Learner-Centered Design

The Challenge For HCI In The 21st Century

Elliot Soloway, Mark Guzdial and Kenneth E. Hay

In the 1980's a major transformation took place in the computing world: attention was finally being paid to making computers easier-to-use. You know the history: in the 1970's folks at Xerox were exploring so-called personal computers and developing graphical, point-and-click interfaces. The goal was to make using computers less cognitively taxing, thereby permitting the user to focus more mental cycles on getting the job done. For some time people had recognized that there would be benefits if users could interact with computers using visual cues and motor movements instead of textual/linguistic strings. However, computer cycles were costly; they could hardly be wasted on supporting a non-textual interface. There was barely enough zorch (i.e., computer power, measured in your favorite unit) to simply calculate the payroll.

About the Authors

ELLIOT SOLOWAY *is an Associate Professor in the Department of EECS, University of Michigan, Ann Arbor, MI, email: soloway@umich.edu*

KENNETH E. HAYS *is an Assistant Professor in the School of Education at Indiana University & Purdue University, Indianapolis, IN., email: khay@indyvax.iupui.edu*

MARK J. GUZDIAL *is an Assistant Professor, Georgia Institute of Technology, College of Computing, Atlanta, GA., email: guzdial@cc.gatech.edu*

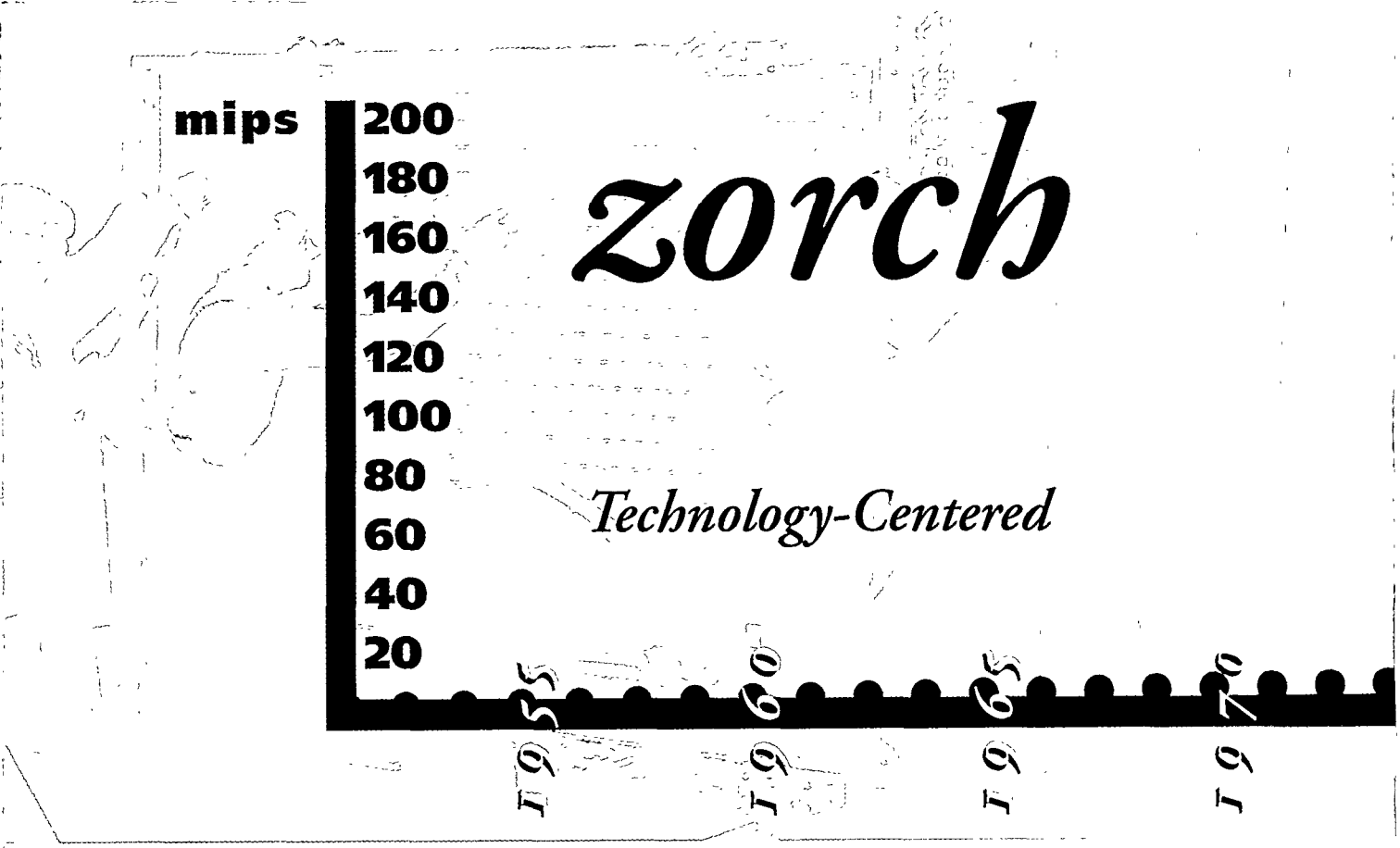


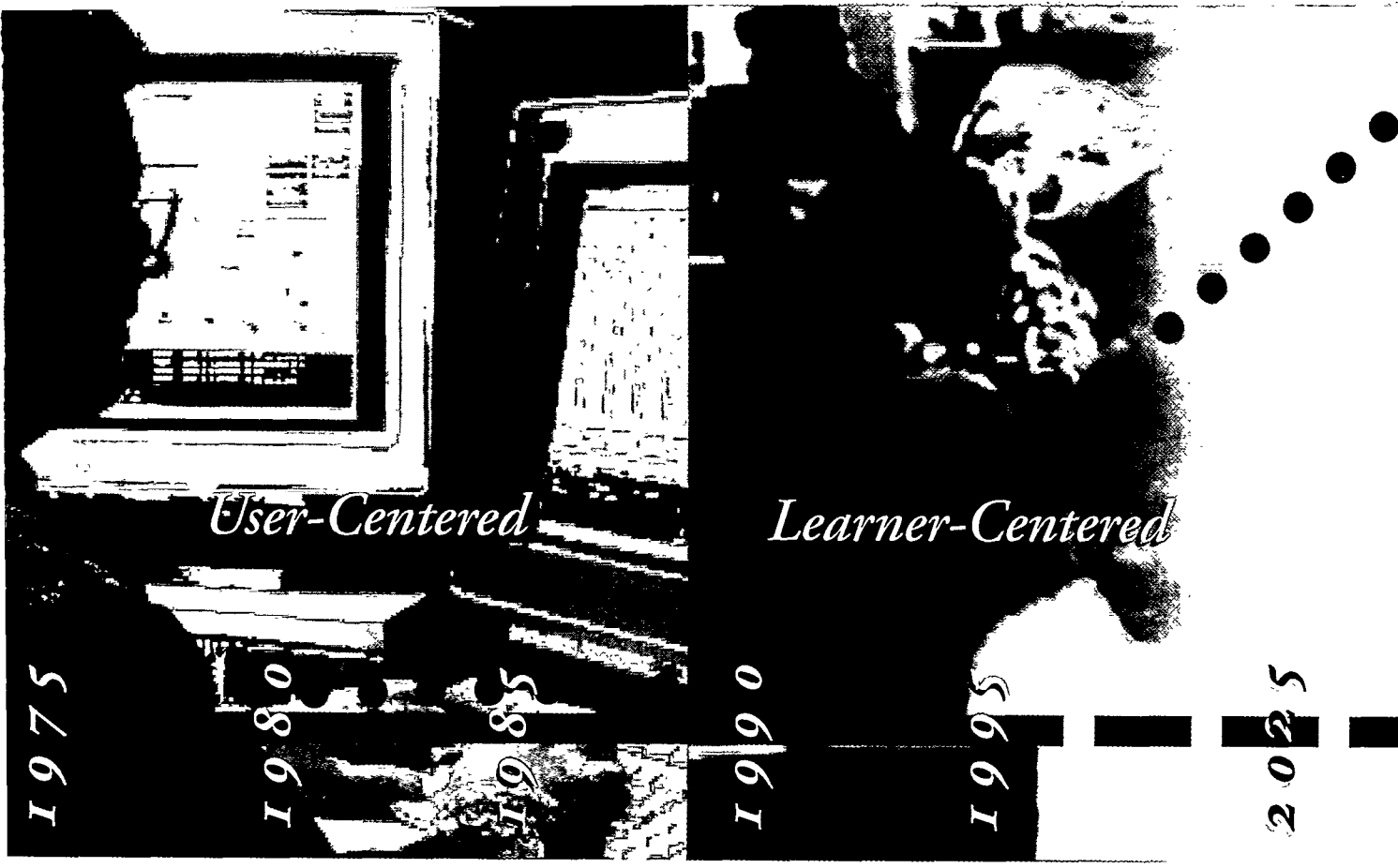
Figure 1
Plotting the Computer Zorch Curve Against HCI Goals

The Xerox researchers were ahead of their time. It wasn't until the 1980's that sufficient zorch–at almost consumer prices–was available to permit the computer to do its work and still have enough power left over to support an easier to use graphical user interface. With the publication of several seminal books (e.g., Card, Moran, and Newell's "Human Computer Interaction" in 1983, and Norman and Draper's "User-Centered Systems Design" in 1985) and the grand successes of the early ACM CHI conferences (e.g., Gaithersburg in 1983), the Human-Computer Interaction movement shifted into high gear.

