

THE DESIGN OF SOFTWARE TOOLS FOR MEANINGFUL LEARNING BY EXPERIENCE: FLEXIBILITY AND FEEDBACK*

DAVID F. JACKSON
University of Georgia

BILLIE JEAN EDWARDS
Detroit Public Schools

CARL F. BERGER
University of Michigan

ABSTRACT

Experience in using commercially available software to teach students about principles of graphical data analysis suggests that several critical design modifications are advisable. In a quasi-experimental study, three different versions of an original graphing program were used by inner-city high school students solving scientific data analysis problems. A version incorporating "coaching" feedback into a highly flexible interface was found to be significantly superior to either an "open" version giving no extrinsic feedback or a "restrictive" one that disabled program options whose use was deemed inappropriate based on the data analysis context. As an illustration of one of the graph-based critical thinking skills developed by the students, results are presented as contrasting pairs of graphs in which one is designed to emphasize, and the other to downplay, the effects of interface design, gender, and their interactions.

Discussions of educational uses of computers commonly distinguish between three distinct roles for the computer [1]: first, as personal tutor; second, as a medium for experiential learning, especially through programming ("tutee" [2]); and third, as a multipurpose tool. The first two of these categories have frequently been the focus of research, and existing software spans a great range of sophistication in both

*An earlier version of this article was presented at the annual meeting of the National Association for Research in Science Teaching, Atlanta, Georgia, April 9, 1990.

cases: from simple drill-and-practice exercises to "intelligent" tutoring systems and from student-designed programs to expertly prepared simulations.

Recently, more attention has been paid to the "tool" software most commonly used in science education, microcomputer-based laboratory (MBL) materials. In particular, the results of research on teaching about line graphs representing physical phenomena have been more consistently positive than in any other area of computer use [3]. The software used in this study is more general in scope, providing for the use of four different major types of graphs (column/bar, line, pie chart, and scatterplot) with any data in the form of a standard cases-by-variables matrix. An important distinction is that both the data and the graphic representations are static, as opposed to the dynamic, real-time graphs typically generated in MBL activities.

While the use of more general applications programs (word processors, databases, spreadsheets, graphing utilities, etc.) in schools is increasing, the effects of the characteristics of these programs on subject matter learning have not often been critically examined. What are the consequences of using such software as a vehicle for experiential learning? This approach straddles the boundaries in the above classification scheme, and places teachers and curriculum developers in the position of adapting a technology to an unintended purpose [4, p. 7]:

In many cases, applications software was originally developed for business rather than educational uses, and was aimed at adults, rather than children, as users. . . . Because applications software is, almost by definition, not intentionally educational in nature, it is in many ways fundamentally different from other kinds of software used in the classroom.

One critical difference between applications software and other educational software is that there are not specific learning objectives. That is, . . . applications software is not *designed* to teach anything.

This is not to say, however, that applications software cannot be used to teach a large set of problems, concepts, and skills.

Discussions of the design of the interface of applications software in relation to novice users are very common in the literature of computer science and industry (e.g., [5]). Such research, however, is most often concerned with the process of learning about the mechanics of the program itself, rather than the uses to which it can best be put in teaching and learning in content domains.

In educational applications, user interface design has received little attention, despite the fact that the interface is particularly important for educational software because "it must provide an entry to the content domain of the program rather than vice versa" [6, p. 93]. This issue is especially crucial when, as in the case of this study, no previous experience with the topic at hand (graphing and data analysis) is presumed. This concern goes much deeper than the nebulous concept most often represented by the buzz phrase, "user friendliness." The students in this study had already worked with the Macintosh™ operating system (renowned for its

favorable learning curve) for a minimum of five weeks when they first encountered graphing software. Nevertheless, many had several serious difficulties with the commercially-available software used in the early stages of the project.

In this study we investigate what special considerations should enter into the design of graphing software for *educational* purposes, rather than for use by scientists, engineers, or businesspeople (cf. [7] for the special case of graphs as representations of algebraic equations).

THEORETICAL BACKGROUND: EXPERIENCE, EXPLORATION, AND COMPUTER-BASED ACTIVITIES

Like anyone claiming to be concerned with "learning by experience," we should first return to our roots in Dewey's technical definition of education: "the reconstruction of experience which adds to the meaning of experience, and which increases ability to direct the course of future experience" [8, p. 76]. Dewey is known for his emphasis on the inherent (vs. instrumental) value of "meaningful" experiences in the present, but the typical caricature of his work ignores the value that he also placed on past "truly educative" experiences by virtue of their effect on the future [9]. We have noted elsewhere [10] that the essence of instructional design in this spirit lies in an effort to "facilitate *and* regulate the exploration of alternatives on the part of the learner" [11, p. 43] (*italics added*). As Dewey later wrote, in response to the rampant misinterpretation of his ideas [12, p. 75]:

It is a mistake to suppose that the principle of the leading on of experience to something different is adequately satisfied simply by giving pupils some new experiences any more than it is by seeing to it that they have greater skill and ease in dealing with things with which they are already familiar.

Many of the participants in the past decade's debate about the role of microcomputers in education could profit from reminding themselves of the oft-ignored complexity of Dewey's and Bruner's thought. At one extreme are "visionaries" [1] such as Papert [13], who place tremendous faith in the ability of the child-computer dyad to achieve valuable learning from repeated, *self-regulated* experience using flexible, powerful software. This idea has been summarized as "the opportunity does the teaching by itself" [14, p. 13]. To be fair, this seems to be one of many instances in which Papert's view has itself suffered the indignities of oversimplification at the hands of its critics.

At the other extreme lie those, also hailed as visionaries during an age in which the most modest computing resources were prohibitively expensive, who see the educational potential of computers primarily in terms of more consistent and efficient methods of didactic instruction (e.g., [15]). Translated into a world of

relatively inexpensive and powerful microcomputers in classrooms, this view rejects the "romanticism" of experiential learning in favor of sophisticated, artificially intelligent tutoring systems that have the potential to fully replace a human teacher in well-defined knowledge domains such as biology, chemistry, or physics [16].

Regardless of one's predilection for designing explicitly educational software as either tutee or tutor, as outlined above, the development of a theoretical basis for the design and use of tool software for education remains problematic. Is complete flexibility in exploration of alternatives always a reasonable model? Is it possible for a truly multi-purpose tool to provide any useful feedback or guidance?

If students, not just teachers, are ever to regard computing power as a useful means to a worthwhile end, rather than as a surrogate teacher or as an object of study in itself, perhaps tool software should be designed to call as little attention to itself as possible. The ideal would be to replace human-computer communication with "human-problem-domain communication" [17], allowing students to forget that they are using a computer and concentrate on the subject-matter issues at hand. At a more basic philosophical level, this recalls Polanyi's vivid metaphor concerning the "internalization" of familiar tools: a hammer, for instance, gradually comes to be viewed as a natural extension of one's hand, and ceases to be thought of as a separate technological artifact at all [18]. Winograd and Flores, attributing this metaphor to the work of Heidegger, have specifically included it in their high-level theorizing about the appropriate aims of software and computer systems design [19].

This argues against an attempt to burden a tool program with so many *ad hoc*, prescriptive functions that it becomes more like a clumsy and thinly veiled tutorial [20, p. 6, 9]:

What are we trying to get the computer to get the student to do, in designing a teaching program? We need some model of the optimal learning situation to emulate. The one-to-one tutorial is not appropriate for a medium where the teacher is necessarily absent. The focus is not going to be student-teacher, but student-subject.

... the way the student experiences the domain being learned [should be] through direct manipulation of ... the objects in the domain, [with] feedback about the behaviour of the system in terms of the results of the manipulations.

... The advantage of ... goal-oriented manipulation is that it ... allows the student to "act like a scientist."

These arguments were originally advanced in the context of open-ended simulations of natural processes, many of which have been used in teaching physics (e.g., [21]). We believe that they can also fruitfully be applied to tool software in a case (such as that of a graphing utility) in which the "system" involved is a set of

general-purpose mathematical representations, rather than a mathematical model of the operation of natural laws. Any such program gives *intrinsic* feedback, that is, the substantive results of the student's computer commands in terms of the system's behavior (including both modified graphs and messages such as "that is not possible"). *Extrinsic* feedback, in contrast, is a nonspecific evaluative comment on the match between the students' input and some predefined goal state ("correct," "incorrect," or "close, but . . ."), which is more characteristic of a tutorial program [20]. One version of our original graphing software used in this study is strictly limited to intrinsic feedback—learning about graphs from the samples it provides relies heavily on self-regulated critical thinking on the part of students.

