Constructivism in Computer Science Education*

[*Journal of Computers in Mathematics and Science Teaching*, in press.]

Mordechai Ben-Ari
Department of Science Teaching, Weizmann Institute of Science

Constructivism is a theory of learning which claims that students construct knowledge rather than merely receive and store knowledge transmitted by the teacher. Constructivism has been extremely influential in science and mathematics education, but much less so in computer science education (CSE). This paper surveys constructivism in the context of CSE, and shows how the theory can supply a theoretical basis for debating issues and evaluating proposals. An analysis of constructivism in computer science education leads to two claims: (1) students do not have an effective model of a computer, and (2) computers form an accessible ontological reality. The conclusions from these claims are that: (1) models must be explicitly taught, (2) models must be taught before abstractions, and (3) the seductive reality of the computer must not be allowed to supplant construction of models.

Introduction

The dominant theory of learning today is called *constructivism*. This theory claims that knowledge is actively constructed by the student, not passively absorbed from textbooks and lectures. Since the construction builds recursively on knowledge that the student already has, each student will construct an idiosyncratic version of knowledge. To the extent that such knowledge is not identical with 'standard' scientific knowledge, the student is said to have *misconceptions*. Teaching techniques derived from the theory of constructivism are supposed to be more successful than traditional techniques, because they explicitly address the inevitable process of knowledge construction.

Constructivism has been intensively studied by researchers of science education (Glynn, Yeany, & Britton, 1991) and mathematics education (Davis, Maher, & Noddings, 1990), to the extent that "radical constructivism represents the state of the art in epistemological theories for mathematics and science education" (Ernest, 1995, p. 475). However, there has been much less work on constructivism in computer science education.

This article is logically divided into two parts. The first part—after a motivating example—is a survey of the theory of constructivism and its application in science education. The second part of the paper contains my analysis of the theory in the context of computer science and my attempts to apply the theory to issues that are of current interest in CSE.

The discussion will be concentrated within the framework of novice programmers, but constructivist principles are applicable at all levels of computer science education. Given the rapid rate of change of software

*This article is an extended version of a paper was presented at the *Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, Atlanta, GA, 1998.

للاستشارات المنارة
www.manaraa.com

tools and applications, most software engineers in industry and business are continually engaged in education: not only in formal training sessions, but also—perhaps more importantly—in the development of manuals, interfaces and help files. They will find the theory and its applications to be both thought-provoking and relevant to their day-to-day work.

Computer science education (though not perhaps theoretical computer science) probably has more in common with engineering education than with science education. Readers with a background in engineering are invited to speculate about the applicability of my analyses to their fields.

Previous work

There is a large literature on the psychology of programming (Hoc, Green, Samurçay, & Gilmore, 1990; Soloway & Spohrer, 1989; Mayer, 1988); in particular, researchers interested in teaching programming to children or to non-majors are often cognitive psychologists deeply immersed in Piagetian principles. Occasionally, these researchers explicitly acknowledge their commitment to constructivist principles (diSessa, Abelson, & Ploger, 1991, p. 12).

The literature on constructivism in computer science education is in no way comparable with the vast literature in mathematics and physics education. Even today, a search of 'constructivism' in the ACM Digital Library returns only a handful of papers. While many computer science educators have been influenced by constructivism, only recently has this been *explicitly* discussed in published work (Boyle, 1996; Brandt, 1997; Gray, Boyle, & Smith, 1998; Hadjerrouit, 1998).

Motivation

WYSIWYG (What You See Is What You Get) word processors are considered to be the epitome of user-friendliness, because working with them is supposed to be exactly analogous to writing with pen or pencil on a sheet of paper—a routine familiar to everyone who has graduated from elementary school. But consider the following scenario. You type in the title of your term paper, select the text and request boldface font. Unfortunately, as you begin to type the text of the paper, it is also displayed in boldface font! Your pre-existing knowledge of a WYSIWYG word processor is almost certainly the metaphor of ordinary writing which consists of placing blobs of ink sequentially, but arbitrarily, on a sheet of paper (Figure 1). This metaphor cannot furnish an explanation for the phenomenon you have encountered, so you become frustrated, anxious and lose self-confidence.

(Place Figure 1 here.)

Of course, the explanation is trivial: the word processor is not storing blobs of ink, but symbols including implicit symbols for font changes and for indicating the end of a line (Figure 2). (Here we are arbitrarily using HTML notation: ... to delimit boldface font and <br> to indicate a line break.) If your selection of the text fragment to change to boldface included an invisible(!) line break character, text typed before the line break will be mysteriously displayed in boldface.

(Place Figure 2 here.)

The correct explanation of WYSIWYG should now be clear. What you *get* is: (1) a data structure for storing text and formatting specifications, and (2) a set of operations on that data structure. What you *see* is: (1) a rendering of the data structure on the screen, and (2) icons and menus to invoke the operations. To *learn* how to use the word processor, you must: (1) create a mental model of the data structure and the effect of each operation, and (2) attribute to each icon and menu item a meaning as an operation.

Constructivism claims each individual *necessarily* creates cognitive structures (models) when learning to use the word processor. Furthermore, it claims that each individual will perform the construction differently, depending on his or her pre-existing knowledge, learning style and personality traits. Hopefully, the

www.manaraa.com

construction is *viable* and the user can successfully use the word processor. Unfortunately, but perhaps inevitably, many users construct non-viable models.

Teaching *how* to do a task can be successful initially, but eventually this knowledge will not be sufficient. As the example tries to show, a student who only knows the procedure for changing from ordinary to boldface font will be helpless when faced with this novel situation. The problem is caused not by stupidity on the part of the novice, nor by incorrectly following the instructions, but by a misconception that is attributable to the lack of a viable model that can explain the behavior of the word processor. The teacher must guide the student in the construction of a viable model so that new situations can be interpreted in terms of the model and correct responses formulated.

The word-processor example illustrates two aspects of learning that are characteristic of computer science. First, since computer science deals with artifacts—programming languages and software, the creator of the artifact employed a very detailed model and the learner must construct a similar, though not necessarily identical, model. Second, knowledge is not open to social negotiation. Given that the word processor is an extant artifact, you cannot argue that its method of using fonts is incorrect, discriminatory, demeaning, or whatever. You may be able to choose another software package, or to request modifications in an existing one, but meanwhile you must learn the existing reality. These two points will be extensively discussed in the rest of the paper.

Epistemology and Constructivism

Educational paradigms

An educational paradigm is composed of four components (Ernest, 1995):

- An *ontology* which is a theory of existence.
- An *epistemology* which is a theory of knowledge, both of knowledge specific to an individual and of shared human knowledge.
- A *methodology* for acquiring and validating knowledge.
- A *pedagogy* which is a theory of teaching.