The early interface researchers dared to imagine what using a computer could be like if the computer were truly more powerful. And we are in an analogous position today. The 500 MIP notebook-sized machine is just around the corner. What could using such a computer be like? Of course, *one* goal should still be to use that increased zorch to make computers even easier to use, in as many ways as possible, e.g., from even more clever visual, gestural, auditory interfaces to intelligent agents. However, is ease of use the *only* need? The answer is a resounding NO! The challenge, as posed by Buxton (1991) for HCI is how to support individuals and groups of individuals in developing expertise in their professions, in developing richer and deeper understandings of content and practices. Making people smarter is really the long-term goal of computing.

It is our position that computing technologies are finally reaching the point where the HCI community can address the "make people smarter" (Norman, 1993) challenge. Consider the graph in Figure 1. In the early days of computing, interface design was driven by concern for over-taxing the technology. In the 1980's, a favorable cost/performance ratio for computing afforded us the opportunity to be concerned with over-taxing the computer user. The cost/performance ratio of computing in the second half of the 1990's–and beyond–will permit even more cycles to be devoted to the interface. What function, then, can the interface serve? Simply put, the HCI community must make another transition: we must move from "user-centered" design to "learner-centered" design. In what follows, we address three key questions.

- Why support learners and learning?
- How might the interface support learners and learning?
- What are the issues involved in providing such support?



Why Support Learning and Learners?

Who is a learner? After all, a professional is supposed to be an expert. Table 1 provides a characterization of the traditional differences between students and professionals. For example, at first blush one might reasonably assume that a professional accountant would understand the principles underlying a spreadsheet, but one can not assume that same understanding when introducing spreadsheets to students in introductory accounting. Moreover, one might assume that a professional accountant will be motivated to persevere in learning to use a spreadsheet; again, a student's motivation, and hence the student's perseverance, is very much in question.

