solution	7 (25%)	5 (17.9%)	7 (15.2%)	20 (62.5%)	1 (5.3%)	2 (13.3%)
Reductive solution	21 (75%)	23[4] (82.1%)	39[5] (84.8%)	12 (37.5%)	18 (94.7%)	13 (86.7%)
level 1		2 (7.1%)	27 (58.7%)			
level 2		8 (28.6%)	1 (2.2%)			
level 3		4 (14.3%)				
level 4		4 (14.3%)				
level 5		-				
other[6]		5 (17.9%)	11 (23.9%)			
Explicit mentioning of the word reduction						
in terms of problems	1[7] (4.8%)	1 (4.3%)	2 (5.1%)	1 (8.3%)	1 (5.6%)	-
unspecified	2 (9.5%)	-	2 (5.1%)	-	-	-
in terms of "a problem"	-	-	-	-	-	-
in terms of solutions	2 (9.5%)	1 (4.3%)	-	-	-	-
in terms of input	4 (19%)	1 (4.3%)	6 (15.4%)	-	7 (38.9%)	4 (30.8%)
none	12 (57.1%)	20 (87.0)%	29 (74.4%)	11 (91.7%)	10 (55.5%)	9 (69.2%)
Explicit reference to the reduced-to problem						
before mentioning its solution	-	3 (13.0%)	4 (10.3%)	5 (41.7%)	2 (11.1%)	2 (15.4%)
right after mentioning its solution	-	-	1 (2.6%)	1 (8.3%)	8 (44.4%)	3 (23.1%)
at a later phase	-	-	2 (5.1%)	3 (25%)	2 (11.1%)	-
no reference	21 (100%)	20 (87%)	32 (82.1%)	3 (25%)	6 (33.3%)	8 (61.5%)
Reference to inside-details of the solution to the reduced-to problem	15 (71.4%)	15 (65.2%)	10 (25.6%)	2 (16.7%)	-	1 (7.7%)
Proof of correctness						
bidirectional	13 (61.9%)	7 (30.4%)	7 (17.9%)	2 (16.7%)	9 (50%)	1 (7.7%)
only one direction proved	4 (19.0%)	3 (13.0%)	14 (35.9%)	-	4 (22.2%)	3 (23.1%)
nonformal, narrative	2 (9.5%)	6 (26.1%)	12 (30.8%)	10 (83.3%)	3 (16.7%)	5 (38.5%)
no proof	2 (9.5%)	7 (30.4%)	6 (15.4%)	-	2 (11.1%)	4 (30.8%)

[2]E1 directed explicitly towards reduction.
[3]E2 directed explicitly towards reduction.
[4]We identified a sequence of five reductive solutions to E2.
[5]We identified a sequence of two reductive solutions to E3.
[6]The level could not be determined because the solution was incomplete, or that it was based on an erroneous strategy not corresponding to any of the possible levels.
[7]Without specifying the problem.

in relation to problems. All nine answers were given for one question, and the word reduction was mentioned in that question (Table III).

Similarly, for the two questions in Groups V and VI respectively (Table VI), the reductive strategy was hardly acknowledged; only one solution to question

Table III. Questions That Can Be Solved by Reduction to Path Counting

Question	**E7**	**E8**[8]	**E9**	**E10**
N	30	32	30	31
Direct solution	2 (6.7%)	2 (6.2%)	-	-
Reductive solution[9]	28 (93.3%)	30 (93.8%)	30 (100%)	31 (100%)
level 1	20 (66.7%)	9 (28.1%)	4 (13.3%)	21 (67.7%)
level 2	3 (10%)	10 (31.3%)	24 (80%)	2 (6.5%)
other[10]	5 (16.7%)	11 (34.4%)	2 (6.7%)	8 (25.8%)
Explicit mentioning of the word reduction				
in terms of problems	-	-	-	-
unspecified	-	1 (3.3%)	-	-
in terms of "a problem"	-	1 (3.3%)	-	-
in terms of solutions	-	1 (3.3%)	-	-
in terms of input	-	6 (20%)	-	-
none	28 (100%)	21 (70%)	30 (100%)	31 (100%)
Explicit reference to the reduced-to problem				
before mentioning its solution	8 (28.6%)	7 (23.3%)	15 (50%)	10 (32.3%)
right after mentioning its solution	-	5 (16.7%)	5 (16.7%)	5 (16.1%)
at a later phase	13 (46.4%)	12 (40%)	6 (20%)	12 (38.7%)
no reference	7 (25%)	6 (20%)	4 (13.3%)	4 (12.9%)
Reference to inside-details of the solution to the reduced-to problem	2 (7.1%)	-	-	-
Arriving down to the level of				
path counting	-	1 (3.3%)	1 (3.3%)	-
matrix multiplication	14 (50%)	12 (40%)	10 (33.3%)	22 (71.0%)
solution to matrix multiplication (Strassen)	12 (42.9%)	11 (36.7%)	17 (56.7%)	8 (25.8%)
irrelevant (not reducing to path counting)	2 (7.1%)	6 (20%)	2 (6.7%)	1 (3.2%)
Proof of correctness				
bidirectional	5 (17.9%)	3 (10%)	7 (23.3%)	2 (6.5%)
only one direction proved	9 (32.1%)	5 (16.7%)	1 (3.3%)	-
nonformal, narrative	6 (21.4%)	14 (46.7%)	15 (50%)	21 (67.7%)
no proof	8 (28.6%)	8 (26.7%)	7 (23.3%)	8 (25.8%)

[8]E8 directed explicitly towards reduction.

[9]For each of these four questions, we identified a sequence of two reductive solutions.

[10]The level could not be determined because the solution was incomplete, or it was based on an erroneous strategy not corresponding to any of the possible levels.

E14 mentioned reduction, in terms of inputs, and reduction was not mentioned in any of the solutions to question E15.

