

L. LOUCA¹, A. DRUIN, D. HAMMER, D. DREHER

STUDENTS' COLLABORATIVE USE OF COMPUTER-BASED PROGRAMMING TOOLS IN SCIENCE: A DESCRIPTIVE STUDY

Submitted to CSCL Conference 2003

Abstract: This paper presents a small-scale study investigating the use of two different computer-based programming environments (CPEs) as modeling tools for collaborative science learning with fifth grade students. We analyze student work and conversations while working with CPEs using *Contextual Inquiry*. Findings highlight the differences in activity patterns between groups using different CPEs. Students using *Stagecast Creator (SC)* did twice as much planning but half as much debugging compared with students using *Microworlds (MW)*. Students working with MW were using written code on the computer screen to communicate their ideas whereas students working with SC were using the programming language to talk about their ideas prior to any programming. We propose three areas for future research. (1) Exploring different types of communication styles as compared with the use of different CPEs. (2) Identifying students' nascent abilities for using CPEs to show functionality in science. (3) Further understanding CPEs' design characteristics as to which may promote or hamper learning with models in science.

1. INTRODUCTION

There has long been an interest in using computer-based programming environments (CPEs) for modeling in science learning (diSessa, Abelson, Ploger, 1991; Louca and Constantinou; Redish and Wilson, 1993; Sherin, 1996; Sherin, diSessa, and Hammer, 1993; White and Fredriksen, 1998; Wilensky and Resnick, 1999; Smith and Cypher, 1999; McCoy, 1996). Traditional research has investigated different ways that CPEs can be used in developing conceptual understanding and reasoning skills (for examples see Enkenberg, 1990; De Corte, Verschaffel, Schrooten, Olive and Vansina, 1993). More recent research has been exploring the use of CPEs for modeling with older students.

In this study we investigate the ways young learners use CPEs as scientific modeling tools for developing models of physical phenomena. Two different CPEs (Microworlds (MW) and Stagecast Creator (SC)) were used by nine fifth graders as tools for communicating and investigating their ideas, and developing models of relative motion.

Modeling is an essential part of science. Science proceeds through the construction and refinement of models, and learning science entails learning how this happens, in addition to learning particular models scientists have constructed (Constantinou, 1996; Golin, 1997). That is, learning in science largely entails learning the process of developing and refining models (National Research Council, 1990; White and Frederiksen, 1998).

In using modeling as means for learning science, the process of model development and model deployment (Constantinou, 1996; Louca and Constantinou) may be compared to the process of writing and implementing a computer program. Most powerfully, modeling can be carried out through a computer program, when the program itself becomes the scientific model. In this way, the programming language becomes the design medium for the

¹ 2226 Benjamin Building, University of Maryland, College Park, MD 20742, USA LLouca@umail.umd.edu

scientific model and the programming outcome becomes a way of clearly articulating one's understanding about scientific phenomena.

1.1 Computer programming tools as means for modeling in science

Having to design a model of a physical phenomenon, students have to *deconstruct their understanding* of that physical mechanism into small programmable pieces in order to transform an idea in science into specific, technically precise code. As such, programming can be a useful tool for understanding and thinking about science, serving a role traditionally reserved for mathematics. (See Sherin (1996) for an example.)

Unlike mathematics, *a program can be run* on a computer and its results observed, allowing an iterative process of testing and debugging. In addition, *the code itself can be more easily read and explained* than algebra for instance. Rather than equations depicting relations among quantities, lines of code represent procedural instructions that students can read and follow, thinking through what the computer is being told to do.

Programming tools that we investigated in this study differ from prepackaged computer simulations of physical phenomena. Simulations are pre-constructed models which students can explore and possibly alter some of their parameters (program inputs). With CPEs, in contrast, *learners participate in developing and debugging the actual code to generate a simulation* and as such are engaged in the process of developing models.

1.2 Different Types of Programming Languages

Currently there are a number of CPEs designed for young learners. We focus our discussion below on the main differences of the two CPEs that we used, MW and SC.

CPEs differ in the programming language they use. MW uses formal textual programming language. It includes the capability of using advanced graphical representations (e.g. import sound, creating animations etc), but those are limited only to the programming outcome and are not used in the process of programming. In SC, however, programming is done by demonstration using "drag-and-click" techniques with only the computer mouse (Smith and Cypher, 1999). The user creates a script of visual "if-then rules," determining an action for a given situation. Figure 1 provides an example of programming a character to move on the computer screen using the two different CPEs.

Given the different programming languages, *different CPEs require different types of programming strategies.* In SC objects are directly manipulated; programming is creating rules for the objects' behaviors. In MW, however, programming only involves typing code that describes in a procedural manner the objects' behaviors without any physical manipulation of the objects.

The representation of objects also varies. SC uses analogical representation: an object is represented in the same way in both the programming and the outcome level (Smith and Cypher, 1999). In MW, however, objects are represented by an image (turtles can take any shape) in the output window and by a given name in the programming code. Thus, the way of creating, running and debugging program varies both in difficulty and complexity.

