Adapting User Interface Design Methods to the Design of Educational Activities

Clayton Lewis, Cathy Brand, Gina Cherry, and Cyndi Rader

Department of Computer Science
and Institute of Cognitive Science
University of Colorado, Boulder CO 80309
{clayton, brand, gina, crader}@cs.colorado.edu

ABSTRACT

We have adapted the programming walkthrough technique to help design computer-supported educational activities in elementary school science. We present examples from a case study which illustrate ways in which design of an educational activity is similar to and different from design of a user interface. We have found that the walkthrough approach is useful in this new setting, and that it sheds new light on the general task-centered orientation to design.

KEYWORDS: analysis methods, children, design techniques, educational applications, end user programming, task analysis.

INTRODUCTION

The central challenge in traditional user interface design is to create a computer system that supports some collection of potential user tasks in such a way that user productivity is enhanced. Productivity can be enhanced by permitting faster correct performance, by reducing the incidence of errors, or by some combination. The task-centered design approach [9], addresses this challenge by using specific example tasks to evaluate and compare design alternatives. In the early stages of task-centered design, walkthrough methods are used to generate and critique *scenarios*, which are generated by spelling out how a given example task would be performed given a particular system design.

In our work on educational software we have encountered a somewhat different design challenge. Our problem is to design not just a computer system, but also a collection of activities with that system, such that learners engaging in those activities will achieve given educational goals. Thus, the design challenge for computer-supported educational activities includes the design of tasks as well as of the system, and the measure of success is not productivity, considered simply, but a more complex evaluation of the effects of performing the tasks. For example, an educational activity that learners complete quickly and accurately is of no value if they learn nothing from it.

Despite these differences in design problems, we hypothesized that the core logic of task-centered user interface design, and of the associated walkthrough methods, could be adapted to the design of educational activities. We present the results of our exploration of this possibility by describing a case study in which we adapted the programming walkthrough technique [2, 9] to the design of a suite of educational activities about plants for an elementary school science unit.

THE CASE STUDY

The sTc project

Science Theater/Teatro de Ciencias (sTc) is a research project exploring the educational value of model creation by elementary school science students [14]. Children participating in sTc use software to create animated, graphical models of processes and mechanisms they are studying. Like other researchers in educational technology, we quickly found that simply making software available and encouraging its use did little to support learning. We have found that our design and implementation efforts have been more and more directed to the design of activities supported by the software, rather than to the design of the software itself.

The pedagogical objectives of these sTc activities fall into four major categories:

Science content. A primary goal is to deepen students' understanding of their science topics by using models to focus on "how" and "why" questions. This approach contrasts with most existing elementary science curricula, which tend to focus more on observation and the use of experiments to show, but not explain, interesting phenomena. We are also interested in addressing common science misconceptions.

Science as inquiry. In accordance with current science education reform, modeling activities are used within a

framework of scientific inquiry in which students are encouraged to ask questions, speculate about possible answers, and test and revise their ideas [22, 23]. This inquiry cycle promotes a view of science as a process, rather than as a body of facts, and encourages students to bring their own experiences and conjectures into the science classroom.

Model-based reasoning and epistemology of models. A number of research efforts have focused on the potential benefits of modeling [3, 7, 8, 13, 15]. Grosslight et al. [7] have identified three levels of understanding about models. At level 1, models are viewed as simple copies of reality. At level 2, students understand that models often highlight and/or simplify some aspects of the phenomenon being modeled, rather than matching reality exactly. At level 3, students display a more sophisticated understanding of models, which includes three factors: the use of models to develop and test ideas, rather than to copy reality; the consideration of alternative models to explain the same phenomenon; and the manipulation and testing of models in order to inform and revise ideas. In sTc, our goal is to help students gain a level 3 understanding of models, with particular emphasis on the use of models to demonstrate and test ideas, and on the importance of revising models when initial ideas are found to be incorrect, or to incorporate new information.

Software mastery. A well-conceived activity will pave the way for subsequent activities by enhancing the child's knowledge of and skill in using the modeling software. By the time the students complete an introductory series of activities which includes the two in this case study, they should understand and program well enough to create their own models. In our case study, the modeling software is Visual AgentTalk [16, 17, 18], a rule-based visual programming system.