(See Scheffler (1965) for an introduction to epistemology in the framework of education. Scheffler gives a slightly different decomposition; in particular, he includes *evaluation*: deciding what knowledge is reliable or important.)

We can use this framework to succinctly describe the classical educational paradigm:

- There *is* an ontological reality. Even though scientists accept the theories of relativity and quantum mechanics, the Newtonian model of absolute space and time is the model we generally use for reality. Furthermore, we function as Platonist mathematicians who hold that mathematics has an existence independent of ourselves in which $2 + 2 = 4$ is absolutely true.
- Epistemology is *foundational*. The truth is out there. We come to believe foundations—necessary truths such as $2 + 2 = 4$ and empirical sensory data—and then use valid forms of logical deduction to expand the extent of true knowledge.
- The mind is a *clean slate* that can be filled with knowledge. Once you know enough facts and rules of inference, you can create new knowledge by logical deduction. Carroll (1990) cites the legend of the Nurnberg Funnel which can be used to 'pour' knowledge directly into the learner's head.

www.manaraa.com

- Listening to lectures and reading books are the primary means of *knowledge transmission*. Repetition (drill and practice) will ensure that the knowledge is retained.

The constructivist paradigm is dramatically different:

- Ontological reality is either rejected or at best considered irrelevant. Since we can never truly 'know' anything, ontology cannot influence our educational paradigm.
- The epistemology of constructivism is *nonfoundationalist* and *fallible*. Absolute truth is unattainable, so there is no foundation of truth on which to build. Even $2 + 2 = 4$ is not a necessary truth (Barnes, Bloor, & Henry, 1996, Chapter 7)! Knowledge is constructed by each individual and thus necessarily fallible.
- Knowledge is acquired *recursively*: sensory data is combined with existing knowledge to create new cognitive structures, which are in turn the basis for further construction. Knowledge is also created cognitively by reflecting on existing knowledge. These concepts come from the seminal work of Jean Piaget on the acquisition of knowledge by children; Piaget's work was instrumental in the development of constructivist theories.
- Passive learning will likely fail, because each student brings a different cognitive framework to the classroom, and each will construct new knowledge in a different manner. Learning must be active: the student must construct knowledge assisted by guidance from the teacher and feedback from other students.

Constructivists believe that effective learning demands not just discovery of facts, but the construction of viable mental models, and that teachers must actively *guide* the student in this effort. The task of the teacher in the constructivist paradigm is significantly more difficult than in the classical one, because guidance must be based on the understanding of each student's currently existing cognitive structures.

Note that constructivism does not reject classical means of instruction such as lecturing and reading books. As Mason notes, tongue-in-cheek: "Many educators espousing constructivism have been known to attend lectures on constructivism, and even to have enjoyed them!" (Mason, 1994, p. 197). The problem is not the lecture itself, but the assumption that 'students know what the lecturer told them'. And Mason continues with the suggestion that:

> ...when preparing a lecture, it is the fact of the imminent audience which enables the lecturer to contact the content in fresh ways, in a state conductive to creativity and connection-finding. (Mason, 1994, p. 198)

The concept that the student is trying to construct a model from what are, after all, only words is an appealing theoretical framework for an educator to use in assessing the success or failure of a lecture or other teaching activity.

Conversely, constructivism in not co-extensive with 'modern' teaching methods such as group projects, discovery learning and active tasks. These methods are favored by constructivists *only if* they are designed to enable the students' to build a viable mental model based on pre-existing knowledge. A hands-on activity is useless if "their hands are on, but their heads are out" (Resnick, 1997, p.28).

Constructivism does have a lot in common with *discovery* or *inquiry learning*, where students are expected to discover knowledge by themselves when placed in the appropriate situation. The benefits of discovery are claimed to be:

المنارة للاستشارات
www.manaraa.com

> ... (1) the increase in intellectual potency, (2) the shift from extrinsic to intrinsic rewards, (3) the learning of the heuristics of discovering, and (4) the aid to conserving memory. (Bruner, 1962, p. 83)

Note that Bruner (1962, p. 85) seems to agree with the constructivist viewpoint that unfettered discovery is not helpful; he distinguishes between *episodic empiricism*, where the student accumulates unconnected facts, and *cumulative constructionism*, where the discovery is organized.

Constructivists differ among themselves as to the relative importance ascribed to the individual learner and to the group in constructing knowledge; these variants are known as *radical* and *social* constructivism, respectively. A discussion of the variants of constructivism is beyond the scope of this article; see Ernest (1995), Phillips (1995).

Constructivism in science education

Studies have shown that relatively few students reach an acceptable level of achievement in high-school science and mathematics (Duit, 1991). Physics teachers seem to have the worst time, as students retain a naive theory of physics despite intensive instruction in Newtonian mechanics (McCloskey, 1983). For constructivists this is not surprising: everyone who has ever thrown a ball—that is, everyone—*knows* that if you don't keep applying force, an object in motion will eventually come to rest. Apparently, these ideas are so entrenched that mere lectures and even experiments have a difficult time evicting them. At most, a certain facility in manipulating formulas is achieved, but this fails as soon as the student attempts to solve a problem that requires deep understanding.

The discrepancy between performance and understanding has also been noted in mathematics education:

> The pupil's fundamental problems with such ideas as negative or complex numbers tend to be overlooked by the teacher mainly because the latter's own implicit beliefs make him or her oblivious to the possibility of somebody having a different ontological stance. ... Another circumstance that helps in concealing ontological difficulties is the fact that a student may become quite skilful in manipulating concepts even without reifying them. (Sfard, 1994, p. 268)

Physics educators are very receptive to constructivist principles. After all, physicists have undergone two massive restructurings of their world within a short period of history: from Aristotelian physics to Newtonian physics and then to Einsteinian physics. One cannot fault them for their reluctance to believe that $E = mc^2$ is an absolute truth. This openness is demonstrated by their willingness to attribute to the student *alternative frameworks* rather than misconceptions.

In fact, von Glasersfeld, a pioneer of constructivism, would never say that something is wrong, because he does not believe in the possibility of establishing universal truths. Instead, he says that concepts are viable "if they prove adequate in the contexts in which they were created" (Glasersfeld, 1995, p. 7). This is analogous to the use of the word in biology to denote an organism adapted to its environment. The box metaphor for variables, and the communications model of reference parameters (discussed below) are simply non-viable, because they cause the student to fail on programming tasks.

According to constructivism, a teacher cannot ignore the student's existing knowledge; instead, he or she must question the student in order to understand exactly what theory the student is currently using, and only then attempt to guide the student to the 'correct' theory. It is perhaps axiomatic for a constructivist that students have consistent theories—they just happen to be at variance with the (currently accepted) scientific theory.