Reduction was hardly mentioned in the solutions to the three Group I questions that were included in homework assignments of institution A, and could be solved by reduction to shortest path problems (Table I). In none of these few cases was it mentioned in terms of problems. Things may seem a bit better for the other Group I questions, the exam questions that could be solved by reduction to shortest path problems, given in institution B (Table II), but that is true mostly in terms of mentioning the word reduction. Yet, only in few cases was it mentioned in terms of problems.

Table IV. Questions That Can Be Solved by Reduction to Maximum Flow Problems

Question	**HW4**	**E11**[11]	**E12**
N	13	18	36
Direct solution	1 (7.7%)	-	-
Reductive solution	12 (92.3%)	18 (100%)	36 (100%)
Explicit mentioning of the word reduction			
in terms of problems	2 (16.7%)	1 (5.6%)	5 (13.9%)
unspecified	1 (8.3%)	4 (22.2%)	-
in terms of "a problem"	-	-	-
in terms of solutions	-	-	-
in terms of input	-	2 (11.1%)	3 (8.3%)
none	9 (75%)	11 (61.1%)	28 (77.8%)
Explicit reference to the reduced-to problem			
before mentioning its solution	11 (91.7%)	not relevant	19 (52.8%)
right after mentioning its solution	1 (8.3%)	not relevant	11 (30.6%)
at a later phase	-	not relevant	1 (2.8%)
no reference	-	not relevant	5 (13.9%)
Reference to inside-details of the solution to the reduced-to problem	-	not relevant	3 (8.3%)
Proof of correctness			
bidirectional	8 (66.7%)	4 (22.2%)	17 (47.2%)
only one direction proved	2 (16.7%)	3 (16.7%)	2 (5.6%)
nonformal, narrative	1 (8.3%)	8 (44.4%)	8 (22.2%)
no proof	1 (8.3%)	3 (16.7%)	9 (25%)

[11]In E11 the reduced-to problem was mentioned in the question, and an unspecified solution for it was given to be used as a black box.

On the other hand, while for the three Group IV questions that could be solved by reduction to the minimum spanning tree problem (Table V) the portion of solutions that mentioned reduction (about 86%) is almost the same as the portion reported for all the questions together, most (about 58%) of those that mentioned reduction did so in terms of problems. For the three Group III questions that could be solved by reduction to the maximum flow problem (Table IV), the word reduction was mentioned in 18 of the 66 reductive solutions (about 27%), eight of which (about 45%) mentioned it in terms of problems.

Two of our categories for analysis focused on the level of abstraction in students' solutions. A reference to inside-details of the solution to the reduced-to problem, those that should be hidden inside the black box, indicates a low level of abstraction and so does an omission of reference to the reduced-to problem (referring only to its solution). Apparently, students' solutions to problems involving shortest paths (Group I) exemplified a lower level of abstraction. Many students identified the reduced-to problems used for solving these questions with their standard classic solutions (e.g., BFS, Dijkstra's algorithm) and many students wrote their answers in terms of these solutions. For six of the nine questions in this group, relatively high portions of the solutions referred to inside-details of the solution to the reduced-to problem. In total, of the 286 reductive solutions to questions involving shortest paths in graphs, in 34% we

Table V. Questions That Can Be Solved by Reduction to Minimum Spanning Tree Problems

Question	HW5	HW6[12]	E13
N	58	59	30
Direct solution	4 (6.9%)	-	6 (20%)
Reductive solution	54 (93.1%)	59 (100%)	24 (80%)
Explicit mentioning of the word reduction			
in terms of problems	11 (20.4%)	-	-
unspecified	3 (5.6%)	-	-
in terms of "a problem"	-	-	-
in terms of solutions	2 (3.7%)	-	-
in terms of input	1 (1.9%)	-	2 (8.3%)
none	37 (68.5%)	59 (100%)	22 (91.7%)
Explicit reference to the reduced-to problem			
before mentioning its solution	36 (66.7%)	41 (69.5%)	12 (50%)
right after mentioning its solution	8 (14.8%)	10 (16.9%)	5 (20.8%)
at a later phase	9 (16.7%)	7 (11.9%)	7 (29.2%)
no reference	1 (1.9%)	1 (1.7%)	-
Reference to inside-details of the solution to the reduced-to problem	4 (7.4%)	1 (1.7%)	4 (16.7%)
Proof of correctness			
bidirectional	19 (35.2%)	3 (5.1%)	11 (45.8%)
only one direction proved	3 (5.6%)	-	3 (12.5%)
nonformal, narrative	30 (55.6%)	52 (88.1%)	10 (41.7%)
no proof	2 (3.7%)	4 (6.8%)	-

[12]HW6 was the last item in a three-item question. The first two items directed students toward a reductive solution for item three.

noted reference to inner details of these solutions. Examples for such references are basing the correctness proof on the fact that BFS uses a queue as an inner data structure, using the principle that guides BFS (checking neighbors) or that guides Dijkstra's algorithm ("Dijkstra will prefer this edge over that edge because it checks . . . ") to explain the resulting solution.

Analysis by the other category relating to abstraction level also indicates a low level of abstraction for these problems. In about 48% of the reductive answers, the reduced-to problem was not mentioned at all, and only its solution was mentioned. That is, the solution was phrased in terms of BFS or Dijkstra's algorithm, without mentioning at all the problem of finding a shortest path in a graph. In approximately 80% of the other reductive answers, the reference to the reduced-to problem was made after its solution was mentioned; in 46%, the reference was made very late in the answer, sometimes even implicitly.