But learning is not just for students in classrooms! Professionals are–should be–constantly learning, e.g., a professional accountant may not understand the power of forecasting methods. Moreover, when the professional is acting as a learner, that person is susceptible to all the challenges faced by students. For example, the professional may be unable to stop staring out the window when faced with the process of learning the forecasting tool. In sum, professionals are students who happen to learn outside of a classroom. In fact, companies are coming to realize that investing in the growth and development of their people is good business. The notion of the "learning organization" (Senge, 1990) is a provocative and compelling idea in management these days: first, the value of a company is directly related to how deeply its employees understand their business, and secondly, effectively competing in today's marketplace requires that this understanding continues to grow and change. The company that is successful, time after time, is the company that learns from its successes and mistakes, that deals effectively with the many changes in the marketplace, that can nimbly respond to opportunities, etc. And, who is "the company" –it's people; it's individuals from increasingly diverse backgrounds. The need to support

Table 1

Traditional Differences Between Professionals and Students

Nature of Professionals	Nature of Student
Do Know Domain	Do NOT know domain
Are motivated	NOT motivated
Homogeneous populations	Diverse populations
Growth IS the issue	Growth is not an issue

learning and learners in the workplace of the 21st century is clear.

Fortunately, the intensive use of computers in the workplace sets up just the right conditions to effectively support learning and learners, thus there are good reasons to be optimistic that computers and interfaces can be successful in providing such support. That is, computers are coming to be used on a moment-by-moment basis for all aspects of work. In knowledge-intensive industries, personal computers are ubiquitous; standard issue to a new employee is a desk, a phone and a computer linked into the company's network.

Given that computers are being used for doing one's job, there is a clear opportunity to use those same computers supporting learning as one is engaged in "doing." While not necessarily widely practiced in schools, the effectiveness of the educational philosophy of "learning by doing" is widely acknowledged. Plato, Dewey, your mother–they all knew the value of learning from direct experience, from on-the-job training.

Table 2
Addressing the Needs of Learners

Nature of Learners	Scaffolding Strategies for Learner-Centered Designed Software	Example
Understanding Is the Goal	Modeling, coaching, critiquing	GPC Editor
Motivation Is The Basis	"Low overhead, Immediate success" use	MediaText
Diversity is the Norm	Collections of various techniques	MediaText, ScienceWorks
Growth Is the Challenge	Adaptable	Emile

Why is learning by doing so effective? When one really cares about what one is doing, one seems able to muster the energy, the attention, the patience to master whatever is necessary to achieve the goal. However, when learning is divorced from doing a meaningful task–as are many arbitrary, decontextualized activities in the classroom–then learning becomes just another chore, low on the priority stack. Performing well on the job is a motivating goal; walking through a cookbook set of steps to find Avogadro's Number in the chemistry lab, on the other hand, may not engender much enthusiasm, and hence not much genuine learning.

Why devote computer zorch to the interface to support learning and learners? Answer: **there is a clear need** to support students and professionals in developing their expertise. And, **there is a clear opportunity for success;** the way in which computers are being used in the workplace provides precisely the right conditions under which learning should take place, namely, learning in the context of doing.

User-Centered Design to Learner-Centered Design

In designing a software environment, there are three top-level issues that must be addressed:

- Tasks: What tasks need to be undertaken in the software?
- Tools: What tools are provided to cope with those tasks?
- Interfaces: What is the interface to those tools?

The insight offered by the user-centered design movement (Norman & Draper, 1985) was that the user needed to be at the center of those issues (see Figure 2a).

In putting learners at the center of the design, however, the special needs of learners must be addressed:

- **Understanding is the Goal:** Learners will not know accounting principles or practices when a spreadsheet is presented to them. How will they learn to use that spreadsheet?
- **Motivation is the Basis:** One can not count on the motivation of learners: both students and professionals have a strong tendency to procrastinate, to fritter away time, when confronted with a task for which they are unprepared. Why can't software play a role in supporting the learner's wavering motivation?
- **Diversity is the Norm:** Classrooms and professions are composed of individuals from a diverse set of backgrounds, with a diverse set of interests, skills and abilities. How can an application be "one size fits all"?
- **Growth is the Challenge:** A spreadsheet is by and large the same on day 1 as it is on day 100. But an individual can be very different, e.g., that person may have learned quite a bit

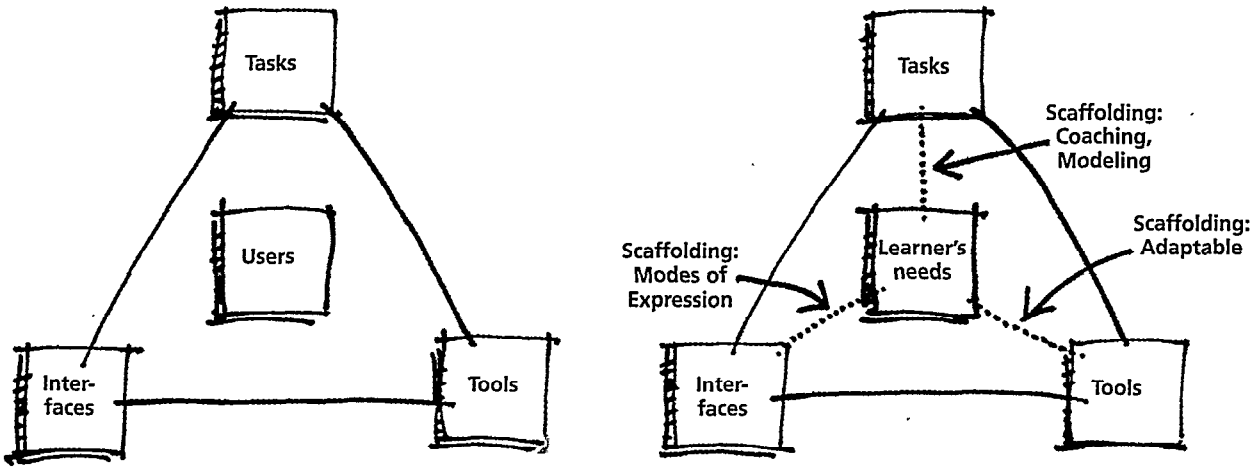


Figure 2
a (left) Tasks–Tools–Interfaces (TTI) user-centered model
b (right) Tools–Interfaces–Learner's Needs–Tasks (TILT) learner-centered model

about a problem domain and might have developed a set of skills and practices in that domain. The individual has changed but the software hasn't.

