From Theory to Experiment to Practice in CS Education

Mordechai Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
and
Department of Computer Science
University of Joensuu
Joensuu FIN-80101 Finland

May 8, 2006

1 Introduction

In this paper, I will summarize a five-year excursion into computer science education (CSE). The excursion began with a theoretical insight, continued with experimental studies and can now be summarized in suggestions for improving the pedagogy of computer science (CS). The theoretical insight pertained to the applicability of constructivism in CSE: I concluded that students find it difficult to construct viable mental models of computers and software artifacts, because they have no pre-existing models upon which to build. I conjectured that the creation of conceptual models by educators can make it easier to acquire viable mental models, and my graduate students have performed empirical experiments to verify this conjecture. These experiments identified the important features of conceptual models that can guide educators, both teachers and developers of learning material and software.

2 Theory: constructivism, minimalism and bricolage

The following summary is based upon [3]. An educational theory has four components: an *ontology* which describes what exists; an *epistemology* which characterizes knowledge; a *methodology* which explains how knowledge can be obtained; and a *pedagogy* which presents teaching techniques based upon the methodology. According to a classical view of education:

- There is an ontological reality.
- Epistemology is foundational: truth can be discovered through experiment and logic.
- Methodology: the mind is a clean slate that can be filled with knowledge.
- Pedagogy: lectures and books are the primary means of knowledge transmission.

According to a *constructivist* view of education [14]:

- Ontological reality is at best irrelevant because we can never truly know anything.
- Epistemology is nonfoundational: absolute truth is unattainable and fallible.
- Methodology: knowledge is acquired by creating new cognitive structures out of existing ones. Each student will construct new knowledge differently.

- Pedagogy: The task of the teacher is to guide the student in constructing her own knowledge. Feedback from teachers and students ensures that the knowledge is viable.

Proponents of constructivism often hold postmodernist positions similar to the idealist position in philosophy. I shall ignore these philosophical aspects here and concentrate on the pedagogical implications of constructivism, which are usually expressed by saying that constructivist teaching is learner-centered and based on active participation, though opponents of constructivism claim that these pedagogical principles have been known since Socrates and do not need to be justified by postmodernist philosophy.

Closely allied with constructivism is *minimalism* [6], which focuses on documentation for the novice who is learning how to use software packages. The principles of minimalism are:

- Start with meaningful realistic tasks.
- Reduce reading and other passive activity, and eliminate or defer conceptual material.
- Make error recovery pedagogically productive.

The success of minimalism has been empirically demonstrated in training tasks like learning to use a word processor. But I questioned whether it can be successful for learning to perform tasks that involve complex concepts.

A final theoretical influence is the concept of *bricolage*, originally enunciated by anthropologist Claude Lévi-Strauss, and used by [12, 13] to describe an exploratory approach to programming and other CS activities:

> Bricoleurs construct theories by arranging and rearranging, by negotiating and renegotiating with a set of well-known materials. ... The bricoleur resembles the painter who stands back between brush strokes, looks at the canvas, and only after this contemplation, decides what to do next.

I distinguish between bricolage and trial-by-error, in that trial-by-error contains elements of analysis, conjecture and prediction, rather than just aimless trials. I believe that bricolage is not an effective methodology for professional programming, and that the normative planning style must eventually be learned, because systems involving concurrency, real-time or communications are simply not amenable to bricolage.

3 Mental models and conceptual models

A central claim in [3] was that constructivism has to be adapted to CSE because a (beginning) computer science student has no effective model of a computer; therefore, one must be provided by the teacher. I would now express this in the terminology used by [9]: the computer or software or language is the *target system*; the student must develop a viable *mental model*; an essential step in this development is the creation of an explicit *conceptual model*:

> A conceptual model is invented to provide an appropriate representation of the target system, appropriate in the sense of being accurate, consistent, and complete. Conceptual models are invented by teachers, designers, scientists, and engineers. [9, p. 7]

The distinction between mental and conceptual models can be clarified by analyzing the concept of *WYSIWYG (What You See Is What You Get)*. WYSIWYG word processors are supposed to be user-friendly, because working with them is supposed to be analogous to writing with a pencil on a sheet of paper, but this metaphor cannot furnish an explanation for the phenomena the user encounters, so he becomes frustrated, anxious and loses self-confidence. In fact, WYSIWYG really is much deeper than the metaphor:

- What you *get* is a data structure for storing text and a set of operations on that data structure.
- What you *see* is a visual rendering of the data structure, and icons to invoke the operations.
- What you have to *do* is to construct a viable mental model of the data structure and the effect of each operation, and to give each icon its meaning as an operation.