The available empirical evidence calls into question the notion that many students will learn important lessons from relatively unsupervised exploration of problems, whether or not they are aided by powerful and appropriate computer software. Research in this area has not been limited to the specific, emotionally-charged issue of evaluation of the educational use of LOGO programming (e.g., [22, 23]). Powerful, potentially helpful program features are rarely used by students, no matter how much they might help, if they are clearly presented by teachers as optional [24]. The educational value of time- and labor-saving software can depend heavily on the context in which it is used, including the level of student motivation: ". . . it seems that the avoidance of cognitive load in the absence of any real press to behave more elaborately leads to a failure to exercise higher-level skills" [14, p. 13]. Extensive studies of problem solving in abstract mathematics have shown that students also fail to recognize or internalize useful ideas or skills that were practiced or explicitly modeled but not explicitly identified or explained [25].

Certainly, software designed to be used by students to gain experience in graphical data analysis must be flexible enough to allow them to explore unknown territory, and therefore to make some mistakes. The hope is that these will ultimately contribute to learning. The implications of actions which are logically possible but inadvisable, reflecting the application of erroneous concepts, can then be confronted and evaluated.

Our early experience, however, clearly indicated that unrestricted freedom of choice from among a large array of powerful software commands can be disorienting. Taking a cue from many successful commercial programs, another version of our graphing software selectively disables ("grays out," in terms of the Macintosh™ interface) many program options in certain contexts. We characterize this as "preemptive" feedback—a tacit yet tyrannical form.

Students cannot reasonably be expected to induce the major principles of the appropriate use of graphs without some degree of appropriate feedback. Applications software that is too restrictive or prescriptive, however, is probably not conducive to thoughtful experience and, therefore, to meaningful learning. If an important goal of computer-based educational activities is to cultivate an

appropriate balance between the exercise of raw computing power and of human judgment (cf. [26]), how should software best be designed?

A theoretical perspective on tool software that we find most intriguing is Salomon's concept of "artificial intelligence in reverse" [27]. Rather than a computer modeling human intelligence, might human students come to emulate certain aspects of the computer's performance? In this view, the power embodied in the responses of a computer program serves a role analogous to that of Vygotsky's "more capable peer," in the spirit not of a tutor but more an assistant or partner, with whom a student can accomplish things that she or he could not handle alone [28].

The primary benefit of a computer tool is its "enabling function," allowing the user to test new possibilities and examine their consequences. But perhaps tool software can (although programs aimed at professionals rarely do) perform somewhat intelligent "guidance" and "modeling" functions by *subtly* raising questions, signaling possible errors, or providing externalized metacognitive guides [27]. One version of our graphing software allows great freedom in experimenting with different graphic forms for displaying data, but periodically gives verbal reminders and warnings ("coaching" notes) about the assumptions implicit in the action that the user has taken. It provides, in other words, not artificially intelligent tutoring but "heuristic guidance" in order to facilitate "semantically constrained exploration" [29].

Salomon also raises the issue of evaluating the success of students in computer-assisted activities [27]. Particularly well-designed software as an intellectual partner can be expected to upgrade performance during the partnership, but is there a useful "cognitive residue" that can eventually be used without the scaffolding of the external, intelligent tool? Building on Vygotskian terminology, there may be a zone of proximal *performance*, but not necessarily a zone of proximal development. Improved performance is necessary but not sufficient to demonstrate learning. In other words, besides the enabling function of the software, which cannot (and need not) be internalized by the students, do the guidance and modeling functions seem to stick with them? This concern is our reason for evaluating the various software interfaces by the criterion of their effect on written post-test scores, rather than on immediate success in answering the "research questions" addressed in the computer-assisted problem sessions.

DESIGN AND PROCEDURES

This study was part of a curriculum development project, undertaken jointly by University of Michigan researchers and teachers in the Detroit Public Schools (DPS) whose classes were involved. In total, eighteen teachers and over 800 students from six high schools participated.

As a group, students from these schools typically fall far below state and national means both in overall achievement and in exposure to science and

mathematics courses. Detailed background data on individual students were not available from the district for our research purposes. Most participating students were in the ninth and tenth grades, and fewer than 15 percent reported having taken bona fide algebra or laboratory science courses in the past.

The unit on graphical data analysis was part of the DPS Computer Applications Program (CAP) curriculum, and comprised seven fifty-minute class sessions. The computer lab classrooms were equipped with Apple Macintosh™ (512K, unenhanced) hardware. The application program used most recently, *CAPGraph*, was developed to incorporate several software design principles derived from our experiences with students in the first stage of this study, in which the commercial program *Cricket Graph*™ was used [30].

In the first three class sessions we used didactic, whole-class instruction aimed at familiarizing the students with reading and using graphs and with the capabilities of the software. Exposition of general principles was intertwined with numerous examples of the use and misuse of graphs in various data analysis contexts. Students were repeatedly given "hands-on" exercises in using the computer commands and interpreting their results.

Three sessions were then devoted to loosely guided practice in graphical data analysis in the context of problem sets. For each problem, students were given a data set (in the form of a spreadsheet file) and a "research question" which could be answered by creating and modifying an appropriate graph on the computer. Students most often worked in pairs at each computer station throughout the unit, an arrangement originating in necessity but which has several advantages both for teaching and learning and from the perspective of cognitive process research [30].

The final class session comprised two tests, a hands-on evaluation of facility with the mechanics of the software and a written test on graphing principles. The latter was a paper-and-pencil, free-response test, designed to correspond as closely as possible to the task environment familiar to the students from the computer problem-solving sessions. Questions asked on the test probe not only the students' basic competence in reading and drawing inferences from graphs, but their ability to make evaluative judgments about the design and modification of graphs appropriate to various kinds of problems.

This study was conducted in three distinct phases, corresponding to an evolving set of research questions and methodological approaches. First, in early trials with a small number of classes at two schools, students used a commercial program, *Cricket Graph*™, which is extremely popular in business and academia and known for its especially "friendly" interface. It eventually became apparent that this extremely powerful software had several serious flaws from the point of view of experiential learning by high school students.

Second, we programmed an original application, *CAPGraph*, designed specifically to alleviate the most troublesome problems, from the students' point of view, of the commercial software. This software was pilot-tested by students at three schools, whose critical comments and suggestions were actively encouraged and

usually heeded. During this phase, significant aspects of the program's interface were often changed, literally, overnight. This mode of operation empowered everyone involved, researchers, teachers, and students, rather than immediately adopting a more circumspect mode of research in which existing computer artifacts are implicitly taken as the universe of possibilities. This is a reflection of [31, p. 95]:

. . . a "two-directional image": not only do computers affect people, but people affect computers . . . we affect computers when we study their use, reflect on what we see happening, and then act to change it in ways we prefer or see as necessary to get the effects we want. Such software engineering is fundamentally a dialectical process between humans and machines. We define the educational goals (either tacitly or explicitly) and then create the appropriate software. . . . Through experimentation, new goals and new ideas for learning activities emerge.

This process of progressive refinement characterized not only our instructional and software design process, but also our theoretical perspective. For a reflective participant observer, iterations of such "feedback loops" cannot help but also modify the set of values and assumptions that color the very process of observation and measurement ("feedforth" [32]). In this intermediate phase, hypotheses were generated for the quasi-experimental study that followed.

Third, we explicitly took stock of our educational goals, "froze" certain basic aspects of the design of the software, and created three systematic variations of the interface, which were then used by students in four schools. Entire classes were assigned on a random basis within each school to work with a particular interface. The differences among the three versions of the otherwise identical software were very well defined, avoiding the thorny issue of satisfactorily "equivalencing" qualitatively different experimental treatments [33], a common cause of confounding of variables in educational computing research [34].

Our three different versions of *CAPGraph* represent different combinations of flexibility and feedback (Table 1): first, a maximally flexible interface, permitting all logically possible command choices and showing their resulting effects on the graph without comment; second, the flexible interface with "coaching" feedback presented in conjunction with the graphical results when a questionable command is issued; and third, a relatively restrictive interface that disables several program commands in contexts in which they are logically possible but not advisable, according to widely-accepted principles of exemplary graphing practice (e.g., [35]).

The programmed functions generating the "coaching" messages are based only on the data structure of the current graph and a general model of appropriate graphing practices for each combination of data types. The modestly artificially intelligent performance of the software did not depend on any information specific

**Table 1. Versions of CAPGraph Software,
Characterized by Degree of Flexibility and Feedback**

Interface Design	Flexibility	Feedback
"Open"	flexible	implicit
"Coaching"	flexible	explicit
"Restrictive"	less flexible	"preemptive"

to the "research questions" in the problem sets with which the students were working.

We then compared and contrasted the experiences of students using each style of interface, using anecdotal observations to help interpret the results from the written test of practical knowledge of graphing.

RESULTS

Phases 1 and 2:

Experiences with Commercial Software; Development and Refinement of the Original Prototype Software

Before this study began, the only alternative available to Detroit teachers wishing to include computer-assisted graphing in their curriculum was to use the graphing facilities of *Microsoft® Works™*. While this integrated software tool includes a word processor and spreadsheet that are more than adequate for student purposes, the user interface to its graphing functions leaves most students (and many adults) overwhelmed and befuddled.