As noted above, the four Group II questions that could be solved by reduction to path counting, offer another way of examining the level of abstraction in students' solutions. On the highest level, these solutions can be phrased in terms of the already-solved problem of path counting, while a lower level answer involves the already-solved problem of matrix multiplication, and at an even lower level, answers refer to a specific solution to the matrix multiplication problem (Strassen's algorithm). Whereas very few (only two solutions) relied on details of the solutions inside the black box in designing their

Table VI. Other Questions

Question	**E14**[13]	**E15**[14]
N	32	23
Direct solution	9 (28.1%)	-
Reductive solution	23 (71.9%)	23 (100%)
Explicit mentioning of the word reduction		
in terms of problems	-	-
unspecified	-	-
in terms of "a problem"	-	-
in terms of solutions	-	-
in terms of input	1 (4.3%)	-
none	22 (95.7%)	23 (100%)
Explicit reference to the reduced-to problem		
before mentioning its solution	not relevant	not relevant
right after mentioning its solution	not relevant	not relevant
at a later phase	not relevant	not relevant
no reference	not relevant	not relevant
Reference to inside-details of the solution to the reduced-to problem	9 (39.1%)	not relevant
Proof of correctness		
bidirectional	13 (56.5%)	not relevant
only one direction proved	3 (13.0%)	not relevant
nonformal, narrative	7 (30.4%)	not relevant
no proof	-	not relevant

[13]For E14 we had difficulties categorizing the answers by reference to the reduced-to problem, since the problem and its standard solution share the same name (topological sort).

[14]In E15 the reduced-to problem was mentioned in the question and an unspecified solution for it was given to be used as a black box. The proof was composed of just a series of matrix calculations.

solutions, most reductive solutions did not remain at the abstraction level of path counting (Table III). Of the 108 solutions that used reduction to path counting, only two remained at that level, 58 (almost 54%) went down to the level of matrix multiplication and 48 (approximately 44%) went even one level lower, mentioning Strassen's algorithm. Actually, about 18% of the reductive solutions did not go through the level of path counting, relating only to matrix multiplication (with or without mentioning a solution for that problem). Another 36% mentioned path counting late in the solution.

Things look a bit better regarding Group III (maximum flow) questions. For a moment, we omit from the discussion question E11, and deal only with the other two questions in this group. Question E11 will be discussed below, due to its special characteristics. As for the other two questions, the reduced-to problem was mentioned in 43 out of the 48 reductive solutions (approximately 90%) and in 30 of them (approximately 70%) the reference to the reduced-to problem was made before its solution was mentioned; in another 12 (almost 28%), it was made right after mentioning its solution. Inner details of the solution to the reduced-to problem were referred to in only three (approximately 6%) of the solutions. So it seems that for these two problems, in addition to students being more aware of the reductive nature of their solutions, they also worked at a higher level of abstraction.

Table VII. Summary of Findings by Groups

Group of questions	**I**	**II**	**III**[15]	**IV**	**V**	**VI**	**Total**[16]
N (solutions)	346	123	67	147	32	23	738
Direct solution	60 (17.3%)	4 (3.3%)	1 (1.5%)	10 (6.8%)	9 (28.1%)	-	84 (11.4%)
Reductive solution	286 (82.7%)	119 (96.7%)	66 (98.5%)	137 (93.2%)	23 (71.9%)	23 (100%)	654 (88.6%)
Explicit mentioning of the word reduction							
in terms of problems	6 (2.1%)	1 (0.8%)	8 (12.1%)	11 (8.0%)	-	-	26 (4.0%)
unspecified	7 (2.4%)	1 (0.8%)	5 (7.6%)	3 (2.2%)	-	-	16 (2.4%)
in terms of "a problem"	2 (0.7%)	-	-	-	-	-	2 (0.3%)
in terms of solutions	5 (1.7%)	1 (0.8%)	-	2 (1.5%)	-	-	8 (1.2%)
in terms of inputs	22 (7.7%)	6 (5.0%)	5 (7.6%)	3 (2.2%)	1 (4.3%)	-	37 (5.7%)
none	244 (85.3%)	110 (92.4%)	48 (72.7%)	118 (86.1%)	22 (95.7%)	23 (100%)	565 (86.4%)
Explicit reference to the reduced-to problem							
before mentioning its solution	30 (10.5%)	40 (33.6%)	30 (62.5%)	89 (65.0%)	not relevant	not relevant	189 (32.0%)
right after mentioning its solution	50 (17.5%)	15 (12.6%)	12 (25%)	23 (16.8%)	not relevant	not relevant	100 (16.9%)
at a later phase	68 (23.8%)	43 (36.1%)	1 (2.1%)	23 (16.8%)	not relevant	not relevant	135 (22.9%)
no reference	138 (48.3%)	21 (17.6%)	5 (10.4%)	2 (1.5%)	not relevant	not relevant	166 (28.1%)
Reference to inside-details of the solution to the reduced-to problem	97 (33.9%)	2 (1.7%)	3 (6.3%)	9 (6.6%)	9 (39.1%)	not relevant	120 (19.6%)
Proof of correctness							
bidirectional	122 (42.7%)	17 (14.3%)	29 (43.9%)	33 (24.1%)	13 (56.5%)	not relevant	214 (33.9%)
only one direction proved	49 (17.1%)	15 (12.6%)	7 (10.6%)	6 (4.4%)	3 (13.0%)	not relevant	80 (12.7%)
nonformal, narrative	42 (14.7%)	56 (47.1%)	17 (25.8%)	92 (67.2%)	7 (30.4%)	not relevant	214 (33.9%)
no proof	73 (25.5%)	31 (26.1%)	13 (19.7%)	6 (4.4%)	-	not relevant	123 (19.5%)

[15]In this column, percentages for rows 10-14 (dealing with level of abstraction) were calculated with respect to the relevant entries only (48 solutions).

[16]In this column, percentages for the last nine rows were calculated with respect to the relevant entries only.