The Software

Models in Visual AgentTalk consist of a collection of software agents whose behavior is specified by a set of rules, and a worksheet in which the agents are arranged. The visual appearance of an agent is a small picture called a depiction. An agent's rules can cause it to move around on the worksheet, to change its depiction (so that it looks different), and to interact with other agents. For example, one agent can cause a nearby agent to disappear, a behavior which might represent something being eaten, or a reagent being consumed in a chemical reaction.

Models can be built from scratch, with all rules and depictions for the agents created by the user, or they can be built using agents already provided. This flexibility makes it possible to devise modeling activities that differ greatly in the sophistication required of the learner, from models in which all behavior must be specified by the learner, to ones in which the learner has only to arrange a collection of agents whose behavior is already provided.

Sample Activities

In our case study, we used a walkthrough technique to refine the design of two educational activities from a unit on plant science for fourth and fifth graders. In the *Extend-Sugar-Production* activity, the children would be given a simple model in which water was transported from the roots to the leaves. In the leaves, the water would trigger the creation of sugars, which would then be transported back to the roots. In this simple model, carbon dioxide and light, both of which are necessary for sugar production, would be present at the surface of the leaf, but the carbon dioxide would not be consumed when sugar is produced. Also, only sugar, and not oxygen, would be produced in the simple model. The learner's task would be to elaborate this model so that carbon dioxide would be consumed and oxygen produced when sugar is created.

This activity has three pedagogical objectives. The science content objective is for learners to see how plants create their own food, a difficult concept for elementary-age science students [19]. In Extend-Sugar-Production, students deal with this concept explicitly by viewing and modifying the rule in which the leaf creates food. The model epistemology objective is to show how simple models can be extended to show phenomena at deeper levels of detail. The software mastery objective is to develop skill in modifying rules, as a step towards being able to create one's own rules.

The initial conception of the *Build-A-Flower* activity was that children would build two model flowers using a set of flower parts provided for them. If the parts of their flowers were properly arranged, a model bee (also provided) would carry pollen from one flower to the other, and seeds would be produced.

Build-a-Flower has two pedagogical objectives. The science content objective is for students to solidify their understanding of flower anatomy, following up an activity in which they dissect real flowers and make labeled drawings of their parts. At the same time, Build-a-Flower should help learners link flower anatomy to flower function by demonstrating the phases of pollination and seed production. The software mastery objective of this activity is to help learners understand the relationship between the behavior of an agent in a model and the rules that specify its behavior.

The Walkthrough Method

The core logic of walkthrough methods is that a scenario is developed to represent what will happen when some complex system is deployed, and this scenario is then critiqued. In code walkthroughs in software development, the scenario is an imagined execution trace of a program, and the trace is examined to detect such faults as uninitialized data. In a cognitive walkthrough in user interface design (10, 20, 21), the scenario represents the actions a user must perform to carry out an example task using a particular interface design. The scenario is critiqued

for poorly-motivated actions, actions inadequately cued by the interface, and the like.

The programming walkthrough [1, 2, 9] is a somewhat more complex technique in which a scenario consists of a series of steps, including hypothetical mental steps, through which the programmer would progress to accomplish some goal using a given programming environment. Key points in the critique include identifying choices not supported by adequate knowledge, and choices which require difficult problem solving.

The programming walkthrough includes the specification of *guiding knowledge* - knowledge which is needed by the user to make programming decisions, and which might be provided as part of the documentation or training for the programming environment. The inclusion of guiding knowledge makes it possible to apply this technique to situations in which specific background knowledge not cued by the system under evaluation is needed, while the cognitive walkthrough is more appropriate for highly-cued situations, such as arise in typical end-user applications.

We used the programming walkthrough as the starting point for our work because our tasks require programming-like operations that are not directly cued. For example, modifying a rule is done by dragging new elements from a palette into a rule, not by (say) choosing "modify rule" from a menu and interacting with a dialog box.