In most fields of science education including computer science, there is a large body of research that catalogs *misconceptions*. A constructivist would view a misconception not as a mistake, but as a logical construction

المنارة للاستشارات
www.manaraa.com

based on a consistent, though non-standard theory, held by the student. Even Matthews—who is critical of constructivism—is careful to point out that:

> It is with respect to [contemporary physics] that [students] have misconceptions, it is not with respect to the behavior of the natural world. (Matthews, 1994, p. 133)

Merely listing misconceptions is fruitless; a misconception must be accompanied by a description of the underlying model that caused it, and by a suggestion how to base the construction of a viable model on the existing one. Smith III, diSessa, and Roschelle (1993) go so far as to claim that misconceptions form the prior knowledge that is essential to the construction of new knowledge!

It is important not to confuse the use of computers in science education with the study of computer science. Computers are often seen as a tool to increase the constructive content of science education. For example, Hatfield (1991) considers programming, or more generally algorithmics, as constructive. However, his paper is essentially concerned with the contribution of algorithmics to mathematical education, rather than to the constructivist aspects of computer science and programming. Similarly:

> The role of the computer activities is ... to provide an *experiential basis* for all other learning modes. ... the main point is spending the time and effort on the problem, not solving it. (Leron & Dubinsky, 1995, pp. 231, 236)

In CSE, the computer is not just providing an experiential basis, nor is it creating a microworld (Harel & Papert, 1991) in order to facilitate construction of knowledge in another domain. Instead, the students are learning about computing itself—systems, algorithms, languages—and lessons from the use of computers in other fields must be applied carefully.

Criticism of constructivism

Before continuing, we must stress that that there is strong opposition to constructivism. See the articles by Matthews, Nola, Phillips and Ogborn in the Special Issue on Philosophy and Constructivism in Science Education (January 1997) of the journal *Science & Education*. The articles are also available in Matthews (1998).

One critic writes vehemently:

> If radical constructivism is post-epistemological then it is also pre-Copernican and adopts views of science similar to those of the Inquisition that interviewed Galileo. (Nola, 1997, p. 209)

The criticism is not so much of the constructivist theory of learning, but rather of extreme conclusions drawn from constructivist epistemology:

> The one-step argument from the psychological premise (1) "the mind is active in knowledge acquisition," to the epistemological conclusion (2) "we cannot know reality," is endemic in constructivist writing. (Matthews, 1994, p. 151)

Carried to the extreme, radical constructivism leads to *solipsism*, the philosophical claim that the world is one's own mental creation. In turn, this can lead to a rejection of ethics: if the world is my own creation, why should I care what happens to others? Boyle (1996, Section 6.4) takes radical constructivists to task for putting too much emphasis on an individual's cognition at the expense of the biological (Piaget) and social (Vygotsky) foundations upon which cognition must be based.

Carried to the extreme, social constructivism leads to a view of science as a merely political enterprise developed by entrenched elitist groups whose sole purpose is to ensure their own survival. From the fallibility

المنارة للاستشارات
www.manaraa.com

of scientific knowledge, one slips into relativism of truth, and from the sociology of scientific practice, into demands for empowerment detached from any attempt at objective evaluation of scientific knowledge. The extreme position is stated in the Edinburgh 'strong programme' on the sociology of knowledge (Bloor, 1991; Barnes et al., 1996); for criticism of this position see the articles in Matthews (1998).

The essential question is whether being a constructivist requires an epistemological committment to empiricism and idealism (or social idealism), as opposed to rationalism and realism that seem to come more naturally to scientists. This delicate question can perhaps be avoided by taking the position of 'pedagogical constructivists':

> ...who concentrate solely on pedagogy, and improved classroom practices, ... For [whom], the details of epistemological psychology are unimportant, and not worth disputing about. (Matthews, 1997, p. 8)

Empirical Results in CSE

There is no question that many students find the study of computer science extremely difficult, especially at elementary levels. Before proceeding with a theoretical analysis, it is worthwhile to survey some results that demonstrate the depth of the problem:

- Sleeman, Putnam, Baxter, and Kuspa (1989), Samurçay (1989) and Paz (1996) found that the concept of *variable* is extremely difficult for students. For example, students believe that a variable could simultaneously contain two values, and that after executing `A := B`, the variable `B` no longer contains a value. The students have constructed a *consistent model* using the analogy of a box; the model just happens to be non-viable for successful programming.
- Haberman and Ben-David Kolikant (unpublished research) administered a test designed to check the basic concepts of assignment, read and write statements in Pascal. Given the statements:

  ```
  read(A, B);
  read(B);
  write(A, B, B);
  ```

  many students are not at all sure what happens when you read twice to the same variable or write twice from the variable. They find it difficult to construct a model that identifies 'who' is doing the reading and the writing. Similarly, Samurçay (1989) claims that students' models of `read(A)` may not include the assignment to the variable `A`.
- Madison (1995) used extensive interviews to elicit the internal model of parameters (especially reference parameters) held by students in an introductory course. The students were taught a communications model for parameters, rather than a model of the implementation (copy and reference). The interviews demonstrated that students had constructed consistent, but non-viable, models of the implementation of parameters.
- Similarly, Fleury (1991) discovered 'student-constructed rules' for Pascal parameters that were occasionally successful, but non-viable in the general case.
- Deep misconceptions are not limited to elementary programming. Holland, Griffiths, and Woodman (1997) show the extent of the misconceptions held by students studying object-oriented programming. They found inappropriate conflation of the concept of an object with other concepts like variable, class and textual representation.

المنارة للاستشارات
www.manaraa.com

- The difficulties that students have in elementary computer science studies are often attributed to the need to spend too much time on the syntax of low-level procedural languages like Pascal and C. But similar phenomena are encountered even when teaching Prolog, a language whose syntax is about as simple as can be imagined. Taylor (1990) studied novice Prolog programmers and found that students constructed models that were not viable:

  > Prolog's behavioral component is complex, and because its syntax is noncommittal, learners are tempted to hallucinate onto it whatever they think appropriate, rather than referring to an interpretation based upon underlying domain knowledge. (Taylor, 1990, p. 308)

- Algorithm and software visualization is an extremely active field of CSE research. Yet Mulholland (1997) found that software visualization in itself does not necessarily help the student unless the visualization is based on a careful analysis of the pedagogic task.

Constructivism in the Context of CSE

To what extent is constructivism applicable to CSE? According to constructivism, students construct knowledge by combining the experiential world with existing cognitive structures. I claim that the application of constructivism to CSE must take into account two characteristics that do not appear in natural sciences:

- A (beginning) computer science student has no *effective model* of a computer.
- The computer forms an *accessible ontological reality*.