Similarly, a relatively high level of abstraction was observed also for the three questions that could be solved by reduction to the minimum spanning tree problem. The reduced-to problem was mentioned in most (all but two) of the 137 reductive answers, and in 65% of the reductive answers this reference was made before a solution to this problem was mentioned. Inner details of the solution to the reduced-to problem were referred to in nine of the 137 solutions (less than 7%).

For the question that can be solved by reduction to topological sort, nine of the 23 reductive solutions (almost 40%) referred to inner details of the

reduced-to problem. We had difficulties in categorizing the answers by mentioning of the reduced-to problem, since the problem and its standard solution shared the same name.

The last question, E15, and question E11 from group III were quite exceptional, compared with other questions used in this study. In both questions (that were included in the same final exam, thus solved by overlapping groups of students) the students were asked to assume they are given a black-box algorithm (for E11 this was an algorithm that finds maximum flow for special-structured networks, and for E15 this was an algorithm that can square matrices with some special characteristics) and use this algorithm in order to obtain an algorithm that solves a general problem (finds maximum flow in the case of E11, and multiplies two general matrices in the case of E15). Obviously, all 18 and 23 solutions to E11 and E15, respectively, were reductive (yet, as noted above, none of the solutions to E15 mentioned the word reduction).

The reduced-to problem was explicitly mentioned in each of these two questions—the given black box algorithm was defined in terms of the problem it solves—thus the findings relating the category of reference to the reduced-to problem for these two questions are not comparable for those of other questions, and their meaning is even questionable (therefore, the corresponding entries in the tables are marked as *not relevant*). Yet, it is interesting to see, even as anecdotal evidence, that students' solutions differed on these questions with respect to this category. Students' solutions to question E15 were phrased in terms of the reduced-to problem rather than its algorithm. Except for one student, all students, after transforming the input (two general matrices) to obtain a new input, two special matrices, talked about squaring the two new matrices rather than activating the given algorithm on these matrices. In contrast, students' solutions to question E11 were phrased in terms of the given algorithm, rather than the problem it solves. That is, after transforming the input (a general flow network) to obtain a new input, a special-structured network, students talked about activating the given algorithm on the new network, rather than about finding maximum flow in this network. The data body is not sufficient to provide an explanation, but one hypothesis can relate this difference to the complexity of the reduced-to problem. Matrix squaring might seem to be a more atomic operation, easily added to the vocabulary of used operations, without feeling the need to specify their solution. Finding maximum flow, on the other hand, might seem a complex operation which cannot be detached from a solution. This explanation should be examined within a generalizable context, but it is in line with some findings of the previous, qualitative study. On that study we saw that some students felt that hiding the solution inside a black box was illegitimate (cheating) if the reduced-to problem was complex, since then the main part of the solution was not specified. Others felt that hiding simple solutions inside a black box was illegitimate, since they felt they were expected to demonstrate knowledge of such solutions. Thus, the legitimacy of using black boxes seems to be dependant on characteristics of the reduced-to problem, such as complexity.

Due to the special nature of these two questions, we felt that analyzing their solutions with respect to the category of reference to inside-details would

also be incomparable with corresponding findings for other questions, since for these two questions there are no known inside-details to refer to. Therefore the corresponding entries in the tables are marked as *not relevant*. Indeed, as can be expected, none of the solutions to E15 referred to the (unknown) inner details of the given algorithm. This was also the case for almost all the solutions to E11, except for one student, that did refer to inner details. Actually, this student was assuming that the given black-box algorithm was operating in a certain way, and that enabled him to refer to inner details.

As noted above, the rationale behind the last category—the structure of the correctness proof—was that a solution induced by a complete, holistic strategy should include also a proof with a structure that is characteristic of reduction. Here too one can see some differences among the groups of questions. For Group V, 13 of the 23 reductive solutions (56.5%) included correctness proofs based on a bidirectional argument. Almost 43% and about 44% of the correctness proofs in the reductive solutions to questions in Group I and III, respectively, were based on bidirectional mapping. For Group IV, only about 24% of the correctness proofs for reductive solutions were based on a bidirectional argument (whereas about 67% of these proofs were narrative proofs). In Group II, only about 14% of the reductive solutions were proved based on a bidirectional argument. Solutions in Group VI were not analyzed by this category since the proof of correctness essentially consisted of a series of matrix equations.

We could find no evidence of a progressive improvement in students' reductive skills over the semester. Specifically, no significant differences could be found between solutions given to homework questions and solutions given to questions from final exams.

6. DISCUSSION

The findings presented in the previous section were based on a mostly quantitative analysis using hundreds of student solutions to algorithmic questions solvable by reduction. Such an analysis method enjoys some advantages and suffers from some drawbacks. It can be useful in taking a snapshot of a situation or a behavior, in this case the manner in which advanced undergraduate students acknowledge and perform reduction when solving various algorithmic problems. This snapshot can be generalized, due to the big body of data. Through such a quantitative analysis we cannot gain insight into the thinking processes that produced the analyzed solutions. However, these findings, combined with the findings of the previous, qualitative study, depict a consistent picture: undergraduate CS students do not seem to treat reduction as a general problem-solving strategy applicable in many domains for solving different kinds of problems. This result emerged clearly from the previous, qualitative study. It was hinted by this study, since it is consistent with its findings—students neglecting to categorize their solutions as reductive, and omitting important characteristics of reductive solutions. According to the findings of the current study, in many cases, students' reductive solutions are of a low level of abstraction, lacking a clear black-box component. The black box is

corrupted, either by opening it and referring to the inner details of the solution hidden in it, or by failing to work on the problem level and phrasing the answer in terms of solutions. It also seems that students do not fully understand the nature of reduction, failing to relate it to problems.