We held a series of walkthrough sessions for each of the two educational activities in the case study. We began by sketching our original conception of the activity, including the instructions to the learner and the initial software configuration. We then outlined the main steps learners would need to take to carry out the activity. We used “progressive deepening” in each series of sessions; that is, in each series of sessions we started with a high-level list of steps, which we progressively broke down into more and more detailed substeps in later sessions.

For example, an early step list for Extend-Sugar-Production was as follows:

1. Find the leaf rule [the rule that makes sugar is attached to the leaf agent]
2. Understand the leaf rule [that is, understand how the rule specifies what is produced and what is consumed]
3. Find the action(s) to be added to the rule [to consume carbon dioxide and/or produce oxygen]
4. Drag the action(s) into the rule box [this is how components are added to rules]
5. Edit the action as needed for this particular situation [a generic “consume” or “produce” actionhas to be adjusted to consume or produce a specific thing]

In a later session we expanded some of these steps; for example, we identified substeps of “Understand the leaf rule” and “Edit the action”. We also debated whether “Understand the leaf rule” was necessary to accomplish the task.

All but one of the walkthrough sessions were done at a whiteboard without looking at the actual computer implementation. In the final session for the Extend-Sugar-Production activity, we did examine the actual computer implementation, in part to see whether this would suggest issues that we missed in our less detailed analyses earlier.

In each session, we critiqued the step lists, looking for steps that required knowledge learners would lack, or that would otherwise be difficult to carry out. We also examined each step list as a whole to judge whether it was likely to support the pedagogical objectives for the activity. When difficulties were found, we considered how to modify the task or the supporting system to deal with those difficulties, as illustrated below.

These discussions were time consuming. We spent about four hours analyzing Make-Sugar-Better and about eight hours analyzing Build-A-Flower.

We found that the basic mechanics of the programming walkthrough could be applied without change to educational activity design. For example, the development of the scenarios was no different. But significant differences did emerge in the specifics of the process. We describe and illustrate these differences in the following section, and at the same time illustrate the value of the process for our design problem.

RESULTS OF ADAPTING THE PROGRAMMING WALKTHROUGH TO EDUCATIONAL ACTIVITY DESIGN

Shape the task, not the system. When evaluating a programming environment or an end user application, the user tasks are givens. One cannot respond to a potential problem in an interface by changing what the user is trying to accomplish; rather, the system has to be adapted to support whatever the users' goals are thought to be.

In educational activity design, by contrast, one has the freedom to redefine the learner's task if that enhances the educational value of the activity. For example, for reasons described below, we changed the Build-a-Flower task from its original form, in which the learner's objective was simply to make pollination happen, to one in which learners also had to explain why one pollinator pollinated their flowers and another did not. Such changes are sensible in educational activity design, and not in interface or environment design, because the tasks are part of the design space and not simply benchmarks.

The fact that tasks change during the design processes does alter the character of the walkthrough process somewhat. Each time a change to a task is contemplated, at least some part of the walkthrough must be redone. This is not different in principle from what happens in a programming walkthrough or a cognitive walkthrough when a change in the system design is contemplated.

The progressive deepening approach we used for the walkthroughs seemed to work well. Many of the modifications we made to the activities were made on the basis of only a high-level look at the steps of the activities, so that we did not find ourselves revisiting the details of the activity over and over again as the activity changed.

Incorporate guiding knowledge into the task specification, not just into background information. As mentioned earlier, guiding knowledge in a programming walkthrough is normally general orienting information that is made available to users in the form of training or documentation. For example, the canons of object-oriented design form part of the guiding knowledge needed to effectively use an object-oriented programming language like C++ or Java. In our walkthroughs, we exposed some of this kind of general guiding knowledge - for example, the knowledge needed to discriminate the roles of conditions and actions in rules.

However, we also frequently encountered the need for guiding knowledge that was specific to a particular task, and not general to the modeling environment. For example, in Build-a-Flower, our walkthrough suggested that learners would not be sure in what order to assemble their flower parts, and that an unfortunate choice of order of assembly would make the task quite difficult. Our response was to identify guiding knowledge that we could incorporate into the description of the Build-a-Flower task that learners would be given: "It's a good idea to build your flowers by building up from the stem."