Tools-Interfaces-Learner's Needs-Tasks (TILT)

Our suggestion is that the soon-to-be-available zorch can permit us to put the *learner* at the center (Figure 2b)–and provide mechanisms that can address their special needs. Hence, we have proposed the TILT Model (*T*ools, *I*nterfaces, *L*earner's needs, *T*asks) to guide the design of learner-centered software: the objective of the model is to highlight how software might address the special needs of the learner.

In education, scaffolding is a technique for providing support to learners while they are learning a new task (Wood, Bruner, Ross, 1975; Rogoff, 1990). For example, as a student begins a new task, say, calculating a set of numbers, a teacher may scaffold the student by stopping the student by making helpful hints or correcting mistakes. The key is that the student is doing the task and the teacher provides structure and guidance to ensure successful completion of the task. As the student gains expertise, a good teacher provides less and less coaching. Scaffolding, then, is provided to help a learner do a task that he or she can not do alone; as the learner develops the needed knowledge and skills, the scaffolding fades so that the learner is fully in control.

The TILT Model (Figure 2b) identifies specific scaffolding strategies that are particularly appropriate for the special needs of the learner:

- Task: Coaching is a scaffolding technique that could be used to help students acquire knowledge and the specific practices of a task domain. This technique is popular in "intelligent tutoring systems (ITSs)" (Wenger, 1989).
- Tools: In order to support a learner growing in expertise, the tools must be adaptable.
- Interface: In order to enable learners to express themselves and communicate, the interface must scaffold the use of different media and modes of expression.

CAD for Kids, And Teachers, Too

Over the past 5 years, we have implemented, classroom tested, and even commercially sold various pieces of software that are designed to support learners (Guzdial et al., 1992a). But, in order to prepare the way for a description of that software and how it exemplifies the TILT Model, we need to first be clear about our educational philosophy.

We subscribe to the constructivist theory of learning espoused by Jean Piaget (1954), Lem Vygostky (1962), and Seymour Papert (1993). Students learn through an active, social process of meaning construction; understanding is built up through the acts of conversing with others, constructing artifacts, and reflecting on those conversations and artifacts. This theory of learning leads to a theory of teaching: students need to actively engage in projects, and teachers need to act as mentors, coaches, managers. A contrasting view–and one that is prevalent in classrooms today–is that learning is a copying process: what the student hears, the students knows. This theory of learning leads to

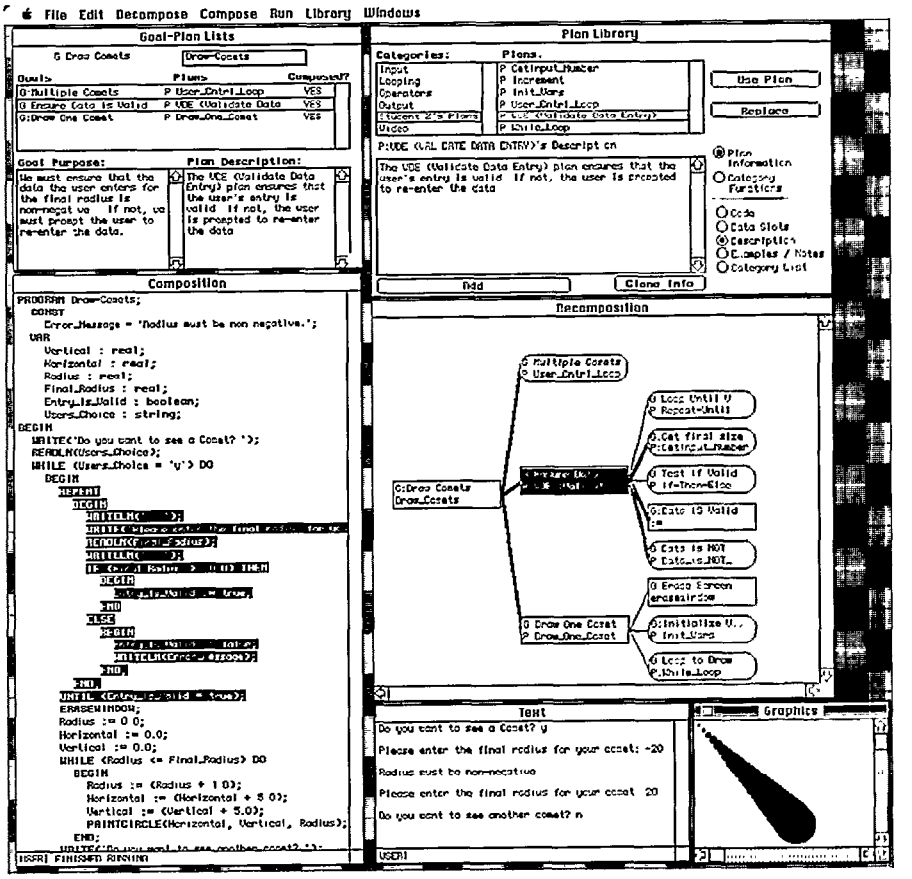


Figure 3
Student selecting a Valid-Data-Entry (VDE) Plan from the GPC Editor library