In order to interpret these results, it is necessary to put them in the right context. A legitimate response to these results would be "It depends on what these students were taught about reduction in the context of the algorithm course, and this was not discussed." We will deal below with the issue of teaching, but as a possible explanation rather than as a restricting context. As already mentioned in the methodology section, we are interested in the final outcomes of an undergraduate CS program. We argue that since reduction is a CS fundamental idea, it should be an effective tool in each CS graduate's toolbox. This is certainly in line with Wing's position [2006] that lists reductive thinking among the characteristics of computational thinking. Our own impression, based on talks with CS faculty in leading CS departments, is that they expect their graduate students to feel comfortable with reduction and use it naturally. In other words, they expect that this skill would be developed during an undergraduate CS program.

For that reason, we aimed at taking a snapshot of an existing situation (again, as discussed in the methodology section, this section was taken at the algorithm course checkpoint rather than on graduation), documenting the results of a standard teaching strategy for a standard algorithms course.

In an article describing the previous, qualitative study [Armoni et al. 2006] we suggested a possible explanation, relating the low level of abstraction in students' solutions to the reported tendency to reduce level of abstraction when learning a new concept [Hazzan 2003]. This is a valid explanation that rests much of the responsibility with the students. But, as discussed in the previous section, students' ways of performing reduction were not uniform, varying among groups of problems. For example, problems reducible to a maximum flow problem were more strongly related to reduction, whereas recognition of reductive solutions as such was not as clear for other groups to varying degrees. We saw that for some groups of problems, such as those reducible to a maximum flow problem or to the minimum spanning tree problem, students worked at a relatively high level of abstraction. For others, such as problems involving shortest paths (even on final exams), the emphasis was on the solution to the reduced-to problem rather than on the problem itself. The portion of correctness proofs based on bidirectional mapping was more satisfying for some questions, such as those involving maximum flow and shortest paths, and lower for others.

So perhaps students' tendency to reduce the level of abstraction was combined here with another factor. Such a factor can be the way that reduction was interleaved into the teaching process of this course. In examining the tutorial notes of institution A (available to students on the course website), one can see a non-uniform treatment of reduction, which in many cases is partial, implicit, or even nonexistent for some parts of the course. Reduction is demonstrated and explicitly mentioned for problems involving shortest paths in graphs and for problems reducible to a maximum flow problem. As

previously mentioned, the textbook by Kleinberg and Tardos, used in institution A, mentions reduction explicitly only in Chapter 7, which deals with maximum flow. Though other problems presented in the tutorials from other domains were also solved by reduction, their solutions were not acknowledged as reductive, and the word reduction was not mentioned in their solutions. At the same time, quite a few problems involving shortest paths were solved by making changes to BFS, and it seems as if this algorithm was given more emphasis. In tutorials, correctness proofs seemed to be more emphasized and more detailed for questions involving shortest paths in nonweighted graphs, perhaps because these were included in the first tutorials, when the need to educate the students to give formal proofs was felt more strongly. In institution B, being a distance-learning institution, the teaching process was mainly governed by a textbook [Cormen et al. 1990] and a teaching guide. None of these mention reduction in the context of the algorithm course. Reduction was mentioned in the course assignment booklet and in the assignment solution handouts, but in a sporadic manner and mainly in the context of shortest path problems.

This inconsistent and incomplete treatment of reduction in this course does not at all mean that this concept is not valued by the teaching staff. We already mentioned that many CS faculty members that we spoke with, some of whom teach the algorithm course, feel that this is an important skill necessary to any computer scientist. These faculty members are also sure that an undergraduate program offers students many opportunities to be exposed to this idea and get used to it. However, our findings suggest that this exposure lacks an overall, systematic treatment, and that such a treatment is essential for the development of reductive thinking. We believe that the manner in which reduction is taught in these two institutions does not differ significantly from the manner it is covered in other CS programs in most schools. This feeling is somewhat supported by anecdotal evidence obtained by surveying course sites of many schools and standard algorithm textbooks.

We therefore feel that these findings lend additional support to the recommendations we gave in a previous article [Armoni et al. 2006] regarding explicit and rational teaching of reductive thinking. Specifically, we strongly believe that reduction should be taught as a fundamental idea. This means a spiral teaching, in different domains and in different intellectual levels. That is, in any course that involves problem solving, some of the demonstrated solutions should be reduction-based, starting as early in the curriculum as possible, even in CS1. As discussed and demonstrated elsewhere [Armoni et al. 2006], algorithmic problems suitable for CS1 can be used for this purpose, and other CS1-concepts such as modularization or decomposition can also be discussed through the prism of reduction.

This does not necessarily imply that CS1 teaching resources (mainly the resource of time) should be dedicated to reduction on the expense of other CS1-issues. Rather, it is a matter of exploiting or creating opportunities, in the regular teaching process of the course, that naturally lend themselves to discussing reduction. As mentioned above, the teaching process of the algorithm course offered many such opportunities, but the instructors failed to take

advantage of some of them, not recognizing explicitly some of their own solutions as reductive. So rather than adding reduction it to the list of issues to be covered during a problem-solving course (CS1 and more advanced courses), while taking out other issues to make room for it, instructors should be aware of the need to relate to reduction throughout the course. This view is consistent with Cuoco, Golenberg, and Mark's call [1996] for organizing mathematics curricula around habits of mind, since it is important "to give students the tools they will need in order to use, understand, and even make mathematics that does not yet exist" [p. 376].

Reduction should be taught under a unifying framework of a solution strategy. Its components and characteristics should be explained and discussed explicitly, and whenever a reductive solution is demonstrated, it should be explicitly acknowledged as such, emphasizing its reductive characteristics, and preferably comparing it to nonreductive or lower-level reductive solutions to the same problem. In addition, reduction should be demonstrated in different situations and in various contexts.