In common interface or environment design the analogous move is hardly possible. At best one can provide special support for common *classes* of tasks, as is done using "wizard" facilities. Thus one can provide advice about how to create charts, but not how to create some specific chart, since one does not know what specific chart the user wants to create. In designing an educational activity one often *does* know what the specific task will be, and one has the freedom to attach very specific guiding knowledge to it.

Design in failure. In user interface design, the aim is to maximize the likelihood that users will find an efficient way to perform the task. In pedagogical task design, it may be necessary to *minimize* this likelihood, so that learners will be forced to learn how to solve some kind of problem, which they would not encounter if the task were too easy.

One of the pedagogical goals of Build-a-Flower was for learners to examine the rules of the model, and to relate the content of the rules to the behavior of the model. One possible occasion for this experience was the need to determine what color flower petals would attract a pollinator. To create this occasion, it would be necessary to design the overall task in such a way that learners would be highly likely to choose the wrong petal color for their flowers: if they chose the correct color for their task, the pollinator would behave as desired, and there would be no need to examine the rules. The walkthrough suggested that our initial version of the task would yield a failure likelihood of only about 50%, too low to be effective.

As a remedy, we first considered modifying the task by providing a wide variety of petals that would not attract the pollinator, and only one that would. Further analysis showed that this approach had the serious difficulty that learners would have to change their incorrect petals to complete the task, an operation likely to be quite tedious and without any pedagogical value. This observation led to a further modification of the task, in which learners would be given two different pollinators and only one color petal. Only one of the pollinators would visit the flowers, since the only petals available would attract only one of the pollinators. The learner would be asked to observe that only one of the pollinators would visit their flowers, and to explain why this was. Looking at the pollinators' rules would help learners figure out the correct explanation.
For this version of the task, the likelihood of failure has been raised to 100%, in the sense that one of the pollinators will necessarily fail to function. On the other hand, there is no need for the learner to modify their flowers in order to observe successful pollination, since one of the two pollinators will function properly.

The fact that designing in failure sometimes makes sense in educational activity design, and not in ordinary systems design, is just one outcropping of a more general difference that we pointed to earlier. The critique applied to the

scenario in activity design is different from that used in the programming or cognitive walkthroughs, in that considerations other than productivity must be included. The whole range of pedagogical objectives must be reflected in the critique, resulting in a considerably more complex evaluation.

Design for creative scope. In user interface design, while one may aim to support a wide variety of tasks, one usually tries to limit the options available to users in performing any one task. The idea is to reduce the need for costly and error-prone decision processes [11, 12]. In pedagogical task design one may wish to expand the options, at the cost of more decisions, as a way of engaging the creative efforts of the learner.

In our analysis of the flower-building task, it became clear that it is possible to build flowers with a small fixed number of parts; however, if this were enforced by the system, learners could create only a narrow range of flowers. By including parts that could be used multiple times, the range of possible flowers can be expanded. For example, if learners are allowed to insert one or more filament parts to support the anthers, learners can create flowers whose anthers are at different heights. Similarly, if petals can be built up from varying numbers of petal parts, many different petal sizes, shapes, and arrangements are possible.

Note that these variations do not increase the range of tasks learners can perform: they are still limited to building flowers that will or will not be pollinated successfully by the model pollinators provided, and the allowed variations have no influence on this. The only effect of the variations is to change the appearance of the flowers, while leaving their function unchanged. The variations permit the learners to approach the same task in different ways, corresponding to their differing conceptions of what a flower should look like. If we did not provide these variations, it is likely that the children would be less interested in the activity, and it would be more difficult for the children to connect their ideas of flowers with the models they create. These pedagogical benefits outweigh the cost of the decisions learners must make in creating flowers with a less constrained collection of flower parts.

This departure from normal interface design practice is just another reflection of the richer set of evaluation criteria appropriate to designing an educational activity. As with designing in failure, it arises from the replacement of simple productivity criteria by pedagogical criteria in the critiquing phase of the walkthrough.

Draw support from other activities. We noted earlier that one of the problems identified in the walkthrough for Build-a-Flower was the likelihood of problems in assembling a flower with its parts in the correct relative positions. We dealt with some of this problem by adding the suggestion to build the flower from the stem up. However, we still anticipated that students could place parts in the wrong arrangement - for example, placing an anther (a male part) directly on top of the ovary (a female part), blocking the pollen's access to the ovary.