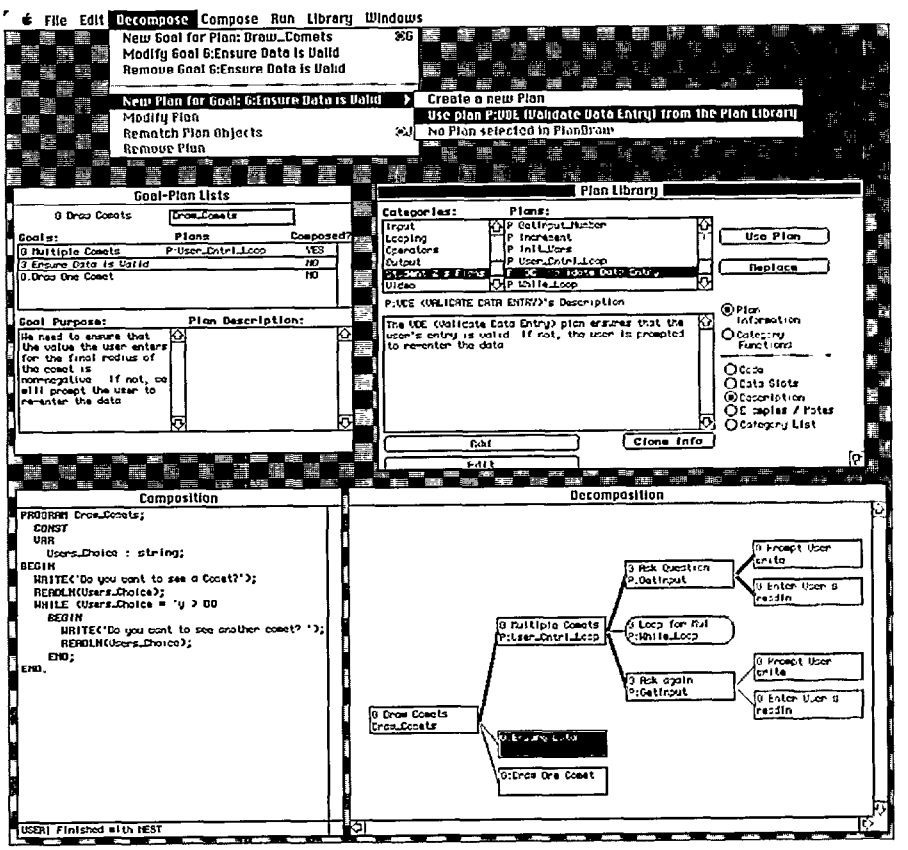


Figure 4
Using the GPC Editor Decompose function to identify the selected Valid-Data-Entry (VDE) Plan

the didactic theory of teaching: the teacher talks, students listen, and knowledge gets deposited directly into the gray matter. In effect, students are empty vessels that need to be filled. We apologize for over simplifying both positions. We do not apologize, however, for our skepticism of the "learning as information transmission" model.

In turn, the constructivist theory of learning suggests that students need software that supports them in constructing artifacts and conversing with others about those artifacts.

We view the construction of artifacts as a design process; and, inasmuch as professional designers routinely employ CAD systems in their design activities, we feel that learners deserve suitably scaffolded CAD support as well. And, since classroom teachers are also learners, and need the same kind of support as their students, we have developed scaffolded CAD tools for their tasks (e.g., project and instruction planning, Soloway et al., 1994). In our software, you will not see drill-and-kill software that teaches specific facts, motivating students by rewarding them with playing time on an arcade-style game. Rather, you will see scaffolded CAD environments where learners construct a wide range of artifacts.

In what follows we illustrate how our software uses scaffolding to address the needs of learners identified in Table 2.

Supporting the Learning of a Task: The GPC Editor

We developed the GoalPlanCode (GPC) Editor to support students learning elementary Pascal programming (Soloway, et al.,1993). Students have little trouble mastering an if-statement or variable assignment. But, they do have considerable difficulty in putting statements together in a coherent fashion to solve a problem. The GPC Editor supports students in carrying out this task in the following ways:

• **Change the nature of the task:** Traditionally, students use some sort of text editor to create programming statements and then run and debug the program in an execution environment. While introductory programming courses try to teach students to first plan their programs, there is little computer-based support for such planning. The upshot, quite reasonably, is that by and large students do not plan their programs.

In contrast, the GPC Editor provides an integrated environment that supports planning, generating, and testing. In fact, one never writes a line of code; rather, one selects plans from a library that achieve the specified goals. For example, in Figure 3, we see a student selecting the Valid-Data-Entry (VDE) Plan to achieve her goal of testing input data for validity. Notice the VDE Plan is a nice sized chunk of code.

Teachers are the gatekeepers of the classroom. They won't let something pass through the gate if they don't feel comfortable with it themselves...inasmuch as they feel comfortable with MediaText, it is permitted to pass through the gate into the hands of the students.

• **Guide student actions:** In creating a program, a student needs to carry out a range of actions. Using a scaffolding strategy we call "tight process control" the GPC Editor channels students to construct a program in a specific order. In particular, the GPC Editor scaffolds students to decompose a problem into goals and subgoals, identify code fragments (plans) from the library that achieve those subgoals, and then compose the plans together

into a program ready for testing. The decompose menu that directs subgoal decomposition-plan identification activities in the GPC editor is displayed in Figure 4.

The GPC Editor has been used for four years at Community High School in Ann Arbor, MI. We have found that students who would be considered "low achievers" (e.g., students in the bottom half of the class) produce computer programs of the same quality as students who would be considered "high achievers." Moreover, we found that students who moved off the GPC Editor and into Think Pascal wrote programs that were highly structured, GPC-like programs. And we found that students initially liked the tight process control, but by the middle of the semester they wished that it would fade away! Adaptive scaffolding, however, was not available in the GPC Editor; we learned our lesson, though, in the construction of Emile, described later.

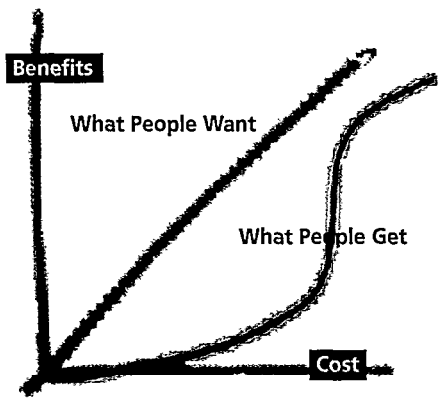


Figure 5
Typical learning curve for a new application is shown in red, but what people really want to experience is closer to the ideal learning curve shown in blue.

Interface Scaffolding For Wavering Motivation

Technology has been making it easier and easier for individuals to create text: from the quill to the pen to the typewriter to the word processor. If one is good with words, this technology is truly empowering. However, as Gardner (1993) argues, there are many types of intelligences and many types of literacies. The challenge for technology, then, is to go beyond crayons and the camera–no mean feat–so that individuals incur no penalty for "writing in" non-textual media.