By *effective* model, I mean a cognitive structure that the student can use to make viable constructions of knowledge based upon sensory experiences such as reading, listening to lectures and working with a computer. By *accessible ontological reality*, I mean that a 'correct' answer is easily accessible, and moreover, successful performance requires that a normative model of this reality must be constructed. The rest of this section expands on these claims.

The important word is effective. The naive theory of physics held by students is clearly effective, as anyone who has seen professional ball players can testify. They have intuitive models that enable them to implicitly calculate the forces required to achieve superb accuracy when throwing or kicking a ball. Note that diSessa (1988) does not believe that students' intuitive concepts form a well-developed theory. Rather, he claims that they have a large number of fragments called *p-prims*, short for *phenomenological primitives*. This does not materially change the argument, as it is doubtful that intuitive knowledge about computers reaches even the level of diSessa's p-prims.

The empirical results cited earlier (especially the work by Taylor (1990)) show just as clearly that intuitive models of computers are doomed to be non-viable. At most, the model is limited to the grossly anthropomorphic giant brain, hardly a useful metaphor when studying computer science. Pea (1986) gives the name 'superbug' to the idea that a 'hidden mind' within the programming language has intelligence.

At the novice level, the claim is supported by many studies:

> Even if no effort is made to present a view of what is going on 'inside' the learners will form their own. (du Boulay, 1989, p. 285)
>
> ... [we] attribute students' fragile knowledge of programming in considerable part to a lack of a mental model of the computer.... (Perkins, Schwartz, & Simmons, 1988, p. 162)
>
> ... even after a full semester of Pascal, students' knowledge of the conceptual machine underlying Pascal can be very fuzzy. (Sleeman, Putnam, Baxter, & Kuspa, 1988, p. 251)

المنارة للاستشارات
www.manaraa.com

The lack of an effective, even if flawed, model of a computer can be a serious obstacle to teaching computer science if we accept the claim by Smith III et al. (1993) that prior knowledge, even in the form of misconceptions, is essential to the construction of new knowledge.

Turning now to the question of ontological reality, the computer science student is faced with immediate and brutal feedback on conclusions drawn from his or her mental model. More graphically, alternative frameworks cause bugs. Computer science is unlike school physics: the consequences of misconceptions are exposed immediately, not when you get your homework back a week later. Similarly, from the social viewpoint, there is not much point negotiating models of the syntax or semantics of a programming language.

This claim is based on the fact that almost all introductory computer science instruction includes programming. If, as Dijkstra (1989) suggested, we taught programs as mathematical objects that need not be executed on a computer, the normal constructivist principles would apply. We could talk about the viability of denotational semantics, or the social processes responsible for the belief in the Church-Turing Thesis. If the latter were ever superseded, we would experience a shock no less intense than that experienced by physicists in the early twentieth-century. Clearly, since computer science is unlikely to become a subject that is primarily theoretical, we must generate the motivation to examine our teaching practices without the benefit of an epistemological shock.

The claim cuts at the heart of constructivist *epistemology*, which is nonfoundationalist and fallible. But the pedagogy of constructivism is relatively independent of its epistemology. A physicist has no way of determining if $E = mc^2$ is true, but few of us can resist the temptation to use a computer if it helps us construct knowledge about a language or system. In fact, one of the ultimate tests of your prowess as a computer programmer or software engineer comes when you have to deal with a bug in the underlying hardware, operating system or language compiler. Since you have come to look upon them as ontological reality—as arbiters of truth so to speak—it is extremely difficult to diagnose a problem in the implementation of your mental model, as opposed to a problem in your personal task such as writing a program.

Application of Constructivism in CSE

Many phenomena of CSE can be explained by constructivism:

- The construction of even elementary computer science concepts is haphazard, leading to frustration and to the perception that computer science is hard. This is due to the fact that—in the absence of a viable pre-existing model—models must be self-constructed from the ground up.
- Autodidactic programming experience is not necessarily correlated with success in academic computer science studies. These students, like most physics students, come with firmly held mental models that are not viable for academic studies.
- Graphical user interfaces (GUI) are often touted as 'intuitive' and 'user-friendly', yet many people earn a comfortable living giving courses to anxiety-ridden users. Icons, scroll bars and menus are merely representations, and seeing a representation alone contributes very little to the construction of a model.
- The reality feedback obtained by working on a computer can be discouraging to students who prefer a more reflective or social style of learning.

In the rest of the paper, I will apply constructivist principles to specific issues in CSE. To avoid misunderstanding, it is important to clarify what is being claimed here. I am not (necessarily) saying that one

www.manaraa.com

approach is superior to another; rather, I am saying that certain conclusions seem to follow directly from constructivist principles, so that if you accept constructivism—which you are not required to do of course—then you must be willing to analyze your teaching methods in light of these conclusions.

GUI and WYSIWYG Angst

Turkle and Papert (1990) wax poetic on the virtues of icons. Yet an icon is just a representation; it is useful only to the extent that the user can *construct* a mental model of object being represented. The icon must undergo *semiosis*: "the process whereby something comes to stand for something else, and thus acquires the status of a sign" (Husén & Postlethwaite, 1994, p. 5411). Today's software packages, both those intended for the general public such as word processors and professional software such as integrated development environments, display dozens of icons. From a semiotic point of view, it may be true that that an icon is better than text, but from a constructivist point of view, what is important is the construction of the model and not the sign that denotes it.

Icons are intuitive to the extent that the analogy between the object shown and the object represented is perfect. But as Glynn (1991) shows, analogies are rarely, if ever, perfect, so one must not lose patience with a novice who has yet to construct a viable model of the underlying machine. For example, consider an icon for the paste operation. The icon is two steps removed from the operation. First, the icon must be deciphered as representing the word paste. (This first step can be skipped if paste is selected from a menu.) Second, the word whose original meaning is 'form a permanent chemical bond between one item and another' must be related to the operation 'insert a copy of the material held in an internal buffer into the current working document at the place pointed to by the cursor'. To understand this operation, you must have a mental model that enables you to understand the four concepts in this sentence. Even if the word 'paste' is avoided, it is hard to see how so many concepts can be contained within an icon.

WYSIWYG (What You See Is What You Get) is another concept that could benefit from constructivist analysis as we showed above. The relevance for CSE is this: courses, help files and tutorials must *explicitly* address the construction of a model, and not limit themselves to behaviorist practices of the form 'to do X, following these steps'. It is a reasonable conjecture that document preparation systems with transparent models like LaTeX and HTML should engender less anxiety among their users than WYSIWYG systems *on complex tasks*. If the underlying model is not accessible, there is a genuine trepidation associated with trying out new or advanced features, for fear that the document will be irrevocably trashed; with a transparent model you can easily insert and then comment-out or remove the explicit commands. Many users of WYSIWYG systems overcome the anxiety and eventually construct viable models, but the anxiety returns as new features are tried or familiar ones used in new contexts. Of course the claims in this paragraph are anecdotal and need empirical verification.