We considered various ways in which the behavior of an incorrectly constructed flower might call attention to the problem. Our initial idea was that an incorrectly constructed flower would not get pollinated, and that this would indicate that the learner should revise the flower. This idea was discarded because it did not give the learner enough information about where the problem was. We then considered having the flower parts fall to the ground if incorrectly placed. This approach would let the learner know that something was wrong with a particular part, but would not help the learner figure out how to correct the problem.

To help the student correct the problem, we decided to link the Build-a-Flower activity explicitly to an earlier class activity: flower dissection. In order to support the computer activity, we decided to recommend to the teacher that the children create labeled drawings of flower parts as part of this activity. We then revised the instructions for Build-a-Flower to ask the children to refer to these drawings. These drawings should help resolve some of the possible positional uncertainties within the Build-a-Flower activity. There is, of course, the chance that some children will not be able to relate the parts in their flower drawings with the corresponding parts as represented in the software for Build-a-Flower but we think working on these connections has pedagogical value in itself.

Here again, we have a kind of design initiative that is not often possible in user interface design. We can do it here because we can rely on a set sequence of learner activities which we created with the teacher. Not only did we draw support from related hands-on activities but to some degree we changed the nature of these activities.

Motivate software mastery by links to content domain. One of the pedagogical objectives of Extend-Sugar-Production is to help learners understand how to modify rules. In our initial thinking about the activity, we had not identified consuming carbon dioxide and producing oxygen as the specific modifications for learners to make; rather, we had a general idea that the initial, unelaborated sugar model would be a good base for learners to work from. The specific activity involving carbon dioxide and water arose from the walkthrough analysis as we tried to envision what activity goals would induce learners to examine and modify one or more rules in the base model.

Our choice of activity was influenced by our desire not to give learners explicit instructions to examine and modify rules. Rather, we wanted to present an activity that was well-defined and well-motivated by its science content, in this case creating a more accurate model, and that would require learners to deal with the software features we wanted them to learn about. This preference derives from our previous experience with more traditional ways of

introducing software concepts which did not interest many of the children and consequently were not effective in teaching them enough programming skills to create conceptually rich models.

Analogous considerations could be brought into ordinary interface design, but seldom are, except implicitly. Implicitly, it is content concerns that presumably drive most learning of applications, at least by discretionary users. But might not interfaces be explicitly designed to "show off" the content possibilities that their features support? The idea of a catalog of application examples as part of an interface [6] is one approach to this issue.

Design to avoid generating misconceptions. Some of the design choices which arose from our walkthroughs were influenced by the need to avoid presenting misleading models to the learners. Unfortunately, using simple modeling tools and methods that are accessible to children, many important physical processes cannot be represented faithfully.

The treatment of light in Extend-Sugar-Production illustrates this point. We considered asking the children to elaborate the base sugar model so as to show photons being consumed when sugar is produced, rather than just showing that light must be present. Unfortunately, making photons move on a two dimensional surface that is also populated by (for example) carbon dioxide molecules, is complex. Simple approaches result in photons being stopped by air molecules. Other approaches would result in complex rules which would be difficult for learners to understand or revise. The walkthrough allowed us to anticipate this problem and redesign the activity to avoid it.

Familiar walkthrough results

In this presentation, we have stressed ways in which the use of walkthroughs in activity design differs from their use in environment or interface design. But in fact, some of our results are just as would be expected from an ordinary interface critique. For example, when we examined the details of Extend-Sugar-Production we found that identifying the portion of a rule that consumes something is potentially quite tricky. The relevant part of the rule does not actually describe the thing being consumed but refers to it by its location relative to the agent to which the rule is attached. To find out what is going to be consumed one has to look at the condition of the rule to see what kind of thing will be in the position where the consumption will take place.

There is a further difficulty here. In the condition of a rule, the agents that must be present are represented only by their depictions - small diagrammatic pictures. This means that the learner must be able to identify these depictions, without the aid of (for example) names. This consideration led to one suggestion for the design of the Visual AgentTalk software itself which came out of our walkthroughs: that conditions be redesigned to contain both an agent's name and its depiction.