I claim that in order to learn how to use a WYSIWYG system, the teacher or the learning materials must construct a conceptual model that will make explicit the existence of the underlying data structure and the mapping between the visual rendering and the data structure.

4 Experiment: conceptual models

I performed an experiment designed to show that users of software packages engage in bricolage [2]. The subjects were asked to perform modifications on an MS-Word document. The subjects were highly-experienced science teachers doing graduate work in science teaching who use MS-Word regularly for writing learning materials, research papers and so on. Thus they are not to be suspected as lacking motivation to learn or an appreciation of quality pedagogy. The tasks were chosen to be relatively easy if the subject knew the relevant concepts (soft carriage returns, the distinction between a table cell and its contents, bidirectional language support, etc.), but quite difficult to perform by aimless trail and error.

Surprisingly, the subjects, in spite of their age, sophistication and experience, performed their tasks like beginning students! They did not analyze the tasks conceptually; instead, they invariably used aimless trial-and-error that I have called bricolage. Furthermore, they did not attempt to use the Help facility. I encountered frequent anthropomorphisms, like "He did it to me again," and "That's not nice of him." They were defensive and displayed a high level of anxiety. Contrary to popular opinion that learning takes place when there are real problems to be solved, when my subjects were desperate they did not use it as an opportunity to learn, but rather used dumbed-down techniques to work around the problem: "I'd work like a donkey."

I conjectured that teaching an explicit conceptual model would improve learning of MS-Word, and this conjecture was experimentally investigated by my M.Sc. student Tzippora Yeshno [15]. Bilingual word processing was taught to three classes of 9th-grade students: the control group received a task-oriented tutorial, while two treatment groups received a concept-oriented tutorial. The students were instructed in common bilingual tasks such as inserting numbers or English words within Hebrew text. Superficially, the behavior of the word processor is quite complex in that the cursor direction and current language change automatically as certain keys are pressed. The control group was taught *how* to perform such tasks, while the treatment groups were taught a *conceptual model* of the target system, namely, that runs of characters in the same family (English letters, Hebrew letters, numbers and special symbols) are arranged in *blocks*, and that the superficial behavior reflects manipulations of these blocks.

Exercises and examinations were given that contained two types of questions: some which could be done using task-oriented instructions, while others required conceptual understanding. Students in the treatment groups used the conceptual model to *verbalize* their answers (both orally and in writing), leading to improved task performance. Students in the control group dealt with the assignments in different manner recalling bricolage, and could not verbalize their attempts to solve the assignments.

5 Experiment: program visualization

The most serious problem in CSE is the difficulty of learning elementary programming; this difficulty causes many beginning students to abandon the study of CS. One suggestion has been to use visualization, which we can interpret as an attempt to use conceptual models that are concrete representations. Hopefully, these models will facilitate the construction of a mental model. However, visualization in and of itself is not a panacea, and certain preconditions must exist for it to be successful. [10] showed that there are significant differences in the visualization needs of experts and novices. Experts are successful at interpreting *secondary notation* (like layout and typography) and can benefit from complex visualization. On the other hand, novices need a constrained system in which secondary notation is minimized in order to reduce the richness and the potential for misunderstanding. In algorithm and program animation, relatively little empirical research has been done and much of that has produced mixed results; see [7] for a survey. For example, in a series of experiments [5, 8, 11], Stasko showed that animations are not pedagogically useful in isolation, but require coordination with learning materials or human interaction. He also showed that the interaction with the learner should not be passive, and that exercises in prediction are important.