Explicitly Teach the Model

If the student does not bring a preconceived model to class, we must ensure that a viable hierarchy of models is constructed and then refined as learning progresses. This means that the model of a computer—CPU, memory, I/O peripherals—must be *explicitly* taught and discussed, not left to haphazard construction and not glossed over with facile analogies. Furthermore, the choice of language is not arbitrary (as is often claimed) because the "simplicity and visibility of the notional machine can be spoiled by poor language design or implementation" (du Boulay, O'Shea, & Monk, 1989, p. 436).

Teaching the model can be done using diagrams Mayer (1975) or *epistemic games*—formalized procedures for constructing knowledge—such as a model computer (Sherry, 1995) or a notional machine (du Boulay, 1989). Kieras and Bovair (1984) showed that a block diagram of an instrument facilitates the learning of an operational procedure, and Mulholland showed that software visualization (SV) of Prolog programs

www.manaraa.com

is most successful if "there is a clear, simple mapping between the SV and the underlying source code" (Mulholland, 1997). Based on observations of expert programmers and electronics engineers, Petre (1991) believes that declarative reasoning does not really occur; instead, the experts reason operationally in terms of an underlying machine.

An important question is: how detailed should a model be? Does an introductory computer science student have to construct a model in terms of the electronic properties of semiconductors?! The extent and fidelity of the model that must be taught to the students can only be discovered from the experience of teachers of the subject. Sherry's model seems to be too detailed; a better approach is demonstrated by Naps and Stenglein (1996) who created a visualization of a specific concept—parameter passing. Much can be done even with non-computerized epistemic games. For example, take three cheap calculators and attach them to a board (Figure 3), covering up all the non-numeric keys except for `'='`. Each calculator represents one variable and it is possible to practice assignment statements without ever touching a programmable computer.

(Place Figure 3 here.)

Don't Start with Abstractions

My conclusion that a model of the computer be explicitly taught has implications for the teaching of object-oriented programming (OOP) in introductory courses. The abstraction inherent in OOP is essential as a way of forgetting detail, and software development would be impossible without abstraction, but it seems to me that there must be an *object-oriented paradox*: how is it possible to forget detail that you never knew or even imagined? If students find it difficult to construct a viable model of variables and parameters, why should we believe that they can construct a viable model of an object such as a window object? Advocates of an objects-first approach seem to be rejecting Piaget's view that abstraction (or accommodation) *follows* assimilation.

Professional software engineers who use abstractions generally have a fairly good idea of the underlying model. For example, few software engineers have actually written programs for manipulating windows on a screen. But even a general understanding of how images are represented in the computer by bitmaps should be sufficient to enable the engineer to construct a viable model.

I appreciate the attractiveness of an objects-first approach; the gap between the standard libraries (especially the GUI libraries) of a modern programming environment and the model of a computer is so great that motivating beginners has become a serious problem. Furthermore, OOP can be used to teach good software development practice from the beginning because "OOP allows—even encourages—one to address the "big picture" by emphasizing a *strategic* approach to programming" (Decker & Hirshfeld, 1993, p.271).

Turkle and Papert go further and claim that OOP is:

> ... not only more congenial to those who favor concrete approaches, but it also puts an intellectual value on a way of thinking that is resonant with their own. (Turkle & Papert, 1990, p. 155)

This claim is strange, because the point of studying OOP is to learn to *create* abstractions, not just to *use* existing concrete objects. The concreteness of reading and using objects is at most a stepping-stone to modifying, extending and defining them, as advocates of OOP are careful to point out (Decker & Hirshfeld, 1993).

Given these advantages of the objects-first approach, it cannot be dismissed out of hand; on the contrary, the trade-offs probably favor this approach. But if the constructivist viewpoint is valid, teachers of introductory courses that use OOP should be very, very careful not to assume that the students will construct the model that the instructor has, nor even to assume that they will construct a viable model at all.

المنارة للاستشارات
www.manaraa.com

This viewpoint is supported by the literature on teaching OOP:

- While Adams (1996) opposes deferring the teaching of OOP until late in the curriculum by which time it is difficult to cure students of the low-level paradigms they have developed, neither does he believe that OOP should be taught first when the students are not mature enough to master the *concepts* involved:

  > CS1 novices do not have the cognitive framework to grasp the concepts underlying object-oriented design, because they have no experience dealing with types and functions, much less classes, function members or inheritance. (Adams, 1996, p.79)

  He advocates a middle road where objects are introduced early but only after sufficient procedural programming has been learned to provide an underlying mental model.

- Wolz and Conjura (1994) propose a three-tiered model for teaching introductory computer science which includes mathematical theory (unusual but refreshing!), implementation and mechanical trivia. They report that teaching OOP using C++ in CS2 is successful because students are able to build on previous knowledge learned from CS1: expressing algorithms procedurally in Scheme. On the other hand, they claim that:

  > There is no reason that students in a first course can't learn to use [data types such as queues, stacks, lists, trees and graphs] before learning how they are implemented. (Wolz & Conjura, 1994, p.224)

  From a constructivist point of view, one must evaluate the mental models these students construct; if they are non-viable, they can impede further study.

- Holland et al. (1997) summarize students' misconceptions in an introductory course that uses OOP. Many of these misconceptions are due to conflation of concepts (object/variable, object/class) that can be attributed to the lack of an effective mental model. Based on experience in other disciplines of science education, cataloging and analyzing misconceptions will not be sufficient to improve students' understanding. Instead, research must be done to identify the mental models that cause these specific misconceptions, and guidelines must be developed so that teachers can diagnose and correct the problems.

For an objects-first approach to work, teachers will have to develop ways of explaining the underlying models without destroying the abstractions. My current belief is that introductory CSE should be based on the functional or logic programming paradigm, not only because these languages minimize mechanical trivia, but also (and primarily) because the underlying models can be explained in relatively high-level, hardware-free terms.

Bricolage

Bricolage is a term coined by the anthropologist Claude Lévi-Strauss, who used it in a derogatory sense for the 'science of the concrete' in primitive societies, as opposed to abstract European science. Turkle and Papert (1990) transferred the concept to the context of learning to program, and vehemently defend it as a learning style as valid as the normative 'planning' style that we attempt to teach. This is consistent with a constructivist view of education: different students will approach the construction of knowledge in different ways, and the educational environment must be supportive of these differences.

The manifestation of bricolage in computer science is endless debugging: try it and see what happens. While we all practice a certain amount of bricolage and while concrete thinking can be especially helpful—if not

المنارة للاستشارات
www.manaraa.com

essential—for students in introductory courses, bricolage is not an effective methodology for professional programming, nor an effective epistemology for dealing with the massive amount of detailed knowledge must be constructed and organized in levels of abstraction (cf. object-oriented programming). The normative planning style that we call software engineering must eventually be learned and practiced.