When the basic "new chart" menu command is selected, this program immediately confronts the user with a large number of simultaneous choices (Figure 1). This ability to specify all aspects of the appearance of a graph all at once, in a single "batch" process, is a boon to efficiency *if* the user is an expert professional who has already planned her or his graphic display in advance *and* had the foresight to note down the appropriate column and row numbers on the (totally obscured) spreadsheet that contains the data to be graphed. A student, in contrast, is very likely to spend nearly as much time figuring out which numbers and letters to type into which boxes as he or she would have spent plotting a graph by hand. In either case, little class time is likely to remain for the students to consider the important substantive issues involved in designing the graph display itself. This program can save some time and labor by using the computer for drafting graphic presentations, but is not conducive to thinking or learning about graphing and data analysis while doing so.

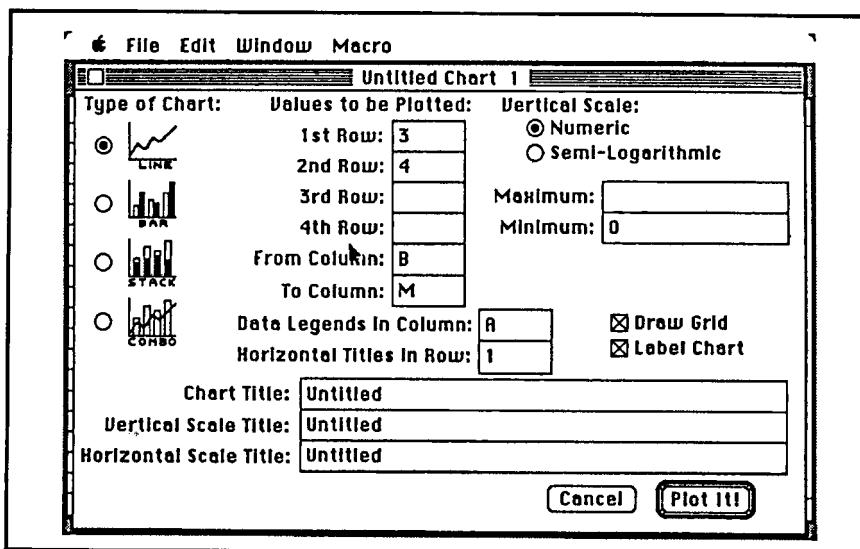


Figure 1. A relatively complicated, non-intuitive interface (*Microsoft Works™*): all important design choices are concentrated in one dialog and must be specified in terms of hidden spreadsheet coordinates.

Cricket Graph™, in contrast, is an outstanding example of highly sophisticated scientific/business software which, because of its relatively convenient and intuitively satisfying interface, is easily learnable by inexperienced students. The user can create a basic graph of any type by choosing from a menu that includes both names and graphic depictions of choices, then merely clicking on the names of the variables that should be plotted (Figure 2). The most basic and important options are presented first, and one at a time, in a way that allows a student user to concentrate on thinking about the substance of the decisions to be made about the graph, rather than on how to communicate their ideas to the computer (cf. [6, 17]). More subtle changes in the resulting graph can then be made individually through separate pull-down menu commands or, in some cases, by simply pointing and clicking the mouse to the part of the graph that needs work.

Almost all of the students involved in this study took immediately to the ease with which graphs could be created, modified, and embellished using this basically well-designed interface. We became increasingly excited about the potential of such computer power and intuitively appealing software to facilitate creative, productive "messing about" on the part of students learning about graphs and scientific data analysis [36]. Eventually, however, a list of serious problems with using *Cricket Graph™* for teaching grew long enough that we were inspired to design and program a new application, *CAPGraph*, specifically for this project.

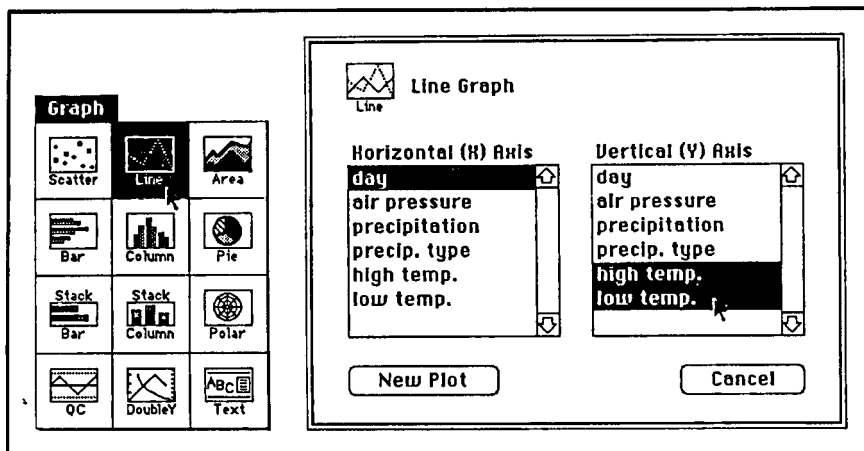


Figure 2. A simpler, more intuitively satisfying interface (*Cricket Graph™*): menu choices include graphics (left) and result in a dialog offering limited choices in terms of meaningful names (right).

Unfortunately, from the teacher's point of view, the combination of a "user friendly" operating system with a full-featured commercial program can rapidly become an attractive nuisance. A sizable proportion of the students were seriously distracted by the constantly available opportunity to play with the Macintosh™ fonts, styles, and sizes of text, or any of several superficially visually impressive, "professional-looking" cosmetic additions to their graphs, such as the illusion of depth ("chartjunk" [37]). These options were eliminated in our new prototype.

For a vast majority of the students, the more advanced mathematical features of the commercial program created a similar problem. Very few were ready to handle the interpretation of such options as logarithmic scales, *n*th-order polynomial curve fitting, inverse-square data transformations, or "smoothing" of data (Figure 3). Such facilities were not included in *CAPGraph*.

As the functions just mentioned would suggest, *Cricket Graph™* is designed to be a general data-manipulation utility as well as a graphing program. Presumably because of this, the software is programmed to respond to some of the most powerful and commonly used commands in a way that students found highly counterintuitive. The "sort" command, for instance, has no effect on the current graph, which fills most of the screen, but rather on the data set itself, as reflected in a spreadsheet window that is typically barely visible beneath the graph window. If any reorganization of the data is to be reflected in a graph, an entirely new graph must be plotted, meaning that all of the design choices and modifications made up to that point in the work session are lost and must be reiterated. Students found this extremely frustrating. Many told us that they learned to avoid the sorting function

altogether for this reason, rather than having to retrace their steps or to plan their graphs in advance to avoid such an annoyance. The design of *CAPGraph* rectifies this situation by making all data analysis functions apply to the current graph and locating them all on a single menu (Figure 4, left).

"Sort now, or forever hold your data organization" is only one way in which this powerful commercial program unreasonably limits the freedom of students to explore the comparisons and contrasts between different graphs on the same data.

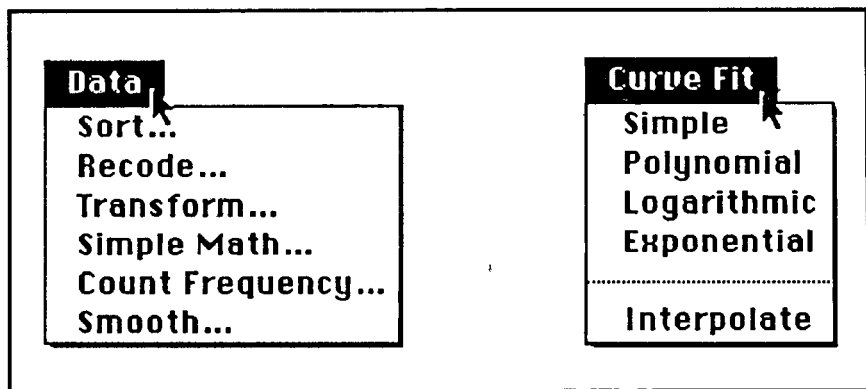


Figure 3. Educational drawbacks of an excellent scientific program (*Cricket Graph*TM): reorganization commands, counterintuitively, act only on spreadsheet data, not on the existing graph (left). Some commands on both menus shown are unintelligible to mathematically unsophisticated students.