My M.Sc. student Ronit Ben-Bassat Levy carried out an empirical experiment with the Jeliot 2000 program animation system [4], a re-implementation of an earlier system now known as Jeliot I. (A survey of both systems appears in [1].) Jeliot 2000 was designed to produce animations that are relevant to total novices, sacrificing the scope and flexibility of the Jeliot I animations. The experiment was performed on tenth-grade high school students studying an introductory course on algorithms and programming. Unlike Stasko's experiments which were short-term laboratory experiments, in our research, two full-year classes were taught, one using Jeliot 2000 and one as a control group. We believe that it is important to evaluate animations and other software tools within the framework of a complete course, rather than in a single episode, so that the students can pass the learning curve for the tool.

Here I give one example of the results; for more detail see [4]. The students had learned simple `if`-statements and then were given an exercise that contained nested `if`-statements, which is considered to be one of the most difficult concepts in elementary programming. In the control group, the stronger students could answer the questions only after many attempts; furthermore, they had difficulty explaining their solutions. Other students could not even answer the questions. Paradoxically, in the animation group, stronger students had difficulties answering this question, because they believed they could understand the material without referring to Jeliot 2000. Weaker students refused to work on the problem, claiming that nested `if`-statements are not legal or that they did not understand the program. The mediocre students were the ones that gave correct answers! They drew a Jeliot display and used it to hand simulate the execution of the program.

Animation does not improve performance of all students: stronger students do not need it, while weaker students are overwhelmed by the tool. However, for many students, the concrete conceptual model offered by the animation can make the difference between success and failure. The explanation is that the animation group learned to use a different and better *vocabulary of terms* than did the control group. In our theoretical terms, animations can provide a conceptual model that will assist students in forming a viable mental model.

6 Practice

The pedagogical lessons learned from this sequence of research projects can be summarized as follows:

- Explicitly teach a model of a computer: hardware, operating system, communications.

- Dig underneath your own expert knowledge to expose the prior knowledge needed to construct a viable model of the material that you are teaching.
- Construct and refine conceptual models for each topic and artifact.
- Design software tools such as visualizations to present conceptual models.
- Take into account the time needed to learn to use tools such as visualization and the time needed to explain the conceptual models.

References

[1] M. Ben-Ari, N. Myller, E. Sutinen, and J. Tarhio. Perspectives on program animation with Jeliot. *Lecture Notes in Computer Science*, 2269:31–45, 2002.

[2] Mordechai Ben-Ari. Bricolage forever! In *11th Workshop of the Psychology of Programming Interest Group*, pages 53–57, Leeds, UK, 1999. http://www.ppig.org/papers/11th-benari.pdf.

[3] Mordechai Ben-Ari. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.

[4] Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 program animation system. *Computers & Education*, 2002. (in press).

[5] Michael Byrne, Richard Catrambone, and John Stasko. Do algorithm animations aid learning? Technical Report GIT-GVU-96-19, Georgia Institute of Technology, 1996.

[6] John M. Carroll. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press, Cambridge, MA, 1990.

[7] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

[8] Colleen Kehoe, John Stasko, and Ashley Taylor. Rethinking the evaluation of algorithm animations as learning aids: An observational study. Technical Report GIT-GVU-99-10, Georgia Institute of Technology, 1999.

[9] Donald A. Norman. Some observations on mental models. In Dedre Gentner and Albert L. Stevens, editors, *Mental Models*, pages 7–14. Lawrence Erlbaum Associates, 1983.

[10] Marian Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, 1995.

[11] John Stasko, Albert Badre, and Clayton Lewis. Do algorithm animations assist learning: An empirical study and analysis. In *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems*, pages 61–66, Amsterdam, The Netherlands, 1993.

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

[13] Sherry Turkle and Seymour Papert. Epistemological pluralism and the revaluation of the concrete. In Idit Harel and Seymour Papert, editors, *Constructionism*, pages 161–191. Ablex, Norwood, NJ, 1991.

[14] Ernst von Glasersfeld. A constructivist approach to teaching. In Leslie P. Steffe and Jerry Gale, editors, *Constructivism in Education*, pages 3–15. Lawrence Erlbaum Associates, Hillsdale, NJ, 1995.

[15] Tzippora Yeshno and Mordechai Ben-Ari. Salvation for bricoleurs. In *13th Workshop of the Psychology of Programming Interest Group*, pages 225–235, Bournemouth, UK, 2001. http://www.ppig.org/papers/13th-yeshno.pdf.