This belief is likely to be shared by anyone who has studied or worked on non-deterministic systems involving concurrency, real-time or communications, subjects that are simply not amenable to bricolage and can be mastered only through abstract techniques. Students who excel at bricolage often cannot make the transition to master the thought patterns and methods required by these systems. This claim has implications for counselling students. If software development is ultimately about abstraction, a students incapable of or uncomfortable with abstract thought should be discouraged from studying for the profession of software engineer.

Gender

Turkle and Papert (1990) published their article arguing for tolerance of concrete thinking in a journal subtitled *Women in Culture and Society*, and they chose two women to exemplify college students who are concrete thinkers. Since the concrete way of thinking advocated by Turkle and Papert can only go so far in computer science, their coupling of a learning style with a gender stereotype would lead to the unacceptable conclusion that women are not suited for careers as computer scientists.

On the other hand, constructivism—especially social constructivism—has much to say about the task of the teacher and the role of peers in education, and the theory can contribute to the analysis of the well-documented social difficulties faced by women in the computer science classroom and laboratory.

Minimalism

Minimalism (Carroll, 1990, 1998) is an approach to instruction that arose in the design of manuals for software documentation. It is apparently little known outside of this community. (For a good introduction see Van der Meij and Carroll (1998).) The minimalist approach to training and documentation can be summarized as follows:

> . . . (1) allowing learners to start immediately on meaningful realistic tasks, (2) reducing the amount of reading and other passive activity in training, and (3) helping to make errors and error recovery less traumatic and more pedagogically productive. (Carroll, 1990, p. 7)

Minimalism has much in common with constructivism as explicitly noted by Van der Meij (1992, p. 7) and Carroll and Van der Meij (1998, p. 84):

- A preference for active learning to enable the student to construct mental models.
- Recognition of the importance of pre-existing knowledge.
- The employment of the inevitable errors and misconceptions as a pedagogical device rather than as a symptom of failure.

Minimalism seems to part company with constructivism in its emphasis—even insistence—on eliminating conceptual material, or at least on deferring it as long as possible:

> It is quite common for training manuals to present a “welcome to the system” preface, a conceptual model of how the system works, ... And none of this, even in the end, does much to facilitate the user’s desire to get started on meaningful activity. Rather, it obstructs this goal. (Carroll, 1990, p. 80)

المنارة للاستشارات
www.manaraa.com

The success of minimalism has been empirically demonstrated in straightforward training tasks like learning to use a word processor. But once the user needs to go beyond elementary tasks, the absence of a viable mental model means that the user's attempts to master advanced material will be frustrating and lead to a reluctance to learn new concepts.

To test this conjecture, I performed an experiment which required the subjects to modify documents in Microsoft Word (Ben-Ari, 1999). The tasks were chosen to be easy if you understand the underlying *concepts*, but quite difficult if you do not. The (sophisticated) subjects almost invariably used bricolage. They restricted themselves to elementary techniques learned in a minimalist setting—behaviorist explanations from colleagues—and made no attempt to investigate the concepts or even to use the Help facility.

Some authors now claim that the dismissal of conceptual material by naive minimalism was mistaken and some way must be found to strike a balance. See the articles by Rosenbaum, Hackos, Redish, Farkas and Draper in the retrospective volume by Carroll (1998). For example:

> ... a manual must: Help users grasp the big picture of the product, that is, help users develop a mental model that helps them predict what to do. (Redish, 1998, p. 240).

Given the empirically proven success of minimalism in the narrow field of technical documentation, it would be interesting to explore a closer integration of minimalist writing techniques with constructivist teaching techniques.

Don't run to the computer

Constructivism suggests that programming exercises should be delayed until class discussion has enabled the construction of a good model of the computer. Too often students become infatuated with the absolute ontology supplied by the computer. Premature attempts to write programs lead to bricolage and delay the development of viable models. While formal methods in computer science education are extremely important, you need not go to the extreme that Dijkstra advocates and entirely give up compilation and execution of programs. There is nothing wrong with experimentation and bricolage-style debugging, as long as it supplements, rather than supplants, planning and formal methods.

Unfortunately, computer science education is heavily weighted on the side of bricolage. A high-school course we are developing comes in for scathing criticism from many students (and some teachers!) because we insist on 'wasting time' on algorithm development and analysis, instead of just getting on with writing and debugging programs.

Laboratory organization

One of the debates in CSE concerns the choice between closed labs—where students work on assignments at an appointed time in a supervised setting, and open labs—where students work on assignments whenever convenient. From a constructivist viewpoint, especially from a social constructivist one, closed labs should be preferable, not only because they soften the brutality of the interaction with the computer, but also because they facilitate the social interaction that is apparently necessary for successful construction. In fact, Thweatt (1994) found empirical evidence for the superiority of closed labs over open labs.

The type of problems assigned is also important; as opposed to minimalism's emphasis on task performance, problems should encourage cognitive operations such as reflection and exploration:

> Another common failing in lab design is to make every task so constrained and explicit that students never need to think about what techniques to use. ... The production of an ill-structured problem is likely to add an element of reality to the lab, and allows the students to have

المنارة للاستشارات
www.manaraa.com

their own Eureka!s about the underlying nature of the exercise. (Fekete & Greening, 1996, pp. 295, 298)

Assessment

Performance on a test is a poor guide to the students' construction of the rich conceptual models of computer science. A student's failure to construct a viable model is a failure of the educational process, even if the failure is not immediately apparent. Furthermore, in the case of group work, performance-based assessment can mask the misconceptions of individual students (Sleeman et al., 1988). Ideally, constructivist-inspired assessment would be based on an instructor's observation and questioning of students engaged in an unconstrained activity such as a lab project. Unfortunately, this is almost always impractical, and instructors must attempt to designed written questions that elicit information about the student's mental model rather than about the contents of his or her factual memory.

Implications for Research

In their book, Maykut and Morehouse (1994) claim that practitioners of qualitative research must understand its philosophical underpinnings, which are essentially constructivist in nature. The claim can be turned around: a researcher working from a constructivist viewpoint should use qualitative methods.

We are now starting to see more empirical research in CSE done using qualitative methods (Madison, 1995; Mulholland, 1997). These techniques which elicit the internal structures of the student are far more helpful than research that measures performance alone and then draws conclusions on the success of a technique.

As computer literacy becomes common, if not universal, students will begin their academic studies with an effective model of a computer. Research must be done to determine if these models are stepping-stones to the construction of effective models, or obstacles like naive physics.

A Guide for Educators

To summarize the paper, here is a guide for educators on the practical application of constructivism.