Instructors should remember that what comes naturally to them as experts, may be misleading for novices. A reductive solution that uses a phrase like "Let's reduce it to BFS" may be clear to an expert, and considered normative. Yet, for a novice, with yet not fully developed abstraction skills, such a phrase may result in confusion. Always talking about reduction in terms of problems, rather than solutions, using only the conceptual interface of the solution to the reduced-to problems rather than its inner details seems to be a good teaching practice. The border line is not always very clear, and there is always a grey area, but we feel it is better to avoid it as much as possible to prevent confusion. For example, prefer using the *distance* returned by a black-box shortest path algorithm rather then using a returned value $d(v)$, that seem to be more strongly connected to a specific solution taught in class for such a problem. Such a strategy can have a positive effect, not just on students' reductive skills but on their general skills of abstraction as well.

We are currently studying the effect of a reduction-emphasized teaching strategy in the algorithm course and hope to report the results in the near future.

APPENDIX A – THE QUESTIONS

Question HW1

> Describe an algorithm, as efficient as you can, that given an undirected graph $G = (V, E)$, a vertex $s \in V$, and an edge $e \in E$, notifies whether e is a necessary edge (that is, there exists a vertex $v \in V$, such that all shortest paths from s to v use e). Prove the correctness of your algorithm and analyze its complexity.

Question HW2

> Describe an algorithm, as efficient as you can, that given a directed graph $G = (V, E)$, a vertex $s \in V$ and an edge $e \in E$, notifies whether

e is an important edge (that is, there exists a vertex $v \in V$ such that there is a shortest path from s to v that uses e). Prove the correctness of your algorithm and analyze its complexity.

(This question was a third item in a three-item question. In the first two items students were asked to prove that every subpath of a shortest path is also a shortest path, and that every shortest path is necessarily simple. Then the students were instructed to use the first two items when solving item three.)

Question HW3

Describe an algorithm, as efficient as you can, that given a directed graph $G = (V, E)$ and a vertex $s \in V$, finds for each vertex $v \in V$ the length of the shortest directed circle that includes s and v. Prove the correctness of your algorithm and analyze its complexity.

Question HW4

Let $G = (V, E)$ be a directed graph, c: $E \rightarrow R^+$ a non-negative capacity function on the edges, $s \in V$ a source vertex, and $t \in V$ a target vertex. Describe an algorithm that finds a minimum flow on the given graph. That is, a flow that minimizes $\sum_{e \in in(t)} f(e) - \sum_{e \in out(t)} f(e)$. Prove the correctness of your algorithm and analyze its complexity.

Question HW5

Let $G = (V, E)$ be an undirected and connected graph and w: $V \rightarrow R$ a function that maps every vertex in G to a cost (a real number). Given a spanning tree T of G, let the cost $w(T)$ of T be $w(T) = \sum_{v \in V} w(v) \cdot d_T(v)$, where $d_T(v)$ is the degree of v in T. Describe an algorithm, as efficient as you can, that finds a spanning tree in G with a minimum cost. Prove the correctness of your algorithm and analyze its complexity.

Question HW6

We analyzed solutions to item three of the following question:
Let $G = (V, E)$ be an undirected and connected graph with:

- A weight function $\ell : E \rightarrow R$
- A set of special vertices $M \subset V$

Let τ be the set of all spanning trees of G, in which all vertices of M are leaves. Assuming $\tau \neq \emptyset$, we wish to find a tree T_{opt} that its weight is minimal, with respect to all trees in τ.

(1) Prove that if $T \in \tau$ then there is no edge (u, v) in T such that $\{u, v\} \subseteq M$.
(2) Let G' be a subgraph of G, obtained by deleting all vertices of M and their adjacent edges. Prove that the subtree T'_{opt}, obtained

from T_{opt} by deleting all vertices of M and their adjacent edges, is a minimum spanning tree of G'.

(3) Describe an algorithm, as efficient as you can, that finds a tree T_{opt} as described above. Prove the correctness of your algorithm.

Question E1 and Question E2

Question E1 and Question E2 were items b and c of the following question, respectively:

Definition:
A directed graph $G=(V, E)$ is a *tricolor* graph if a function f: $V \rightarrow \{A, B, C\}$ is defined on its vertices. A vertex $v \in V$ is *good* if there exists a path p from v to some vertex $u \in V$, such that $f(u) = A$ and for every $w \in p$ $f(w) \neq C$.

a. Show that given a tricolor graph $G=(V, E)$, every vertex $v \in V$ such that $f(v) = A$ is good, and every vertex $v \in V$ such that $f(v) = C$ is not good.
b. Describe a linear-time algorithm that given a tricolor graph $G=(V, E)$ and a vertex $v \in V$ decides whether v is good. Prove the correctness of your algorithm and analyze its complexity.
c. Describe a linear-time algorithm that given a tricolor graph $G=(V, E)$ outputs all the good vertices in G. Prove the correctness of your algorithm and analyze its complexity.

For solving items b and c you can use reduction. That is, you can use a known algorithm as a black box after making appropriate changes to the structure of the given graph.

Question E3

Describe an algorithm as efficient as you can that given an undirected and connected graph $G=(V, E)$, in which every edge is colored by A, B or C, and given two vertices s, $t \in V$, finds a path between s and t that starts with a B-colored edge, ends with a C-colored edge, and the number of its A-colored edges is minimal, with respect to all paths between s and t that start with a B-colored edge and ends with a C-colored edge. Prove the correctness of your algorithm and analyze its complexity.

Question E4

Describe an algorithm, as efficient as you can, that given a directed acyclic graph $G=(V, E)$ with a weight function w: $E \rightarrow R$, and a vertex $s \in V$, finds for each $v \in V$ the weight of the heaviest path from s to v in G. Prove the correctness of your algorithm and analyze its complexity.