There are a range of multimedia authoring tools on the market. However, the learning curve for mastering such tools is steep (Figure 5). A steep learning curve is not uncommon in technology-oriented contexts, and that makes it hard to stay motivated and persevere. People expect a benefit (the area above the blue line in Figure 5) commensurate with the cost, i.e., the blue area in Figure 5.

MediaText is a multimedia processor that has a learning curve that is closer to the blue line than it is to the red line. MediaText enables students (and, most importantly, their teachers) to create documents that include images, animations, video, sound and music, as well as text (Guzdial, et al. 1992b; Hay, et al., 1994). For example, Figure 6 depicts a portion of a document written by a student at Community High School. The margin on the right contains icons that display their contents when clicked. A vanilla word processor enables students to enter text to the left of the margin.

Why is MediaText so learnable? The interface scaffolding strategy adopted in MediaText is to mimic operations that users already know. MediaText employs a page metaphor where a person puts multimedia annotations in the margin. Since users already know how to manipulate marginalia from their experiences with ruled paper, the step to multimedia marginalia is not all that great.

Now, the first thing techies want to do when they see MediaText is move the icons out of the margin and integrate them into the text in a hypertext/hypermedia fashion. We have such a version in our lab; it's technically not complicated to remove that margin. However, the resultant interface is more like a canvas than a document. How many people feel comfortable writing on a canvas?

The interface scaffolding strategy employed in MediaText is one that has been employed quite successfully before in computer-based spreadsheets. The first computer-based spreadsheets were one-for-one with their paper cousins, except for one function: the user could change one number and the computer would automatically recalculate the spreadsheet. Current-day computer-based spreadsheets provide all sorts of functions literally impossible to mimic in paper: voice and video annotations, automatic graphing, presentation tools. Why did the initial spreadsheets catch on so dramatically? The "blue line" (Figure 5) learning curve, is our bet.

After using MediaText for a while, users report that they want to create more visually exciting documents; they want to embed a picture or movie inside the text itself, for example. In effect, they want more control over page layout and they want that margin to go away! Alas, like the GPC Editor, MediaText too is not

able to fade its interface scaffolding.

By the way, the document depicted in Figure 6 is not just another report to be filed in the circular file cabinet. Rather, it is a report on the status of a tree planting project written by a student at Community High School. The student, who has an interest in trees, was taking an Urban Ecology course and approached the Michigan Department of Natural Resources to see if there was a project they needed help with. Well, it turns out that the Michigan Department of Natural Resources had funds to plant trees in Ann Arbor, but no funds to follow up and see how those trees were doing. The student took her camera out to the streets suggested by the DNR, took pictures, and moved them from a Kodak PhotoCD disk into her MediaText document. Without MediaText it would have been much harder to create an appropriate, professional-level report. Given the importance of the task, and the low overhead for using the technology, it's not hard to see why the student's motivation and perseverance remained high.

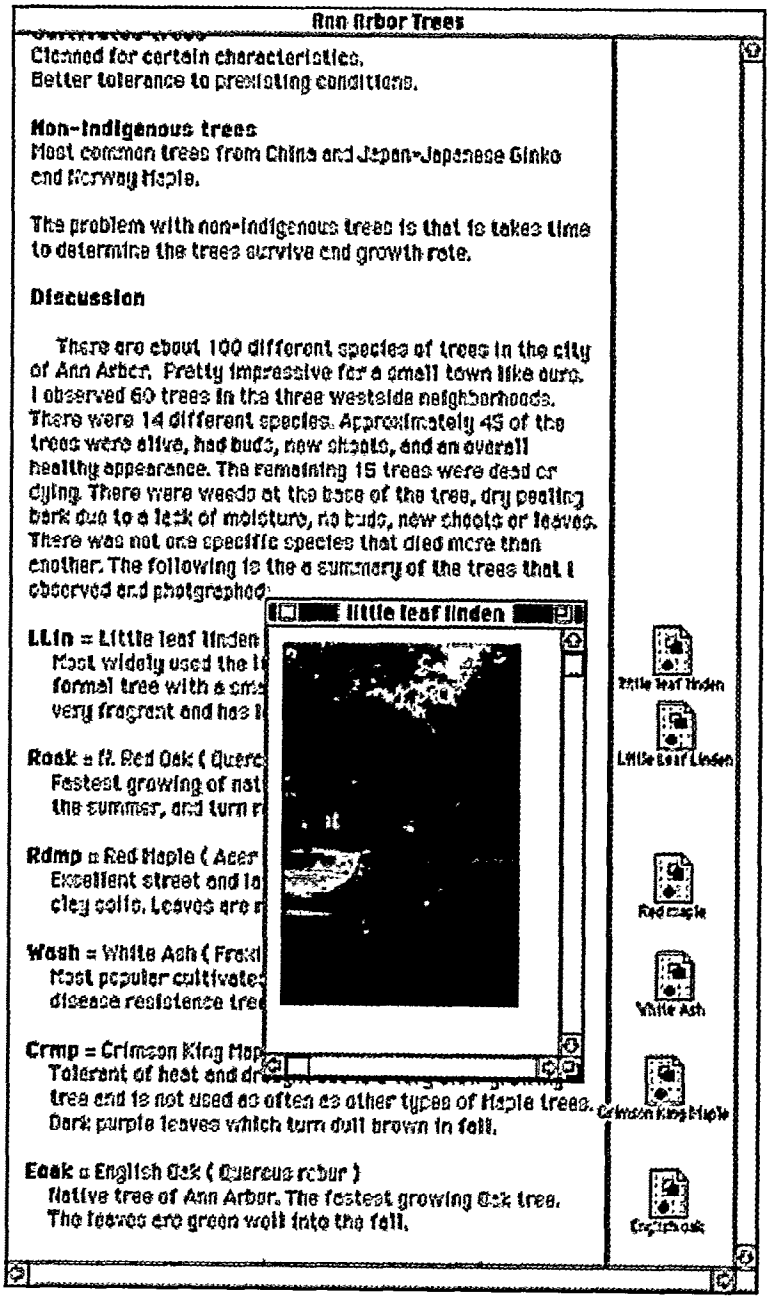


Figure 6 *Media Text "Ann Arbor Trees" document produced by a high school student*