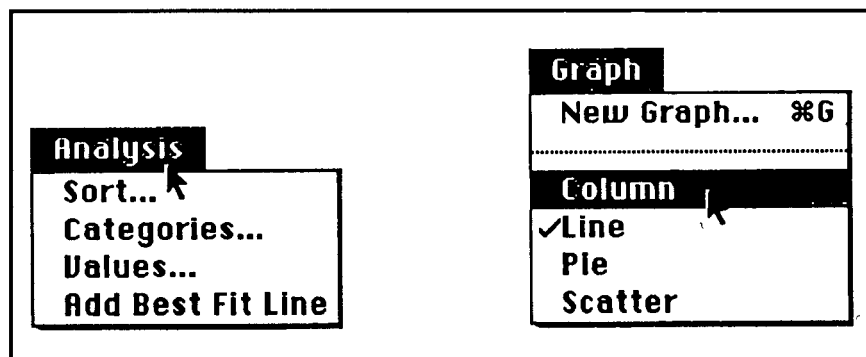


Figure 4. Flexibility provides for changing major organizational components of the current graph at any time (*CAPGraph*): data reorganization (left) or graph type (right).

With *Cricket Graph*TM, the crucial first choice made by the user (type of graph; Figure 2, left) is irrevocable; if a student decides to consider an entirely different graphic form, she or he must start from the computer-assisted equivalent of "scratch." Borrowing a term from the financial world, we refer to this interface design as enforcing a "front-end loaded" decision structure. This places a premium on foresight, while hobbling the students' efforts to explore the effects not only of relatively small refinements in their graphs, but also of large-scale changes in format. For facilitating learning about graphs from experiences in designing and using them, we feel that the single most important feature of *CAPGraph* is its ability to instantly transform otherwise identical graphs from one basic type to another (Figure 4, right).

Once a type of graph is selected, however, there are some inherent, logical limitations on how data can be represented. Except in the case of a sidelong bar graph, for instance, it is impossible to represent categorical information on the vertical or polar scale of a graph—it provides no quantity to determine the height of a column, the angle incorporated by a sector of a pie graph, or the vertical coordinate of a point on a scatter or line graph. For this reason, selecting a categorical variable under "vertical axis" (e.g., Figure 5, top) is a logically impossible command to execute. In the case of *Cricket Graph*TM, this is communicated to the user by an error message (Figure 5, bottom) which, while meaningful from the software algorithm's point of view, is not very meaningful to most people. A very reasonable but erroneous student interpretation might be that data is missing for a large number of the individual cases for the variable in question, rather than the realization that the program had unsuccessfully searched for *numeric* data in the selected column of the spreadsheet. We believe that a more reasonable approach is to disable variables that are logical impossibilities (not merely inadvisable choices) (Figure 6).

This relationship between the choice of an appropriate type of graph and the nature of the information displayed is the single most important concept in graphical data analysis. We therefore chose to include both of these critical decision points in the same initial graph-specification function (Figure 6) in *CAPGraph*, in an effort to encourage students to think about the mutual influence that these two considerations should have on each other.



The design of this dialog box also illustrates several other principles that we have abstracted from our experience with the students about how a graphing tool designed for educational purposes should differ from the interface most typical of commercial software.

Generally accepted interface design practice calls for the first item in a list, the first "radio button" in a set, etc. to be automatically selected as a default unless otherwise specified. The original prototype of *CAPGraph* conformed to this expectation, and in our pilot testing of the program nearly every graph produced in the problem sessions was a column graph. Interviews with students revealed that they were not showing a preference, but merely not bothering to address the





Line Graph

Horizontal (X) Axis

day	 
air pressure	
precipitation	
precip. type	
high temp.	
low temp.	

Vertical (Y) Axis

day	 
air pressure	
precipitation	
precip. type	
high temp.	
low temp.	

New Plot

Cancel



There are not enough data points to complete this request.

OK

Figure 5. An interface with no preemptive feedback and uninformative explicit feedback (*Cricket Graph™*): the student is free to specify a logical impossibility—representing a categorical variable on the vertical scale of a line graph (top)—and then must contend with a nonspecific, cryptic error message (bottom).

issue. This phenomenon was confirmed by similar observations when the ordering of graph types was changed for the next day's class sessions. Since one of our most basic goals was to get these students more *actively* involved in *thinking* in the context of their computer-assisted school activities, the revised version of *CAPGraph* does not use arbitrary program defaults, but rather compels the student user explicitly to choose a type of graph and a combination of variables (Figure 6,

New Graph		Title:	<input type="text"/>
Horizontal Axis:	Vertical Axis:	Graph Type	
<input type="text"/> Date Air Pressure (in Hg) Precip. Amount (in) Precip. Type High Temp. (°C) Low Temp. (°C)	<input type="text"/> Date Air Pressure (in Hg) Precip. Amount (in) Precip. Type High Temp. (°C) Low Temp. (°C) (frequencies)	<input type="radio"/> Column <input type="radio"/> Line <input type="radio"/> Pie <input type="radio"/> Scatter	
		<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

New Graph		Title:	<input type="text" value="Line Graph of Weather"/>
Horizontal Sequence:	Vertical Axis Values:	Graph Type	
<input type="text"/> Date Air Pressure (in Hg) Precip. Amount (in) Precip. Type High Temp. (°C) Low Temp. (°C)	<input type="text"/> Date Air Pressure (in Hg) Precip. Amount (in) Precip. Type High Temp. (°C) Low Temp. (°C) (frequencies)	<input type="radio"/> Column <input checked="" type="radio"/> Line <input type="radio"/> Pie <input type="radio"/> Scatter	
		<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 6. An interface designed to appropriately shape students' thinking (CAPGraph): two decisions—graph type and choice of variables—are juxtaposed because thinking about the two issues should be closely related. Logically impossible choices are disabled (cf. Figure 5). No arbitrary default values are preselected, helping to enforce consideration of the issues. Commands cannot be confirmed—"OK" is disabled (top)—until enough information has been explicitly specified (bottom). More informative, context-sensitive labels and reasonable, customized default values—"sequence," "values," and a generic title—are added after the type of graph is chosen (bottom).

top). This is enforced by the fact that the “OK” button does not “light up,” indicating that the graph specifications can be accepted and the commands executed, until enough selections have been made.

In some cases, reasonable default settings for graph parameters can be determined by the input already given, as in the case of an initial title (Figure 6, bottom). Sensitivity to context can also provide subtle but useful hints to the students in constructing graphs appropriate for a given combination of information. Once a graph type has been chosen, for instance, the headings for the lists of variables can be customized to reflect the kinds of information most typically represented by such a graph (Figure 6, bottom).

An important subset of graph modification commands are those commands controlling the scale of the axes and their extensions (maximum, minimum, increment, grid lines, etc.). In *Cricket Graph™*, these functions can be efficiently and conveniently summoned *en masse* by double-clicking the mouse on one of the axes (Figure 7). We found that there are two major problems with this design from the point of view of student learning. First, since the functions are *only* available through a “hidden” command (rather than a pull-down menu which can be scanned), it was common for students simply to forget that these options exist. Second, in accordance with one of the basic tenets of scientific reasoning, we

Vertical (Y) Axis Format

Axis

Minimum: 20.000

Maximum: 70.000

Increment: 10.000

☒ Linear ☐ Log

Ticks

	Mark	Grid
Major:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Minor:	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Inside:	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Axis on: ☒ Left ☐ Right

OK

Cancel

Figure 7. Convenience, efficiency, and clarity, but perhaps too much for students to consider at once (*Cricket Graph™*): several important and related options are offered in the same dialog box, implicitly discouraging systematic experimentation with one of these graph variables at a time. These commands are available only through a “hidden option,” not a menu selection.

wanted to encourage experimentation with only one structural parameter at a time, so that its unique effect on the resulting modified graph could be judged (cf. [7]). *CAPGraph* deals with these considerations by presenting these features of the program as separate items on a single menu, each calling up a separate dialog box (Figure 8, top) while also allowing more experienced and confident users to use the more subtle, inclusive, and efficient command structure (Figure 8, bottom; cf. Figure 7) instead. This design caters to multiple modalities both in terms of the mechanics of software control (pull-down menus versus double-clicking on graphic elements) and in terms of the grouping of graph design concepts (a single characteristic of both axes versus all characteristics of one axis).

Phase 3: Quasi-Experimental Study of Different Versions of Software

Once the basic design of our educational graphing program had evolved to the satisfaction of the participating teachers and students, we created three systematic variations of *CAPGraph* representing different approaches to the issues of flexibility and feedback. Some sample screens can serve to illustrate the operation of each of these versions.