Incidentally, these detailed criticisms only emerged when we examined the actual software. We had not anticipated them in our whiteboard discussions.

DISCUSSION

We found the walkthrough process to be of great value in developing the activities in the case study. The examples we have presented are illustrative of a large number of changes we were led to make in the activities, some of them quite profound. These results have been very important for a project that needs to have educational activities developed on a tight schedule that is coordinated with children's ongoing science curriculum.

Given our schedule, it may not be practical for us to analyze every activity we plan in such detail. We did find that we learned enough from the case study that we can make some changes to other related activities without performing walkthroughs on them at the same level of detail as we did in the case study.

One of the factors that made our walkthrough process time consuming was that we were simultaneously redesigning the software and the task; thus, a large number of options were available to us at every decision point. This would not be the case if the walkthrough technique were applied to educational activities using commercially available software.

The balance of costs and benefits for the method is hard to assess very accurately. It is easy to argue that the method is cheap at the rate of a few hours per activity designed, if one imagines large numbers of children and teachers working with the resulting activities. We suspect that we, and many other designers, are guilty of under-budgeting for analysis and evaluation in our development work, and that that contributes to a feeling that the method is expensive.

Besides the time required, another reservation we have about the method is that it would not work as well for less well defined activities. The sTc project includes more open opportunities for children to create models of processes or phenomena of their own choosing, and it is clear that it would be much harder to anticipate the difficulties they might face in doing this, and to come up with ways to get around these difficulties, than it was for the better-defined activities we worked on in the case study.

Stepping back, we also found our exploration to be of considerable methodological interest. The central notion of task-centered design is revealed to be more subtle, and more flexible, than we thought. The simple notion of a task as being defined simply by a user's goals and a context, which suffices for common interface work, can be extended to include not only the user's goals but also those of other parties to the design - in our case, our various pedagogical goals.

An earlier effort to pushing task-centered design into a new domain (an unpublished student project [4] for the first author) anticipated some of our findings in another domain. They worked with an artist on the design of a robotic drawing facility, and found that there as the simple notion of task needs to be broadened. Their artist collaborator found it very uncomfortable to imagine an artistic "task" separate from the tools that support it, so in that study as in this one the tasks used in the design process were in flux well into the process.

Returning to standard interface design, it appears that some of the considerations that we have had to deal with in our pedagogical setting might usefully be imported. We can look beyond simple speed and accuracy goals to create interfaces that promote learning and creativity.

An example of the possibilities could be developed from DiGiano's work on self-disclosing systems [5]. These systems display information about how to make more sophisticated use of their facilities while the user is at work; for example, when a user performs an operation by direct manipulation the system shows how commands in a script could accomplish the same result. One can argue that a rational designer would accept some short-run usability penalty for such a facility that would be repaid in the long run by more effective advanced usage. Thus, as in our work, the design evaluation should consider not only usability characteristics of the user's tasks but also their pedagogical implications.

ACKNOWLEDGMENTS

This work was supported by a grant from the Applications of Advanced Technologies program of the National Science Foundation. We thank the other members of the sTc group, Erika Arias, Cory Buxton, Heidi Carlone, Carlos Garcia, Teresa Garcia, Linda Hagen, Page Pulver, Steve Guberman and Mary Lou Salazar, the students at University Hill Elementary School in Boulder, and Alex Repenning, the creator of Visual AgentTalk, for their assistance and contributions.

