

Luehrmann, A. (2002). "Should the computer teach the student..." — 30 years later. *Contemporary Issues in Technology and Teacher Education*, 2(3), 397-400.

"Should the Computer Teach the Student..." — 30 Years Later

ARTHUR LUEHRMANN
Publisher, Computer Literacy Press

Until editor Lynn Bell suggested I write a page or two about how things have turned out since 1972 (the year I presented the paper "Should the Computer Teach the Student, or Vice-Versa?" at a Boston conference) I hadn't reread this little parable in ages—probably not since Bob Taylor republished it in "The Computer in the School: Tutor, Tool, Tutee" in 1980. I enjoyed the reread, and I hope others will too.

And how have things turned out? That's easy. Out of Taylor's trichotomy, teaching tool use is just about the only impact that computers have had on schools. Walk into any middle or high school and ask to see the computers. Most will be found clustered in a computer lab, not in the classrooms. Go to the lab and ask a student what he or she is doing. The most likely answer is, "I'm working on a word processing (or spreadsheet, or database, or graphics) assignment for my computer applications class." They're learning computer tools, in short, even though they rarely use them outside the applications class.

The computer as tutor (then called CAI) is today limited to a few useful keyboarding tutorials and some drill-and-kill programs still inflicted on kids in mainly inner-city schools. The big federally funded CAI projects of the 1970s have come and gone, leaving little trace. And I, of course, say hurrah.

Similarly, the computer as tutee (programming) is limited to a tiny fraction of students aiming for careers in computer science. The days are gone when the central focus of the typical classroom with computers (few as they were) was to solve problems by writing simple programs in Basic or Logo.

This saddens me. Seymour Papert and I argued a lot about Basic versus Logo as a suitable language, but we were in firm agreement that learning the syntax of a particular programming language was far less important than being able to express an understanding of a problem or an idea by means of a simple program. My freshmen at Dartmouth College, for example, could show their understanding of Newton's Laws of Motion by writing 10- or 20-line programs to calculate and plot orbits. Papert observed small children mastering profound concepts in geometry by writing tiny Logo programs to move a turtle about the screen.

Two occurrences killed programming in schools. The first was a commercial program called VisiCalc, which appeared in the early 1980s and was responsible for making the Apple II the business computer of choice—a fact that few remember. VisiCalc was the first spreadsheet program, and a brilliant conception from the start. (It was essentially copied with minimal changes, first as Lotus 1 2 3 and then again as Microsoft Excel.)

Pity the poor high school programming teacher when these spreadsheet programs first appeared. Think of all the effort students spent mastering programming procedures for sorting, searching, and printing out tables of results. All these things could be done in a minute by using a spreadsheet. "What was the point of programming?" students asked.

It was a good question, and the typical programming class did not have the answer. The proper answer, then and now, is that elementary programming does not belong in a programming class (there was none at Dartmouth back then). Rather, it needs to be included in appropriate places across the curriculum. Just as students now learn to express their understanding in many subjects by writing and by using mathematical formulas and procedures, and not just in the English and math classroom, so also should they learn to use computer algorithms when they aid understanding of a subject.

You see the problem here. It was tough enough to find teachers for a programming class. Integrating even the most elementary programming methods across the curriculum is monumentally difficult. It is a fact of life in education that teachers teach what they were taught—often decades ago. Few elementary math teachers were taught turtle geometry (then or now, for that matter).

The second occurrence that killed elementary programming for nearly all students was the creation by the College Board, also in the early 1980s, of

the Advanced Placement Test in Computer Science and the resulting APCS classes in high schools. That may seem counterintuitive. Isn't programming the focus of the AP test?

It is, of course, and the programming tested is indeed "advanced"—essentially what college freshmen learn in their first computer science course, based on C++ or Java. That, however, is not what general students need to be taught if the goal is to learn to use computer algorithms to express ideas and understanding of various subjects in the regular curriculum.

The result, as I predicted in a letter to the College Board in 1980 discouraging the APCS test, has been a fairly complete professionalization of programming as a tool for future computer scientists. A tiny fraction of high school students is enrolled in APCS today, and they are largely people who intend to have a future major or minor in computer science. They are mostly male, and some might argue about their social skills. In any case, many students who might otherwise be interested in programming are not drawn to the APCS classroom.

Well, you win some, and you lose some. The battle against CAI that I argued in my little 1972 parable is a win. The battle for programming methods in the general curriculum is pretty much lost. In between, many of the tool uses that I also argued for at the end of the paper, are gaining ground. I am optimistic that over the long haul subject matter teachers will discover for themselves (or, as we found at Dartmouth, students will show them) how various computer tools can enhance students' understanding of their subject.

How long is the haul? In the September 9, 2002, issue of *Business Week* (p. 26), technology editor Stephen Wildstrom takes schools to task for failure to prepare students for entry into colleges, which have embraced computers and the Web. He quotes from a paper by Cathleen Norris and Terry Sullivan, University of North Texas, and Elliott Soloway, University of Michigan: "To a first-order approximation, the effect of computing technology in the past 25 years on primary and secondary education has been zero." Soloway separately says, "Colleges were willing to take the risk of saying that technology is relevant to education. K-12 doesn't believe it."

So it could be a long haul indeed for K-12, where DWYDLY (do what you did last year) is the prevailing philosophy. That situation might change as

voters realize that “universal public education” is not synonymous with “government run school monopolies.” DWYDLY works when schools have a captive market. When they must compete for (publicly funded) students, free to go where their parents please, suddenly a strong, fully integrated computer education program becomes a selling point (as it is for colleges that compete aggressively for students).

The take-away message about the future of computers in K-12 education is this: As long as the school monopoly is in force, the gloomy appraisal of Norris et al. is likely to remain accurate—DWYDLY wins. Force schools to compete for students, however, and you provide the incentive for massive change: online course schedules and descriptions; class Web sites with resources, problem sets, and solutions; virtual study groups; Web access and e-mail for communicating with parents. Only in such an environment, rich with computer tools, will any of the other computer uses we envisioned 30 years ago become possible.