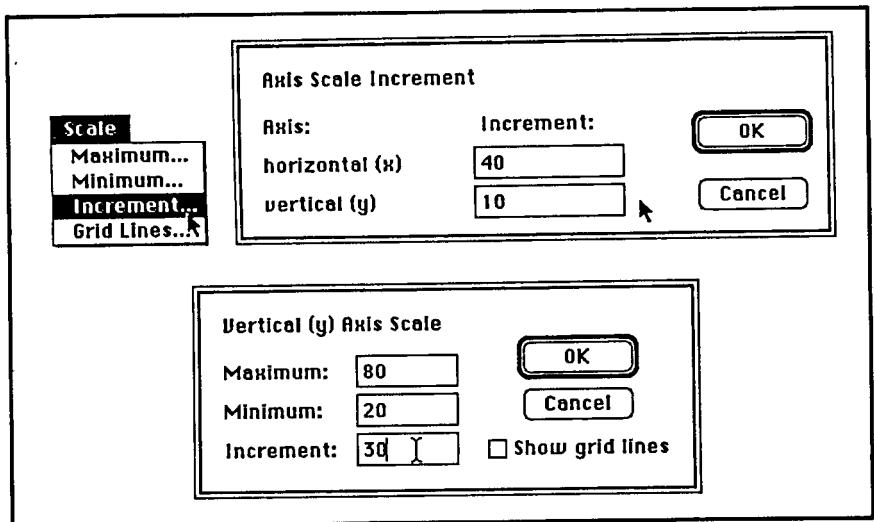


Figure 8. Multiple-mode interface design (*CAPGraph*): individual menu commands and separate dialogs (top) suggest isolated changes in scale parameters, with the resulting effects on the graph display available for evaluation between commands. An alternative (hidden, more efficient) format is also available (bottom; cf. Figure 7).

The "restrictive" version (Figures 9 and 10) precludes many common mistakes. The more flexible versions allow students to experiment freely with all graphing options, including those that would clearly be inadvisable according to conventional wisdom. One of these operates in a completely "open" manner with the software providing no explicit feedback (Figure 11), and another uses "coaching" messages to gently remind students of the assumptions and implications of their actions in constructing and modifying graphs (Figure 12).

The overall effects of each interface on student learning were evaluated based upon scores on a test of graph-reading and graph-design skills which we

New Graph Title: Scatter Graph of Weather

Horizontal Axis Values:	Vertical Axis Values:	Graph Type
Date	Date	<input type="radio"/> Column
Air Pressure (in Hg)	Air Pressure (in Hg)	<input type="radio"/> Line
Precip. Amount (in)	Precip. Amount (in)	<input type="radio"/> Pie
Precip. Type	Precip. Type	<input checked="" type="radio"/> Scatter
High Temp. (°C)	High Temp. (°C)	
Low Temp. (°C)	Low Temp. (°C) (frequencies)	

Buttons: OK, Cancel

New Graph Title: Column Graph of Weather

Column Categories:	Column Height Values:	Graph Type
Date	Date	<input checked="" type="radio"/> Column
Air Pressure (in Hg)	Air Pressure (in Hg)	<input type="radio"/> Line
Precip. Amount (in)	Precip. Amount (in)	<input type="radio"/> Pie
Precip. Type	Precip. Type	<input type="radio"/> Scatter
High Temp. (°C)	High Temp. (°C)	
Low Temp. (°C)	Low Temp. (°C) (frequencies)	

Buttons: OK, Cancel

Figure 9. The "restrictive" version of the *CAPGraph* interface: only numeric variables can be included in scatter graphs (top). Only categorical variables may designate column names (bottom).

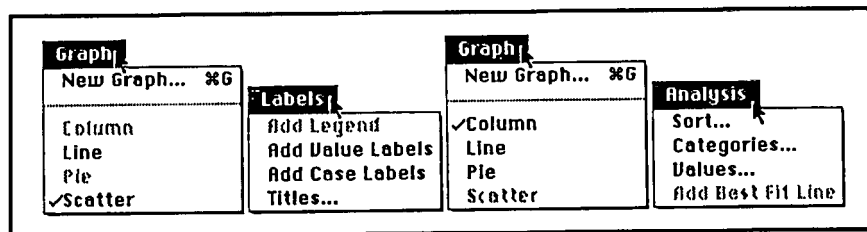


Figure 10. The "restrictive" version of the *CAPGraph* interface: examples of disabled menu choices in the context of different graph types and the nature of different variables used.

developed to reflect the priorities implied by our teaching methods. Students were presented with examples of graphs intended to convey the answer to a given research question. Their tasks for each of sixteen test items were, first, to answer the question as best they could (or indicate that it was not possible) and, second, to describe how the graph could best be improved (Figure 13). Graphs included in the test items were designed to range from highly appropriate to seriously flawed. Face validity of the test was established through review and editing by at least one teacher from each school. The sixteen items used were chosen from among thirty candidate items written by the authors.

Reliability of the test was estimated as .78 by Cronbach's alpha applied to tests from all students and .72 based on test-retest correlation (identical tests, 10 days' delay) for students at one school.

Although participating whole classes were randomly assigned to use one of the three interfaces, several factors complicated the interpretation of this quantitative data. We did not have access to confidential background information such as grade-point averages or standardized test scores for individual students. The grade level and gender of each student was included on the teachers' daily attendance sheets, and we were given copies.

Students in the sample represented a wider than anticipated range of ages and grade levels. Normally, most students in the Detroit CAP classes are in the ninth and tenth grades. For most of the classes at one of the schools, a majority of the students were in the eleventh and twelfth grades. Statistical control for these differences was considered inadvisable for two reasons: first, the effect of grade level on test scores was not linear—tenth-graders performed marginally better than ninth-graders, while participating eleventh- and twelfth-graders had substantially lower scores; second, and perhaps explaining the first problem, participating teachers informed us that official grade-level information for many of these students is often highly misleading due to "social promotion" practices.

Absenteeism was also a serious concern at the participating schools, and its effect on test scores was also not linear. The greatest discrepancy was found

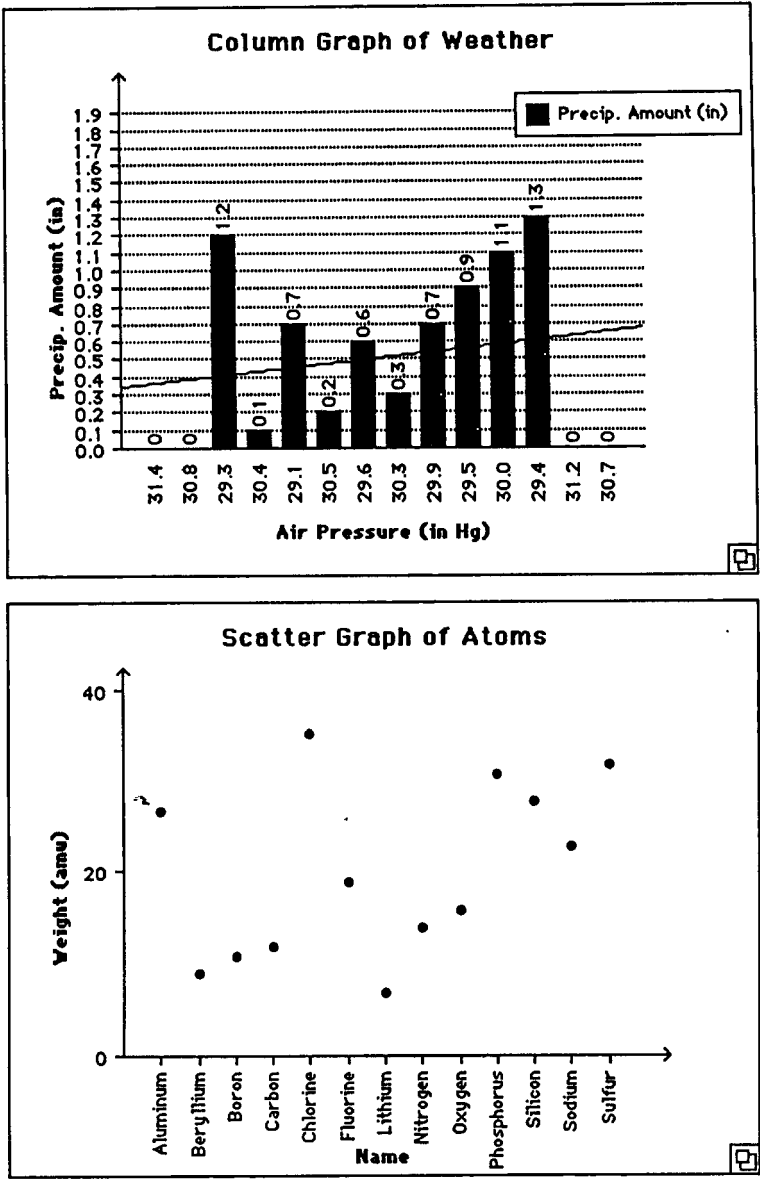
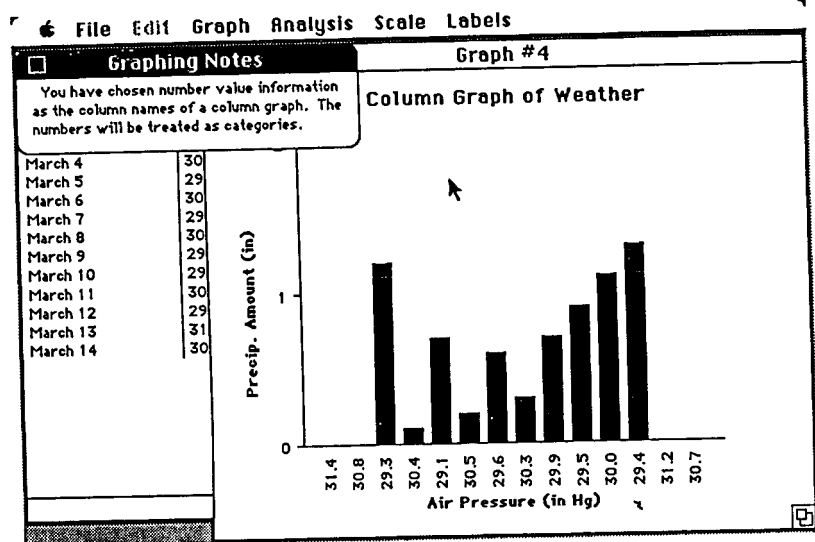