REFERENCES

1. Bell, B., Citrin, W., Lewis, C., Rieman, J., Weaver, R., Wilde, N. and Zorn, B. Using the programming walkthrough to aid in programming language design. *Software Practice and Experience 24*, 1 (1994), pp. 1-25.
2. Bell, B, Rieman, J, and Lewis, C. Usability testing of a graphical programming system: Things we missed in a programming walkthrough. In Proc. *CHI'91 Conference on Human Factors in Computing Systems* (New Orleans, April 28-May 2, 1991), ACM New York, pp. 7-12.
3. Bliss, J. From Mental Models to Modeling, in H. Mellar, J. Bliss, R. Boohan, J. Ogborn and C. Tompsett, Eds., *Learning with Artificial Worlds: Computer Based Modeling in the Curriculum*, The Falmer Press, Washington, DC, 1994.
4. Clark, C., Justus, S. and Santiago, C. Mapping the LEGO Brick to Fine Arts Applications. Course project report, Department of Computer Science, University of Colorado, 1995.
5. DiGiano, Chris, and Michael Eisenberg. Supporting the end-user programmer as a lifelong learner . Department of Computer Science Technical Report CU-CS-761-95, University of Colorado at Boulder, 1991.
6. Fischer, G). Domain-Oriented Design Environments, in L. Johnson and A. Finkelstein, Eds., *Automated Software Engineering*, Kluwer, 1994.
7. Grosslight, L., Unger, C., Jay, E., and Smith, C. Understanding Models and their Use in Science: Conceptions of Middle and High School Students and Experts. *Journal of Research in Science Teaching 28* , 9 (1991), 799-822.
8. Jackson, S. The ScienceWare Modeler: A Learner-Centered Tool for Students Building Models. *Human Factors in Computing Systems: CHI'95 Conference Proceedings*, ACM, New York, 1995, pp. 7-8.
9. Lewis, C, Rieman, J, and Bell, B. Problem-centered design for expressiveness and facility in a graphical programming system. *Human-Computer Interaction 6*, 1991, pp. 319-355.
10. Lewis, C. and Wharton, C. (in press) Cognitive walkthroughs. In Helander, M., Landauer, T., and Prabhu, P. (Eds.) *Handbook of Human-Computer Interaction, 2d Edition*. Amsterdam:Elsevier Science.
11. Olson, J.R., and Nilsen, E. Analysis of the cognition involved in spreadsheet software interaction. *Human-Computer Interaction 3*, 1988, 309-350.
12. Olson, J.R. and Olson, G.M. The Growth of Cognitive Modeling in Human-Computer Interaction Since GOMS. *Human-Computer Interaction 5*, 1990, Lawrence Erlbaum Associates, Inc.
13. Penner, D. E., Giles, N. D., Lehrer, R. and Schauble, L. (1997). Building Functional Models: Designing an Elbow. *Journal of Research in Science Teaching 34*, 2, 125-143.
14. Rader, C., Brand, C. and Lewis, C. Degrees of Comprehension: Children's Mental Models of a Visual Programming Environment, *Human Factors in Computing Systems: CHI'97 Conference Proceedings*, ACM, New York, 1997.
15. Raghavan, K. and Glaser, R. Model-Based Analysis and Reasoning in Science: The MARS Curriculum, *Science Education 79*, 1 (1995), 37-61.
16. Repenning, A. and Ambach, J. Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing. *Proceedings of the Visual Languages Conference*, Boulder, CO, 1996.
17. Repenning, A. and Ioannidou, A. Behavior Processors: Layers between End-Users and Java Virtual Machines. *Proceedings of the Visual Languages Conference*, Capri, Italy, 1997.

18. Repenning, A. and Smith, J.T. II, Perrone, C. Agentsheets Common Ground: Shared Visual AgenTalk. *Proceedings of COOP*, Juan-les-Pins, France, June 1996.
19. Smith, E.L. and Anderson, C.W. Plants as Producers: A Case Study of Elementary Science Teaching. *Journal of Research in Science Teaching 21*, 1984, pp. 685-698.
20. Wharton, C., Rieman, J., Lewis, C. and Polson, P. The cognitive walkthrough method: A practitioner's guide. In J. Nielsen and R. Mack (Eds.), *Usability Inspection Methods*. Wiley, New York, NY, 1994.
21. Wharton, C. and Lewis, C. The role of psychological theory in usability inspection methods. In J. Nielsen and R. Mack (Eds.) *Usability Inspection Methods*. Wiley, New York, NY, 1994.
22. White, B. and Frederiksen, J. (1994), Using Assessment to Foster A Classroom Research Community, *Educator 1994*, 19-26.
23. Wong, E.D. Students' Scientific Explanations and the Contexts in Which They Occur. *The Elementary School Journal 96*, 5(1996), 495-509.