Adaptable Tools To Support Growth of Expertise

Emile is a scaffolded environment in which high school students learn science by building physics simulations– worlds where objects fall, bounce, and launch in parabolic trajectories. Programming in Emile plays on the same themes that made MediaText easy and learnable. Students construct simulations in Emile by assembling components in a plug-and-play fashion, something that students already know how to do from playing with Lego. Over one hundred components are built into Emile's library to start (though students can always add more), ranging from graphical objects that can be dragged and dropped in a simulation of one-dimensional projectile motion, down to code segments that play digitized video or sound.We finally learned our lesson from the GPC Editor and MediaText: Emile enabled the student to adapt the scaffolding. Figure 7 depicts the Preference Page from the student's Design Notebook (where students gather and connect their components in Emile). Note the range of scaffolding strategies employed in Emile over which students had direct control.

• Note the check box for "Process Control." With this preference selected, which it was initially, Emile channeled the student in how he or she went about constructing the simulation. However, as students grew more confident in

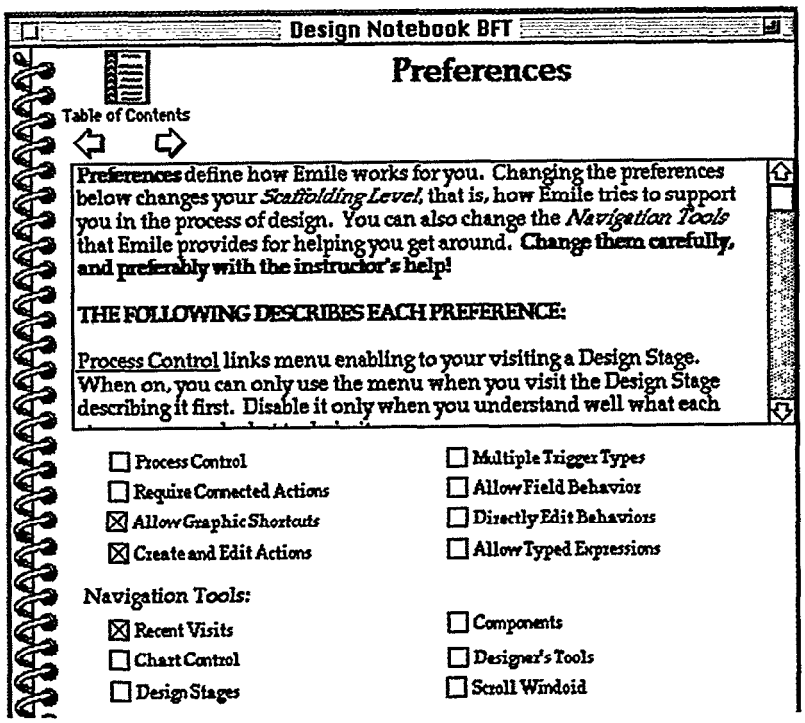


Figure 7
Student-selectable preferences in the Emile scaffolded environment.

their abilities, they went to the Preference Page and turned off this scaffolding. When they were ready, they were on their own to figure out what to do first and then next.

• Note the check box for "Create and Edit Actions." Actions in Emile are the code segments that students used from the library. When students reached the point where they wanted a code segment that was not in the library, they could go back to the Preference Page and check this on. Now they were free to create their own components and add them to the library.

• Note the check box for "Directly Edit Behaviors." A behavior in Emile determined how a graphical object functioned in a simulation. When students knew enough about programming in Emile that they wanted to go beyond components, they turned this switch on. Then, Emile was no longer plug-and-play. It was type-and-go.

Interestingly, some students turned the scaffolding off and on several times during a project. Piaget called this behavior "horizontal decollage." The idea is that students do not progress across the board in all skills at the same rate. Some days they move forward in some areas, some days they stand still in others. A student might feel completely comfortable creating actions and working without process control when he or she is setting things up, say, to show instructions for a simulation, but the student may want all the scaffolding back on when it comes time to actually build the simulation. That's okay, too. Scaffolding is not an all-or-nothing deal. Good scaffolding is there when the student wants it, and is not there when the student wants to work independently.

The bottom line with Emile is that students built simulations that they found exciting and joyful and they learned a lot about physics in the process (Guzdial, 1994). Students went from talking about how heavy objects fall faster, to the role of acceleration due to gravity. They stopped saying that rocks just fall from buildings, and started talking about instantaneous and average velocity. That's the point: Construction in pursuit of learning, and in Emile, at the student's own pace.

Challenges to Realizing Scaffolding in Software

The notion of scaffolding comes out of the education literature where there is an assumption that humans provide the scaffolding. To realize scaffolding in software several issues need to be explored.

• First, the challenge for software designers is in instantiating scaffolding techniques, e.g., coaching, in software. The challenge is very serious: a good human coach can have a substantial battery of strategies for scaffolding, while even implementing one or two in software is quite non-trivial. The human coach can pick and choose which particular strategies and tactics to use, whereas software-realized scaffolding may not be as flexible. How can we ensure the effectiveness of such software-realized scaffolding?

• Second while the education literature argues that scaffolding must, by definition, fade, there appears to be times that fading software-realized scaffolding might not be appropriate. For example, supporting a learner in adapting tools, or providing a help system for the tools, are functions that should always be available. What are the differences, then,

between software-realized scaffolding and human-realized scaffolding?

• Third, our use of the notion of scaffolding (Collins, Brown, and Newman, 1989) is broader than that found in the education literature. For example, coaching would be a legitimate scaffolding strategy, but "adaptable tools" would not. Perhaps we have over-burdened the term; or perhaps the education world has too limited a notion of that term.