Figure 11. Examples of graphs possible with “open” version of the *CAPGraph* interface: column graph with numeric categories, useless legend, and implicitly ordinal best fit line (top). Scatter graph including categorical data, with meaningless apparent “trend” (bottom).



☐ **Graphing Notes**
 You have chosen category information as the horizontal axis sequence of a line graph. The ordering of the sequence should make some sense.

☐ **Graphing Notes**
 You have chosen a scatter graph when the horizontal axis shows category information. Position on this axis is normally based on number values.

☐ **Graphing Notes**
 You have chosen more than one kind of information as vertical axis values. To compare values fairly, all should be measured in the same units.

☐ **Graphing Notes**
 A best fit line makes some sense for this column graph only if the ordering of the categories makes some sense.

☐ **Graphing Notes**
 This graph now shows both value labels and grid lines. Make sure they do not get in the way of each other.

☐ **Graphing Notes**
 Case labels are the same as category names on this graph.

Figure 12. Samples from the "coaching" version of the CAPGraph interface: a view of the Macintosh screen showing the placement of the data spreadsheet, graph, and coaching windows on screen (top). Further examples of coaching messages (bottom).

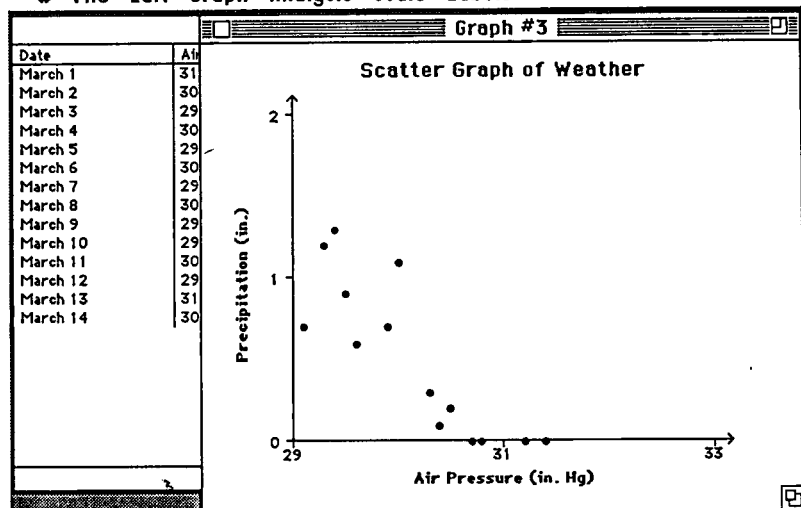
Instructions:

You may use the reference sheet (listing the functions of the application) during this test.

For each problem,

1. If it is possible to answer the question(s) from the graph shown, answer them as best you can. If it is not possible, just say so.
2. Remembering what you can do with *CAPGraph*, describe any changes you would make to each graph in order to make it better for answering the question(s) given in the problem. These could be either major changes (like "use a column graph instead") or minor changes (like "add a legend" or "remove the best fit line"). If the graph needs no changes, just say so.

File Edit Graph Analysis Scale Labels

**Problem 3.**

What amount of precipitation would you expect on a day when the air pressure measures 30?

Figure 13. Instructions and sample item from test of graph-reading and graph-design skills.

between students who were present for at least five of the six instructional days and those who were not.

In an effort to maintain a systematic sample of comparable students who received sufficient exposure to the software, we decided to eliminate from the following statistical analyses the eleventh- and twelfth-grade students and those who were absent more than once. This reduced our sample size by 41 percent to 219 students. Their distribution across the different treatment groups remained tolerably even.

The quasi-experimental design, large sample size, and tolerably normal frequency distributions of the test scores justify using simple and traditional statistical methods for these data. In addition to compiling descriptive statistics, we used a two-way ANOVA to investigate the main effects of the instructional treatments and of the gender of the students, and any interactions between these two factors. Detailed results of these analyses are shown in Tables 2 and 3.

Students using the "coaching" interface for the problem-solving sessions scored significantly higher on the graphing test than those using either the "open" or "restrictive" interfaces. Female students scored significantly higher than their male classmates. There was, however, a significant interaction between these two variables. Although students using the "coaching" interface scored higher than those using the other interfaces regardless of gender, the apparent effect is much more pronounced for male students. Although the young women performed better on the test regardless of which version of the program their class used, this difference was heavily concentrated in the group who used the flexible interface with no explicit feedback.

DISCUSSION AND FUTURE DIRECTIONS

These effects are statistically significant, but how much do they really mean for software design? In the same spirit in which the Detroit high school students were taught about graphs, we present a short object lesson in how decisions about graph design and modification can affect the substantive message of a given set of data.

For instance, the mean test score for students in classes that used the coaching interface was 30 percent higher than for those that used the restrictive interface (Figure 14, top)—probably enough of a difference to get some teachers or software designers to sit up and take notice. Many of those students could tell us that a column graph is most appropriate for this information. It gives a fair, intuitive idea of the magnitude of the differences among three qualitatively

Table 2. Descriptive Statistics

Test Scores	<i>n</i>	Mean	S.D.
Treatment:			
open	75	7.57	3.83
coaching	78	9.37	3.35
restrictive	66	7.23	4.06
Gender:			
female	105	8.70	3.98
male	114	7.57	3.64
Total	219	8.11	3.84

Table 3. Comparative Statistics

Test Scores	<i>n</i>	Mean
Two-way breakdowns:		
Open, female	36	9.08
Open, male	39	6.17
Coaching, female	35	9.49
Coaching, male	43	9.28
Restrictive, female	34	7.47
Restrictive, male	32	6.97
Total	219	8.11

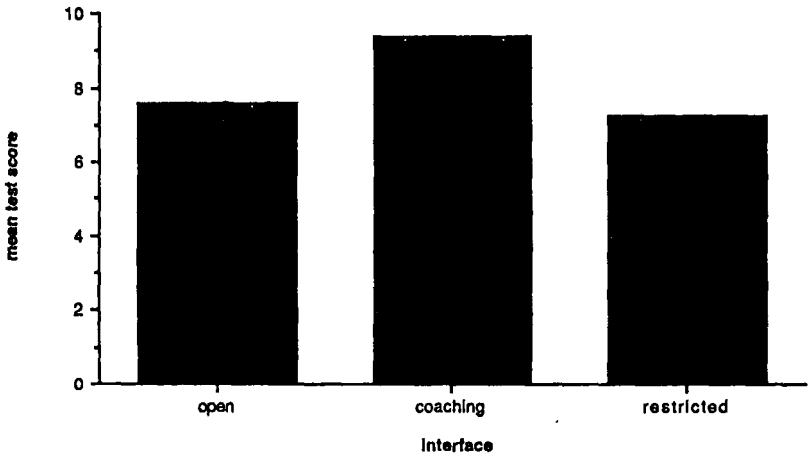
Source	<i>DF</i>	Sum of Squares	Mean Square	<i>F</i>	<i>p</i>
Two-way analysis of variance table:					
Interface	2	194.79	97.39	7.27	<.001
Gender	1	79.21	79.21	5.91	<.05
Interaction	2	82.20	41.10	3.09	<.05
Error	213	2853.25	13.40		

Comparison	Mean Difference	Scheffé <i>F</i>	<i>p</i>
<i>Post hoc</i> pairwise comparisons:			
Open vs. coaching	-1.81	4.46	<.05
Open vs. restrictive	0.34	0.15	NS
Coaching vs. restrictive	2.15	5.89	<.05

different categories, corresponding to three different groups of people using the software at one time.