Question E5 and Question E6

Question E5 and question E6 were items a and b of the following question, respectively:

Definition:

An undirected graph $G=(V, E)$ is a *layer graph* if $V = V_1 \cup V_2 \cup ... \cup V_k$ such that $k \geq 2$, for every $1 \leq i \neq j \leq k$ $V_i \cap V_j = \phi$, and for every edge $e=(u, v)$, $u \in V_i$, $v \in V_j$ and $|i-j| = 1 (1 \leq i \neq j \leq k)$. Note that a bipartite graph is a layer graph for $k=2$.

Given a layer graph $G = \left(\bigcup_{i=1}^{k} V_i, E\right)$ let n_i $(1 \leq i \leq k)$ be the number of vertices in V_i and for every $1 \leq i \leq k$ let $v_1^i, v_2^i, \ldots, v_{n_i}^i$ be a given ordering of the vertices of V_i.

<at this point a graphical example of a layer graph was given>

Given a layer graph $G = \left(\bigcup_{i=1}^{k} V_i, E\right)$, a weight function w: $V \rightarrow R^+$ is defined on its vertices as follows: For a vertex v_j^i in layer i $(1 \leq i \leq k$, $1 \leq j \leq n_i)$ $w\left(v_j^i\right) = j$. For a path p in G, the weight of p is the sum of the weight of the vertices in p.

a. Design an algorithm as efficient as you can that given a layer graph $G = \left(\bigcup_{i=1}^{k} V_i, E\right)$ finds the length of a shortest path between V_1 to V_k in G (that is, between the first layer and the last layer). Prove the correctness of your algorithm and analyze its complexity.

b. A path in a layer graph $G = \left(\bigcup_{i=1}^{k} V_i, E\right)$ is good if it contains at most one vertex v_j^i $(1 \leq i \leq k)$ such that $j \geq 3$. Design an algorithm, as efficient as you can, that given a layer graph $G = \left(\bigcup_{i=1}^{k} V_i, E\right)$ finds the length of a shortest path of all good paths between V_1 and V_k in G (that is, between the first layer and the last layer). Prove the correctness of your algorithm and analyze its complexity.

Question E7

Describe an algorithm that given a directed graph $G=(V, E)$, two edges $e_1, e_2 \in E$, two vertices $u, v \in V$, and an integer k, finds the number of all paths in G from u to v of length k, that pass first through e_1 and then through e_2. Assume that u, v and the four vertices that define e_1 and e_2 are all distinct. Prove the correctness of your algorithm and analyze its complexity.

Question E8

Describe an algorithm that given a directed graph $G=(V, E)$, an integer k, two vertices $x, y \in V$, and two edges $e_1, e_2 \in E$ such that

$e_1 = (u, v)$ and $e_2 = (v, w)$ ($u, v, w \in V$), finds the number of all paths in G from x to y of length k, that do not contain the edges e_1 and e_2 one immediately after the other (that is, do not contain the subpath (u, v, w)). Assume that x, y, u, v and w are all distinct. Prove the correctness of your algorithm and analyze its complexity.
Hint: You may want to perform a reduction by translating the given graph to a new graph G'.

Question E9

Describe an algorithm, as efficient as you can, that given a directed graph $G = (V, E)$, an edge $e \in E$, a vertex $v \in V$ and a positive integer k, finds the number of all paths in G of length k, in which e is the one before the last edge, and v appears (for the first time) as the second vertex. Prove the correctness of your algorithm and analyze its complexity.

Question E10

Describe an algorithm, as efficient as you can, that given a directed graph $G = (V, E)$, an edge $e \in E$, and a positive integer k, finds the number of all paths in G of length k, that pass through e exactly twice. Prove the correctness of your algorithm and analyze its complexity.

Question E11

Suppose you are given an algorithm that can solve the maximum flow problem for networks in which the out-degree of each vertex is at most 2. Show how you can use this algorithm to solve the maximum flow problem for general networks. Prove the correctness of your algorithm, and analyze its complexity, as a function of the complexity of the given algorithm.

Question E12

In a certain college department there are n courses. To be entitled to a bachelor of science degree each student should fulfill a few requirements. Each requirement is of the form "One should take at least k courses of the subset S of courses." A certain course cannot be used to fulfill more than one requirement. For example, if one requirement is that one should take at least two courses of the set $\{A, B, C\}$, and another requirement is that one should take at least two courses of the set $\{C, D, E\}$, then a student who took the courses $\{A, C, D\}$ is not entitled to a bachelor of science degree.
Design an algorithm, as efficient as you can, that given a list of requirements $r_1, \ldots, r_m$, where each requirement r_i is of the form "One should take at least k_i courses of the subset S_i", and given a list L of courses taken by a certain student, decides whether the

student is entitled to a B.Sc. degree. Prove the correctness of your algorithm and analyze its complexity.

Question E13

Describe an algorithm, as efficient as you can, that given an undirected and connected graph $G = (V, E)$ with a weight function w: $E \to R^+$, and a vertex $u \in V$, decides whether there exists a minimum spanning tree of G in which u is a leaf. Prove the correctness of your algorithm and analyze its complexity.

Question E14

Describe an algorithm, as efficient as you can, that given a directed acyclic graph $G = (V, E)$ decides whether there exists a path in G that passes through all vertices of G, exactly once through each vertex. Prove the correctness of your algorithm and analyze its complexity.

Question E15

Definition:

A half-unit matrix of order n is a matrix of the form $\begin{pmatrix} I & 0 \\ A & B \end{pmatrix}$, where A, B, I and 0 are of order $n/2$.

Show that if a half-unit matrix of order n can be squared in time $S(n)$, then two general order n matrices can be multiplied in time $O(S(2n))$.

ACKNOWLEDGMENTS

Many thanks to Judith Gal-Ezer for most helpful comments and fruitful discussions, and to Yuval Ishai and Dudu Amzallag from the Department of Computer Science of the Technion for their cooperation in the different stages of this research.