- Regardless of your teaching technique (lectures, labs, assignments), you must articulate to yourself the cognitive change that you wish to bring about in the students and structure the activity to achieve this aim. Merely transferring knowledge is not a meaningful aim.
- You must dig underneath your own expert knowledge to expose the prior knowledge needed to construct a viable model of the material that you are teaching. You must ensure that that the students have this prior knowledge.
- In any particular course you will be teaching a specific level of abstraction; you must explicitly present a viable model one level beneath the one you are teaching.
- When a student makes a mistake or otherwise displays a lack of understanding, you must assume that the student has a more-or-less consistent, but non-viable, mental model. Your task as a teacher is to elicit this model and guide the student in its modification.
- You must provide as much opportunity as possible for individual reflection (for example, analysis of errors) and social interaction (for example, group labs).

Clearly, each educator must decide how to apply these aphorisms in a concrete situation.

المنارة للاستشارات
www.manaraa.com

Conclusion

My analysis of constructivism has led me to conclude that the epistemology of computer science is significantly different than that of, say, physics. Nevertheless, the basic tenet of the theory—that knowledge is constructed by the student—applies to computer science, and its central implication is that models must be explicitly taught.

Given the central place of constructivist learning theory and its influence on pedagogy, computer science educators should study the theory, perform research and analyze their educational proposals in terms of constructivism. Software and language designers should be guided by constructivist principles, though the individuality of the construction by learners implies that no system will ever be universally easy-to-learn, and we educators must learn how to teach these extant artifacts.

Acknowledgements

I would like to thank Abraham Arcavi, Yifat Ben-David Kolikant, Tom Boyle, Bat-Sheva Eylon, Ann Fleury, Sandra Madison and the referees for their critiques of drafts of this article.

References

Adams, J. C. (1996). Object-centered design: A five-phase introduction to object-oriented programming in CS 1–2. *SIGCSE Bulletin*, *28*(1), 78–82.

Barnes, B., Bloor, D., & Henry, J. (1996). *Scientific knowledge: A sociological analysis.* Chicago, IL: University of Chicago Press.

Ben-Ari, M. (1999). Bricolage forever! In *Eleventh workshop of the psychology of programming interest group* (pp. 53–57). Leeds, UK.

Bloor, D. (1991). *Knowledge and social imagery (second edition).* Chicago, IL: University of Chicago Press.

Boyle, T. (1996). *Design for multimedia learning.* Hemel Hempstead: Prentice-Hall.

Brandt, D. S. (1997). Constructivism: Teaching for understanding of the Internet. *Communications of the ACM*, *40*(10), 112–117.

Bruner, J. S. (1962). *On knowing: Essays for the left hand.* Cambridge, MA: Harvard University Press.

Carroll, J. M. (1990). *The Nurnberg Funnel: Designing minimalist instruction for practical computer skill.* Cambridge, MA: MIT Press.

Carroll, J. M. (Ed.). (1998). *Minimalism beyond the Nurnberg Funnel.* Cambridge, MA: MIT Press.

Carroll, J. M., & Van der Meij, H. (1998). Ten misconceptions about minimalism. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 55–90). Cambridge, MA: MIT Press.

Davis, R. B., Maher, C. A., & Noddings, N. (Eds.). (1990). *Constructivist views of the teaching and learning of mathematics.* Reston, VA: National Council for the Teaching of Mathematics.

Decker, R., & Hirshfeld, S. (1993). Top-down teaching: Object-oriented programming in CS 1. *SIGCSE Bulletin*, *25*(1), 270–273.

Dijkstra, E. W. (1989). On the cruelty of really teaching computer science. *Communications of the ACM*, *32*(12), 1398–1404.

diSessa, A. A. (1988). Knowledge in pieces. In G. Forman & P. B. Pufall (Eds.), *Constructivism in the computer age* (pp. 49–70). Hillsdale, NJ: Lawrence Erlbaum Associates.

diSessa, A. A., Abelson, H., & Ploger, D. (1991). An overview of Boxer. *Journal of Mathematical Behavior*, *10*, 3–15.

المنارة للاستشارات
www.manaraa.com

du Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 283–299). Hillsdale, NJ: Lawrence Erlbaum Associates.

du Boulay, B., O'Shea, T., & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 431–446). Hillsdale, NJ: Lawrence Erlbaum Associates.

Duit, R. (1991). Students' conceptual frameworks: consequences for learning science. In S. M. Glynn, R. H. Yeany, & B. K. Britton (Eds.), *The psychology of learning science* (pp. 65–85). Hillsdale, NJ: Lawrence Erlbaum Associates.

Ernest, P. (Ed.). (1994). *Constructing mathematical knowledge: Epistemology and mathematics education.* London: The Falmer Press.

Ernest, P. (1995). The one and the many. In L. P. Steffe & J. Gale (Eds.), *Constructivism in education* (pp. 459–486). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fekete, A., & Greening, A. (1996). Designing closed laboratories for a computer science course. *SIGCSE Bulletin*, *28*(1), 295–299.

Fleury, A. E. (1991). Parameter passing: The rules the students construct. *SIGCSE Bulletin*, *23*(1), 283–286.

Glasersfeld, E. von. (1995). A constructivist approach to teaching. In L. P. Steffe & J. Gale (Eds.), *Constructivism in education* (pp. 3–15). Hillsdale, NJ: Lawrence Erlbaum Associates.

Glynn, S. M. (1991). Explaining science concepts: a teaching-with-analogies model. In S. M. Glynn, R. H. Yeany, & B. K. Britton (Eds.), *The psychology of learning science* (pp. 219–240). Hillsdale, NJ: Lawrence Erlbaum Associates.

Glynn, S. M., Yeany, R. H., & Britton, B. K. (Eds.). (1991). *The psychology of learning science.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Gray, J., Boyle, T., & Smith, C. (1998). A constructivist learning environment implemented in Java. *SIGCSE Bulletin*, *30*(3), 94–97.

Hadjerrouit, S. (1998). A constructivist framework for integrating the Java paradigm into the undergraduate curriculum. *SIGCSE Bulletin*, *30*(3), 105–107.

Harel, I., & Papert, S. (Eds.). (1991). *Constructionism.* Norwood, NJ: Ablex.

Hatfield, L. L. (1991). Enhancing school mathematical experience through constructive computing activity. In L. P. Steffe (Ed.), *Epistemological foundations of mathematical experience* (pp. 238–259). New York, NY: Springer-Verlag.

Hoc, J., Green, T., Samurçay, R., & Gilmore, D. (1990). *Psychology of programming.* London: Academic Press.

Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. *SIGCSE Bulletin*, *29*(1), 131–134.

Husén, T., & Postlethwaite, T. N. (Eds.). (1994). *The international encyclopedia of education.* Oxford: Pergamon.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science*, *8*, 255–273.