If we had clicked on just one different button when commanding the computer to graph the data for this article, we would have ended up with a different type of graph (Figure 14, bottom) which, based on our experience, almost certainly would be preferred according to the style guidelines of many professional journals, but which also has the potential to be extremely misleading. Here is a very reasonable, at-a-glance interpretation of this line graph: students didn't learn very well, at first, with the open-ended interface, but jumped way up to the top of the scale when exposed to the coaching feedback, then showed a precipitous drop in performance when new restrictions were placed on them. In all fairness, the mere removal of the lines connecting the data points might change this interpretation. A cryptic but authoritative-sounding title using theoretical jargon is thrown in for good measure.

Effectiveness of 3 Software Interfaces



"Cognitive Residue" of CAPGraph Interfaces

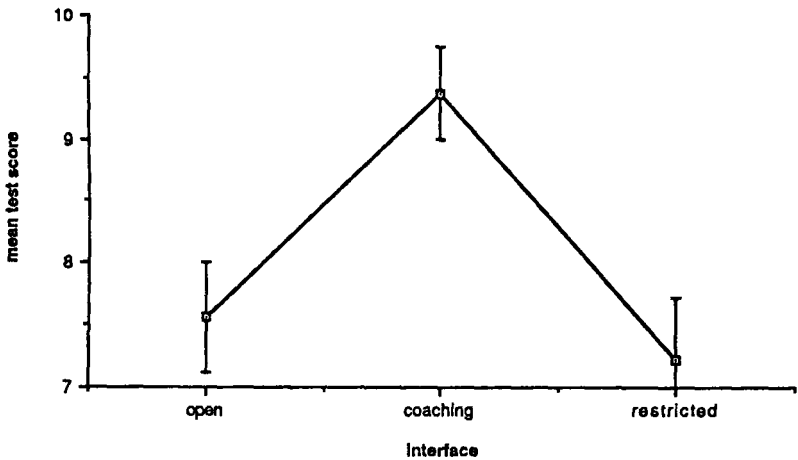


Figure 14. Effect on test scores of working with different versions of *CAPGraph*: an honest graphic representation of the data (top).

Use of a line graph for such a display, as many research reports are wont to do (bottom).

The very basic graph showing that the females scored somewhat higher than the males (Figure 15, top) seems to be lacking in precision and is not very visually striking. Using the computer and *CAPGraph*, about five seconds later it can effectively, but incorrectly, be "shown" that the girls performed more than twelve times better than the boys (Figure 15, bottom).

There were barely any gender differences, in fact, except in the case of the open-ended interface. This interaction clearly stands out when presented graphically (Figure 16, top). Again, a single computer-use choice, perhaps made in the name of using a consistent form for the graphs, could make it look as if the most impressive result is that the young men "improved" because they responded especially well to the computer coaching.

Are these examples unreasonable caricatures? Perhaps, but there is more than a grain of truth in their message, for educational researchers as well as for high school science students. Our anecdotal observations and unstructured interviews with students can provide some insight into what generated these differences in learning. Not surprisingly, some students reacted negatively to being provided with neither stepwise written guidelines nor explicit feedback from the software. A substantial number, especially among the relatively few who worked alone at the computer, were never able to develop an ability to evaluate whether the graphs-under-construction on their screen needed wholesale reorganization or merely small improvements. The open-ended interface overwhelmed them with too many powerful choices and too little assistance in dealing with them.

The coaching messages seem to have helped, but there were relatively few cases in which we observed students taking the time to read them carefully. Most often, the immediate response to the appearance or reappearance of the coaching window was to get rid of it or move it into the background. Most students indicated that they took the messages as *general* indicators that their most recent action might be questionable, thus serving the important purpose of suggesting that they critically examine their work.

The restricted interface was the only one that generated a significant number of vocal complaints. Many students were extremely frustrated with being unable to change a particular graph parameter to evaluate its effect for themselves. Teachers were also at a loss to give illustrative explanations to these students, because instructive examples of inadvisable actions were rendered impossible. It makes intuitive sense to us that these students, who were not allowed to make many major categories of mistakes, could not be expected to learn from them.

From our wealth of classroom experience with this teaching approach and this software, we can offer no explanation whatsoever for the observed gender differences and gender-treatment interaction. Neither we nor the participating teachers ever noticed any prominent cases in which the young men or women as groups might have been reacting differently to either the teacher presentations or to the problem-solving experience with the graphing software. Initially, we had a few misgivings about using a sports-oriented data file (professional basketball

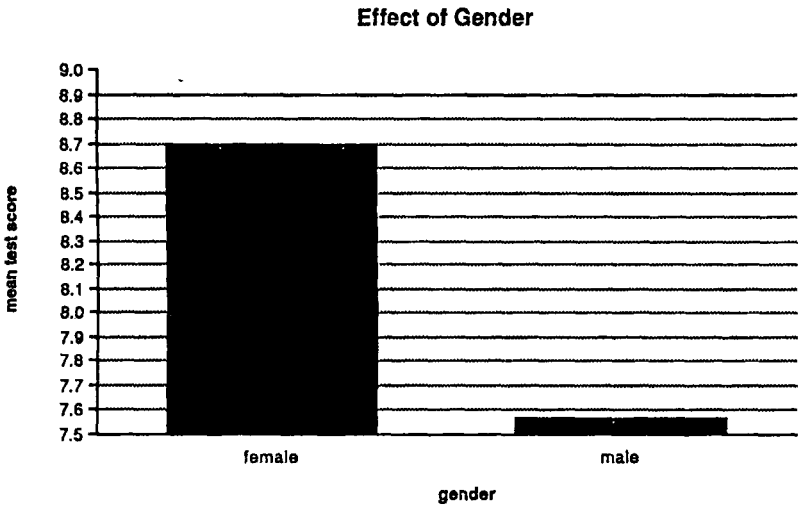
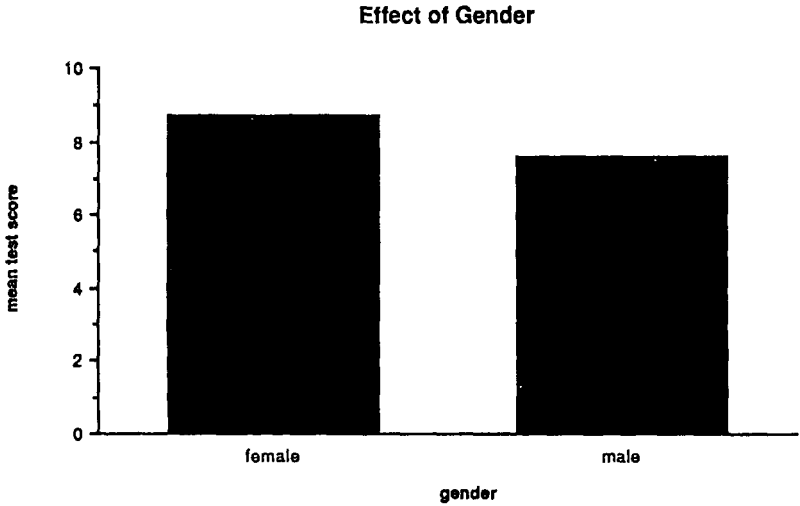


Figure 15. Effect of gender on test scores: an honest graphic representation of the data (top). Using the power of the computer to easily manipulate the scale, visually distorting the relative magnitudes of the numeric data and emphasizing unnecessary precision (bottom).