REFERENCES

ARMONI, M. AND GAL-EZER, J. 2006. Reduction—An abstract thinking pattern: The case of the computational models course. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'06)*, D. Baldwin, P. Tymann, S. Haller, and I. Russell, Eds., New York: ACM Press, 389–393.

ARMONI, M., GAL-EZER, J., AND HAZZAN, O. 2006. Reductive thinking in computer science. *Comput. Sci. Educ. 16*, 4, 281–301.

ARMONI, M., GAL-EZER, J., AND TIROSH, D. 2005. Solving problems reductively. *J. Educ. Comput. Res. 32*, 2, 113–129.

ARMONI, M. AND GINAT, D. 2008. Reversing: A fundamental idea in computer science education. *Comput. Sci. Educ. 18*, 3, 213–230.

BRUNER, J. S. 1960. *The Process of Education*. Harvard University Press, Cambridge, MA.

BUCHBERGER, B. 1990. Should students learn integration rules? *SIGSAM Bull. 24*, 1, 10–17.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. The MIT Press, Cambridge, MA and McGraw-Hill, Boston, MA.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms 2nd Ed*. The MIT Press, Cambridge, MA and McGraw-Hill, Boston, MA.

CUOCO, A., GOLDENBERG, E. P., AND MARK, J. 1996. Habits of mind: An organizing principle for mathematics curricula. *J. Math. Behav. 15*, 4, 375–402.

CORDER, C. 1990. *Teaching Hard Teaching Soft: A Structured Approach to Planning and Running Effective Training Courses*. Gower, Aldershot, UK.

GLASER, B. AND STRAUSS, A. 1975. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago, IL.

DICHEVA, D. AND CLOSE, J. 1996. Mental models of recursion. *J. Educ. Comput. Res. 14*, 1, 1–23.

HABERMAN, B. AND BEN-DAVID KOLIKANT, Y. 2001. Activating "black boxes" instead of opening "zippers"—A method for teaching novices basic CS concepts. In *Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'01)*. S. Fincher, B. Klein, F. Culwin, and M. McCracken, Eds. New York: ACM Press, 41–44.

HABERMAN, B., SHAPIRO, E., AND SCHERZ, Z. 2002. Are black boxes transparent? High school students' strategies of using abstract data types. *J. Educ. Comput. Res. 27*, 4, 411–436.

HAZZAN, O. 2003. How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Comput. Sci. Educ. 13*, 2, 95–122.

HAZZAN, O., DUBINSKY, Y., EIDELMAN, L., SAKHNINI, V., AND TEIF, M. 2006. Qualitative research in computer science education. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'06)*. D. Baldwin, P. Tymann, S. Haller, and I. Russell, Eds. New York: ACM Press, 408–412.

HERSHKOWITZ, R., SCHWARZ, B. B., AND DREYFUS, T. 2001. Abstraction in context: Epistemic actions. *J. Res. Math. Educ. 32*, 195–222.

KAHNEY, H. 1983. What do novice programmers know about recursion? In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'83)*, R. N. Smith, R. W. Pew, and A. Junda, Eds., New York: ACM Press, 235–239.

KLEINBERG, J. AND TARDOS, É. 2006. *Algorithm Design*. Addison-Wesley, Boston, MA.

LEVI, D. 2001. Insights and conflicts in discussing recursion: A case study. *Comput. Sci. Educ. 11*, 4, 305–322.

LIN, F., LEE, Y., AND WU YU, J. 2003. Students' understanding of proof by contradiction. In *Proceedings of the 27th Annual Meeting of the International Group for the Psychology of Mathematics Education (PME'03)*, N. A. Pateman, B. J. Dougherty, and J. Zilliox, Eds. IV, 443–450.

MCKENNA, P. 2004. Gender and black boxes in the programming curriculum. *J. Educ. Res. Comput. 4*, 1.

OR-BACH, R. AND LAVY, I. 2004. Cognitive activities of abstraction in object orientation: an empirical study. *SIGCSE Bull. 36*, 2, 82–86.

PERRENET, J., GROOTE, J. F., AND KAASENBROOD, E. 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 (ITiCSE'05)*. J. Cunha, W. Fleischman, V. Proulx, and J. LourenÇo, Eds., New York: ACM Press, 64–68.

POLYA G. 1957. *How to Solve it 2nd Ed*. Princeton University Press, Princeton, NJ.

SANDERS, K. AND THOMAS, L. 2007. Checklists for grading object-oriented CS1 programs: Concepts and misconceptions. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'07)*. J. Hughes, R. Peiris, and P. Tymann, Eds. New York: ACM Press, 166–170.

SCHOENFELD, A. H. 1985. *Mathematical Problem Solving*. Academic Press, Orlando, FL.

SCHWILL, A. 1994. Fundamental ideas of computer science. *Bull. Euro. Assoc. Theoret. Comput. Sci. 53*, 274–295.

STOLIN, Y. AND HAZZAN, O. 2007. Students' understanding of computer science soft ideas: The case of programming paradigm. *SIGCSE Bull. 39*, 2, 65–69.

THOMPSON, D. R. 1996. Learning and teaching indirect proof. *Math. Teacher 89*, 6, 474–482.

TURKLE, S. AND PAPPERT, S. 1990. Epistemological pluralism: Styles and voices within the computer culture. *Signs: J. Women in Culture Soc. 16*, 1, 128–157.

TURNER, S. A., QUINTANA-CASTILLO, R., PÉREZ-QUIÑONES, M. A., AND EDWARDS, S. H. 2008. Misunderstandings about object-oriented design: Experiences using code reviews. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (CSE'08)*. J. D. Dougherty, S. Rodger, S. Fitzgerald, and M. Guzdial, Eds., New York: ACM Press, 97–101.

WING, J. M. 2006. Computational Thinking. *Comm. ACM 49*, 3, 33–35.

Received April 2008; revised August 2008; accepted September 2008