• Fourth, what **new** scaffolding types might arise precisely because we can realize them in software, in a computational medium? By and large the scaffolding strategies that have been instantiated in software to date mimic human strategies. This is a reasonable place to start; new technology mimics old–for a time. However, the computational medium surely will afford new opportunities; what, then, are the scaffolding strategies and tactics that exploit a computational medium?

Build learner-centered software! Build scaffolding! Easy to say, hard to do.

What To Do Next?

In a rare moment of humility, we recognize that the TILT Model is just the beginning; if the field is truly going to support learners in a serious way, a great deal of invention and hard work needs to be done. Will AI solve "the problem?" The AI community has long promoted the notion of "smart machines." Intelligent interfaces will know about you and use that information to better serve you. Making computers smarter is not incompatible with making humans smarter; quite the contrary, in fact: the two ideas go hand-in-hand. Now, let's get on with it!

Understanding task domains and practices is a high priority: what is it that people need to know and do–and how does that change as a function of the new technology? What new interaction paradigms will arise as a function of that new technology? What is the nature of the conversation between humans and machines when the latter have speech and vision? And, what is learning? What is the impact of individual differences and different rates of development? The laundry list of unknowns is staggering–as it always is.

The fundamental issue remains, however: Is the distinction between the notion of users and learners merely rhetoric or is there truly a substantive distinction being made? If words count for something, then there is a genuine distinction being made: if addressing the needs of users is the driver, then it is natural to focus on ease of use; if addressing the needs of learners is the driver, then it is natural to focus on the development of understanding, performance, and expertise. The former is unquestionably important; however, the latter is unquestionably critical.

The engines of the 21st century inspire awe; what does it mean to have 1000 MIPS inside a 2 ounce, 8.5 x 11 inch notebook? What does it mean to be able to see and talk to anyone, anywhere, anytime? The human-computer interaction community is being handed an unprecedented opportunity: we are the creators of the gates through which millions and millions of people will gain access to those mighty engines. What is our responsibility? Taming those engines for human use is surely a value-added; but the opportunity is grander than that: if we support learning and learners, then we support that which is quintessentially human: *individuals developing into better individuals.* Such a challenge and responsibility *truly* inspires awe.

Acknowledgements

The authors wish to thank the following for their comments on earlier drafts of this article: Kate Bielaczyc, Kathy Brade, Shari Jackson, Jeff Spitulnik, Steve Stratford.

The article owes deeply to the insights of our colleagues Phyllis Blumenfeld, Joe Krajcik, and Ron Marx. And, we gratefully acknowledge the productive and joyful collaboration between us and the teachers and students of Community High School in Ann Arbor, Michigan.

References

[1] Buxton, W. (1991) Presentation at Opening Panel, ACM CHI '91 Conference, Monterey, CA.

[2] Card, S., Moran., T., Newell, A. (1983) *The Psychology of Human Computer Interaction,* L. Erlbaum & Associates, Hillsdale, NJ.

[3] Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* Hillsdale, NJ: Lawrence Erlbaum and Associates.

[4] Dewey, J. (1964) *John Dewey on Education, Selected Writings,* R.D. Archambault (Ed.), U. of Chicago Press, Chicago, IL.

[5] Gardner, Howard (1993) *Multiple Intelligences: Theory in Practice,* Basic Books, New York, NY.

[6] Guzdial, M., Soloway, E., Blumenfeld, P., Hohmann, L., Ewing, K., Tabak, I., Brade, K., & Kafai, Y. (1992a). The future of CAD: Technological support for kids building artifacts. In D.

The preparation of this article was supported, in part, by grants from the National Science Foundation, nsf-red-9359481 and nsf-tpe-9153759.

Balestri, S. Ehrmann, & D. L. Ferguson (Eds.), *Learning to design, designing to learn: Using technology to transform the curriculum* Norwood, NJ: Ablex Publishing Company.

[7] Guzdial, M., Weingrad, P., Boyle, R., & Soloway, E. (1992b). Design support environment for endusers. In B. A. Myers (Ed.), *Languages for developing user interfaces* (pp. 57-78). Boston, MA: Jones and Bartlett.

[8] Guzdial, M. J. (1993). *Emile: Software-realized scaffolding for science learners programming in mixed media.* Unpublished Ph.D. dissertation, University of Michigan.

[9] Hay, K.E., Jackson, S., Boyle, R., Guzdial, M. and Soloway, E. (1994) Student Creation of Multimedia Documents. *Journal of Computers and Education,* In press.

[10] Jackson, S. L., Hu, J. T., Soloway, E., (1994) The ScienceWorks Modeler: Scaffolding the Doing of Science, ACM CHI '94 Conference Proceedings Companion, April 1994, Boston, MA, in press.

[11] Norman, D., Draper, S. (1986) *User Centered System Design,* L. Erlbaum & Associates, Hillsdale, NJ.

[12] Norman, D., (1993) *Things That Make Us Smart,* Addison-Wesley, Reading, MA.

[13] Papert, S. (1993) *The Children's Machine: Rethinking School in the Age of the Computer,* Basic Books, New York, NY.

[14] Piaget, J. (1954) *The Construction of Reality in the Child,* Basic Books, New York, NY.

[15] Rogoff, B. (1990). *Apprenticeship in thinking: Cognitive development in social context.* New York: Oxford University Press.

[16] Senge, P. (199) *The Fifth Discipline: The Art and Practice of The Learning Organization,* Doubleday, New York, NY.

[17] Soloway, E., Guzdial, M., Brade, K., Hohmann, L., Tabak, I., Weingrad, P., & Blumenfeld, P. (1993). Technological support for the learning and doing of design. In M. Jones & P. H. Winne (Eds.), *Foundations and frontiers of adaptive learning environments* New York: Springer-Verlag.

[18] Vygotsky, L.S. 1962. *Thought and language.* Cambridge, MA: MIT Press.

[19] Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Morgan Kaufman.

[20] D. Wood, J.S. Bruner, G. Ross. 1975. The role of tutoring in problem-solving. *Journal of child psychology and psychiatry.* V. 17. 89-100.