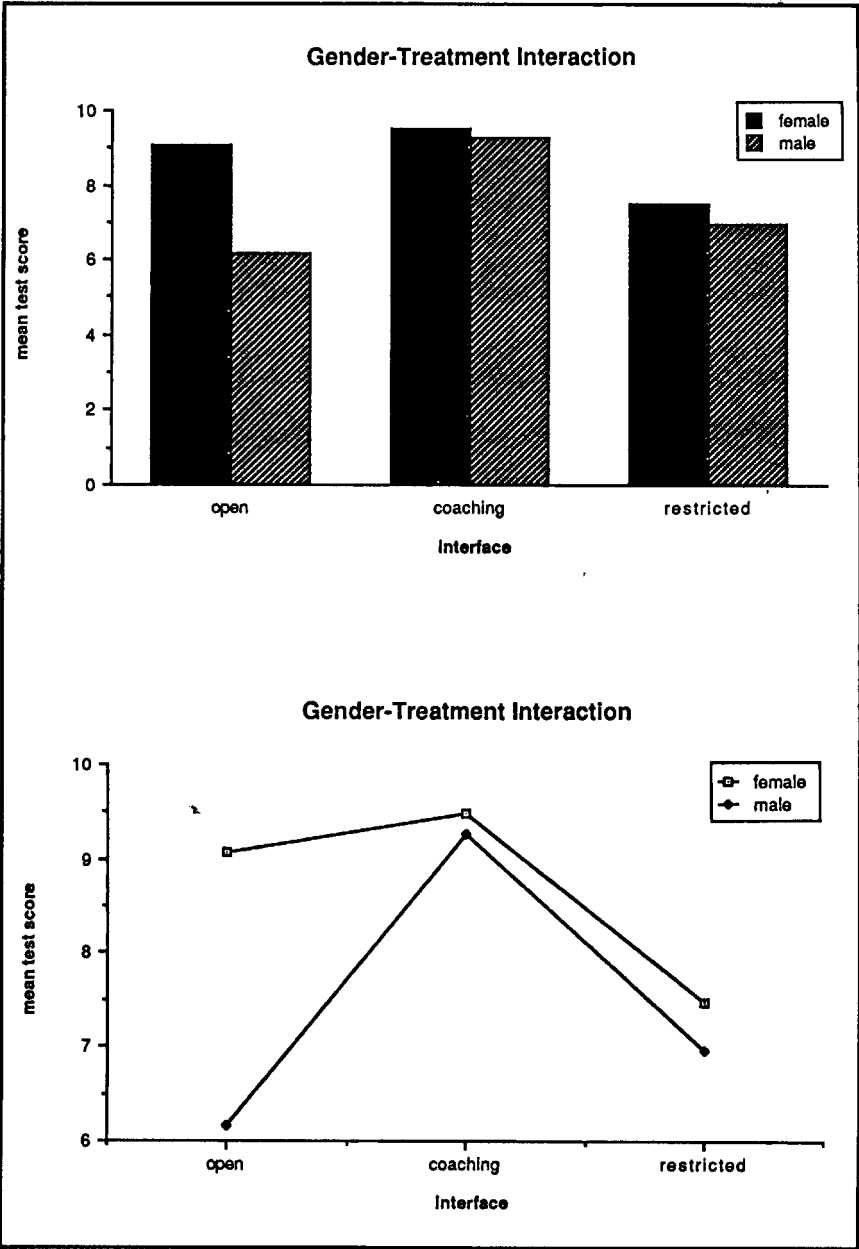


Figure 16. Interaction effect: an honest graphic representation of the data (top). A line graph intuitively implying at least an ordinal scale for the treatment variable (bottom).

statistics) for many of the whole-class demonstrations and models, but all of the teachers assured us that its motivational value outweighed any tendency for the males to be more knowledgeable basketball fans than the females. These results cannot refute that opinion.

An obvious issue for future study is that of *aptitude*-treatment interactions. Will the general consensus in education that more flexibility and less direct feedback is better suited to higher-achieving students hold up in the case of the use of tool software? Although relevant individual data were not available, achievement levels in this urban school district are comparatively low. Most of the teachers involved in this study were very pleasantly surprised by their students' ability to work productively with relatively sophisticated software in a less-structured task environment than is customary in these inner-city classrooms.

Another fruitful follow-up study might be to investigate the *sequential* use of more than one of these interface modes, perhaps beginning with the restrictive interface, then removing the restrictions and adding the coaching functions, and finally using the computer and software in the context of a performance-based test, using the flexible interface with the coaching functions no longer available. Returning to the notion of "artificial intelligence in reverse" and the descriptive terminology that Salomon applies to the capabilities of tool software [27], the extent to which a student has internalized the knowledge of graphing embodied in the computer program could only be determined when the tool's guidance (coaching) and modeling (restrictive interface), but not its enabling features (the substantive functions of the program itself), have been taken away (as in the open-ended interface).

REFERENCES

1. M. R. Lepper and J. Gurtner, Children and Computers: Approaching the Twenty-First Century, *Educational Psychologist*, 44, pp. 170-178, 1989.
2. R. P. Taylor (ed.), *The Computer in the School: Tutor, Tool, Tutee*, Teachers College Press, New York, 1980.
3. Office of Technology Assessment, U.S. Congress, *Power On! New Tools for Teaching and Learning*, Publication No. OTA-SET-379, U.S. Government Printing Office, Washington, D.C., 1988.
4. M. E. Lockheed, J. P. Gulovson, and D. Morrison, *Student Use of Applications Software*, Technical Report 85-3, Educational Technology Center, Cambridge, Massachusetts, 1985.
5. J. M. Carroll and C. Carrithers, Blocking Learner Error States in a Training-Wheels System, *Human Factors*, 26, pp. 377-389, 1984.
6. D. Frye and E. Soloway, Interface Design: A Neglected Issue in Educational Software, in *CHI + GI 1987 Conference Proceedings: Human Factors in Computing Systems and Graphics Interface*, J. M. Carroll and P. P. Tanner (eds.), Association for Computing Machinery, New York, 1987.

7. E. P. Goldenberg, *The Difference Between Graphing Software and Educational Graphing Software*, Educational Development Center, Inc., Newton, Massachusetts, 1989.
8. J. Dewey, *Democracy and Education*, Macmillan, New York, 1916.
9. J. Dewey, My Pedagogic Creed, in *Dewey on Education*, M. S. Dworkin (ed.), Teachers College Press, New York, pp. 19-32, 1959.
10. D. F. Jackson, B. J. Edwards, and C. F. Berger, Teaching the Design and Interpretation of Graphs Through Computer-Aided Graphical Data Analysis, *Journal of Research in Science Teaching*, in press.
11. J. S. Bruner, Notes On A Theory of Instruction, in *Toward a Theory of Instruction*, Harvard University Press, Cambridge, Massachusetts, 1966.
12. J. Dewey, *Experience and Education*, Macmillan, New York, 1938.
13. S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, 1980.
14. D. N. Perkins, The Fingertip Effect: How Information-Processing Technology Shapes Thinking, *Educational Researcher*, 14:7, pp. 11-17, 1985.
15. P. Suppes, The Uses of Computers in Education, *Scientific American*, 215, pp. 206-221, 1966.
16. C. Holden, Computers Make Slow Progress in Class, *Science*, 244, pp. 906-909, 1989.
17. G. Fischer and A. C. Lemke, Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication, *Human-Computer Interaction*, 3, pp. 179-222, 1987-88.
18. M. Polanyi, *The Tacit Dimension*, Doubleday, Garden City, New York, 1966.
19. T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Ablex, Norwood, New Jersey, 1986.
20. D. Laurillard, Computers and the Emancipation of Students: Giving Control to the Learner, *Instructional Science*, 16, pp. 3-18, 1987.
21. A. A. DiSessa, Artificial Worlds and Real Experience, *Instructional Science*, 14, pp. 207-227, 1986.
22. D. H. Clements, Effects of Logo and CAI Environments on Cognition and Creativity, *Journal of Educational Psychology*, 78, pp. 309-318, 1986.
23. S. Papert, Computer Criticism vs. Technocentric Thinking, *Educational Researcher*, 16:1, pp. 22-30, 1987.
24. P. R. Pintrich, C. F. Berger, and P. M. Stemmer, Students' Programming Behavior in a Pascal Course, *Journal of Research in Science Teaching*, 24, pp. 451-466, 1987.
25. A. H. Schoenfeld, *Mathematical Problem Solving*, Academic Press, New York, 1985.
26. J. Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation*, Freeman, New York, 1976.
27. G. Salomon, AI in Reverse: Computer Tools That Turn Cognitive, *Journal of Educational Computing Research*, 4, pp. 123-129, 1988.
28. L. Vygotsky, *Thought and Language*, MIT Press, Cambridge, Massachusetts, 1986.
29. H. Hamburger and A. Lodgher, Semantically Constrained Exploration and Heuristic Guidance, *Machine-Mediated Learning*, 3, pp. 81-105, 1989.
30. D. F. Jackson, C. F. Berger, and B. J. Edwards, Computer-Assisted Thinking Tools: Problem Solving in Graphical Data Analysis, *Journal of Educational Computing Research*, 8, pp. 43-67, 1992.

31. R. D. Pea, Cognitive Technologies for Mathematics Education, in *Cognitive Science and Mathematics Education*, A. Schoenfeld (ed.), Erlbaum, Hillsdale, New Jersey, pp. 89-122, 1987.
32. F. L. Goodman, An Antonym for "Feedback," *Educational Theory*, 13, pp. 105-107, 118, 1963.
33. P. Baggett, Understanding Visual and Verbal Messages, in *Acquiring Knowledge From Text and Picture*, H. Mandle and J. Levin (eds.), North Holland, New York, pp. 101-124, 1989.
34. R. E. Clark, Confounding in Educational Computing Research, *Journal of Educational Computing Research*, 1, pp. 137-148, 1985.
35. C. F. Schmid, *Statistical Graphics: Design Principles and Practices*, Wiley, New York, 1983.
36. D. Hawkins, Messing About in Science, in *The Informed Vision*, Agathon Press, New York, 1974.
37. E. R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983.

Direct reprint requests to:

Dr. David Jackson
 Science Education Department
 University of Georgia
 212 Aderhold Hall
 Athens, GA 30602