Leron, U., & Dubinsky, E. (1995). An abstract algebra story. *American Mathematical Monthly*, *102*(3), 227–242.

Madison, S. K. (1995). *A study of college students' construct of parameter passing: Implications for instruction.* Unpublished doctoral dissertation, U. of Wisconsin.

Mason, J. (1994). Enquiry in mathematics and mathematics education. In P. Ernest (Ed.), *Constructing mathematical knowledge: Epistemology and mathematics education* (pp. 190–200). London: The Falmer Press.

Matthews, M. R. (1994). *Science teaching: The role of history and philosophy of science.* New York, NY: Routledge.

Matthews, M. R. (1997). Introductory comments on philosophy and constructivism in science education. *Science & Education*, *6*(1-2), 5–14.

Matthews, M. R. (Ed.). (1998). *Constructivism in science education.* Dordrecht: Kluwer Academic Publishers.

المنارة للاستشارات
www.manaraa.com

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. *Journal of Educational Psychology*, *67*(6), 725–734.

Mayer, R. E. (Ed.). (1988). *Teaching and learning computer programming.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Maykut, P., & Morehouse, R. (1994). *Beginning qualitative research.* London: The Falmer Press.

McCloskey, M. (1983). Naive theories of motion. In D. Gentner & A. L. Stevens (Eds.), *Mental models* (pp. 299–323). Hillsdale, NJ: Lawrence Erlbaum Associates.

Mulholland, P. (1997). *Using a fine-grained comparative evaluation technique to understand and design software visualization tools.* Paper presented at the Empirical Studies of Programmers: Seventh Workshop.

Naps, T. L., & Stenglein, J. (1996). Tools for visual exploration of scope and parameter passing in a programming languages course. *SIGCSE Bulletin*, *28*(1), 295–299.

Nola, R. (1997). Book review of Kenneth Tobin (ed.), The practice of constructivism in science education. *Science & Education*, *6*(1-2), 197–201.

Paz, T. (1996). *Computer science for vocational high-school studuents: Processes of learning and teaching.* Unpublished master's thesis, Technion—Israel Institute of Technology. (in Hebrew)

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, *2*(1), 25–36.

Perkins, D., Schwartz, S., & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 153–178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Petre, M. (1991). *Shifts in reasoning about software and hardware systems: do operational models underpin declarative ones?* Paper presented at the Psychology of Programming Interest Group Workshop.

Phillips, D. (1995). The good, the bad, and the ugly: The many faces of constructivism. *Educational Researcher*, *24*(7), 5–12.

Redish, J. (1998). Minimalism in technical communication: Some issues to consider. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 219–245). Cambridge, MA: MIT Press.

Resnick, M. (1997). *Turtles, termites, and traffic jams: Explorations in massively parallel microworlds.* Cambridge, MA: MIT Press.

Samurçay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 161–178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Scheffler, I. (1965). *Conditions of knowledge: An introduction to epistemology and education.* Chicago, IL: University of Chicago Press.

Sfard, A. (1994). Mathematical practices, anomalies and classroom communications problems. In P. Ernest (Ed.), *Constructing mathematical knowledge: Epistemology and mathematics education* (pp. 248–273). London: The Falmer Press.

Sherry, L. (1995). A model computer simulation as an epistemic game. *SIGCSE Bulletin*, *27*(2), 59–64.

Sleeman, D., Putnam, R. T., Baxter, J. A., & Kuspa, L. (1988). An introductory Pascal class: A case study of student errors. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 237–257). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sleeman, D., Putnam, R. T., Baxter, J. A., & Kuspa, L. (1989). A summary of misconceptions of high school Basic programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 301–314). Hillsdale, NJ: Lawrence Erlbaum Associates.

Smith III, J. P., diSessa, A. A., & Roschelle, J. (1993). Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The Journal of The Learning Sciences*, *3*(2), 115–163.

المنارة للاستشارات
www.manaraa.com

Soloway, E., & Spohrer, J. C. (Eds.). (1989). *Studying the novice programmer.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Steffe, L. P., & Gale, J. (Eds.). (1995). *Constructivism in education.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Taylor, J. (1990). Analysing novices analysing Prolog: What stories do novices tell themselves about Prolog. *Instructional Science*, *19*, 283–309.

Thweatt, M. (1994). CS1 closed lab vs. open lab experiment. *SIGCSE Bulletin*, *26*(1), 80–82.

Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and cultures within the computer culture. *Signs: Journal of Women in Culture and Society*, *16*(1), 128–148.

Van der Meij, H. (1992). A critical assessment of the minimalist approach to documentation. In *Tenth annual ACM conference on systems documentation (SIGDOC92)* (pp. 7–17). Ottawa, Canada.

Van der Meij, H., & Carroll, J. M. (1998). Principles and heuristics for designing minimalist instruction. In J. M. Carroll (Ed.), *Minimalism beyond the Nurnberg Funnel* (pp. 19–53). Cambridge, MA: MIT Press.

Wolz, U., & Conjura, E. (1994). Integrating mathematics and programming into a three tiered model for computer science education. *SIGCSE Bulletin*, *26*(1), 223–227.

M y T e r m P a p e r

T h e q u i c k f o x

Figure 1: What you (think you) see

 M y T e r m P a p e r

 ← *cursor*

 M y T e r m P a p e r

T h e q u i c k f o x

Figure 2: What you (really) get

المنارة للاستشارات
www.manaraa.com

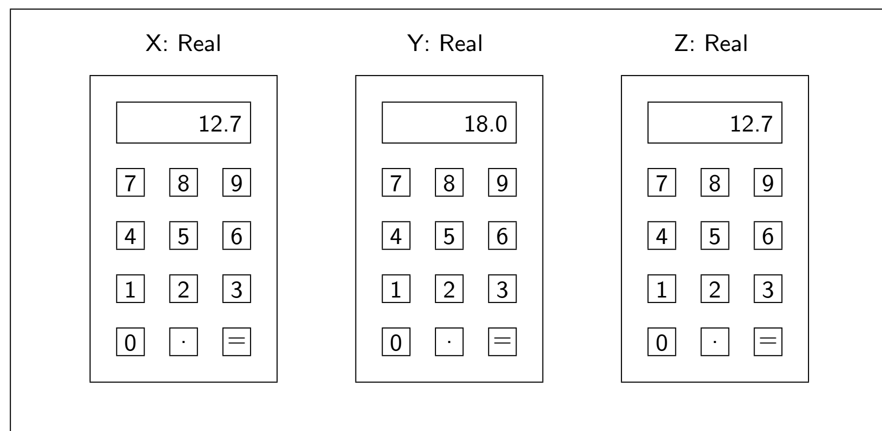

Figure 3: An epistemic game for studying variables

المنارة للاستشارات
www.manaraa.com