The representation of physical values also varies. SC distinguishes between variables (boxes) and their relationship with characters (double click on a character to see its variables). In MW, the relation between variables and objects is not physically represented, nor is there a distinction between variables and procedures; everything is written code.

Given the differences in CPEs, we need to better understand which characteristics are useful for young learners and how students use them. It is equally important to learn how the limitations of available CPEs may affect learning in science.

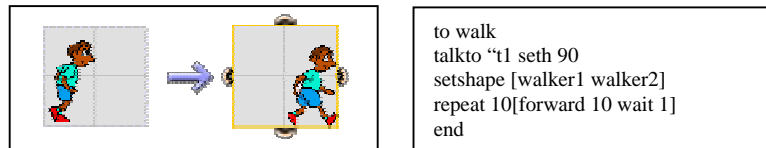


Figure 1. Programming example from Stagecast Creator (left) and Microworlds (right)

2. METHODOLOGY

This descriptive study took place at a suburban elementary school in Bethesda, Maryland, in March 2001. Nine fifth graders were chosen by their science teacher (forth author) to participate in this study among those who volunteered. Students were familiar with using computers but they were not familiar with the CPEs used in this study.

Students met with the first and forth authors for a total of 10 meetings during a two-week interval, for 45-60 minutes for each meeting. Students were divided among three groups (three students in each group). One group worked with SC, and two groups worked with MW. All meetings with the students were video taped and transcribed. Computer monitors were also videotaped. The decision for the two software packages used in this study was based in part on their capability to represent relative motion, which was the physical phenomenon studied during the study.

During the first week of the study students were introduced to the process of programming and to basic programming primitives. Students working with SC used the software tutorial that had previously proven to be a successful tutor (Louca and Constantinou). For MW, the first and third author developed a sequence of activities consisted of simple ready-made programs, designed to show students fundamental functions of the programming environment (examples of simple programs and primitives). Students were asked to make changes to programs that reflected descriptions provided.

During the second week of this study students explored a relative motion situation in which a boy was walking on a train in one direction, while the train was moving fast in the opposite direction. We started with a whole group conversation, in which students decided to describe the situation from the perspective of a person standing on the ground. During the next three meetings students worked with the CPEs they learned, to develop a model of the relative motion situation. In the last meeting students presented each other their designs.

2.1. Methods of Analysis: Contextual Inquiry

Video data of children's work with the CPEs were analyzed using a modified version of Contextual Inquiry (Druin, 1999; Druin, 2002), a participant observation technique developed for analyzing data of children's activities while working with technology. In our modified version of contextual inquiry, adults' roles were minimized to walk around to different groups prompting students' clarifications for their designs. Field notes were also taken after each meeting.

The first author reviewed video data from the second week of the study and transcribed all student conversations while working with CPEs. Two 10-minute segments were chosen from each group, one from the first and the other from the third day of the second week of the study. Transcripts were transferred in a cell-based diagram (matrix) with four columns. The first column included the transcript of the conversation; in the second column we identified children's activities during particular segments of their conversation. In third column we identified activity patterns and the fourth column to identify the type of the students' interaction (verbal/non-verbal, with adults/peers).

The first author coded working patterns for the first chosen segment from each group. Several working patterns have been adapted from another study (Druin, 1999) and several new working patterns were identified. He first read the three transcripts identifying different activities. Activities were coded as activity pattern if students were using them for more than five times during the 10-minute period. Then the first and second author reviewed all activity patterns to check for consistency of codings across different groups, and final decisions for the categories of working patterns were made. At that point, we decided to use the same categories of coding for students working with the two different CPEs.

After an initial analysis, the first author watched the video segments of the conversations to add non-verbal student interactions. This way, in moments of silence we determined what students were doing. These non-verbal interactions were added to the transcript as descriptions of interactions, and where applicable coded as activity patterns.

2.2. Methods of Analysis: Analysis of Student Thinking

A second goal for this study was to investigate different thinking strategies that students were using throughout this study, within the context of using CPEs. For this purpose the first author reviewed all videos from the second week of the study, identifying snippets where students used programming skills and/or the programming language to address issues in science. The first and the third author watched and analyzed the segments identified.

For this analysis, we followed examples of similar work in mathematics education (e.g. Ball, 1993) and in science (e.g. Gallas, 1995). Ball (1993) and Gallas (1995) analyzed student thinking in mathematics and science, using the communication among students and the teacher in regular classrooms. They used transcripts of student conversations to isolate students' ideas for the topic under investigation, tried to identify their reasoning behind their ideas and the different ways students articulated those ideas. Our goal was to approach data from an additional theoretical perspective, trying to provide justification for the identified working patterns.

3. RESULTS

3.1 Contextual inquiry

Figure 2 provides a comparison of the activity patterns (by percentage) between groups using different CPEs. For MW, the figure represents the means of the percentages of the working patterns of the two groups for students. Percentages provide the number of times that students were observed doing something relatively to the total number of activities during the 10-minute segments that we analyzed.

We identified six major categories of activity patterns as repeated patterns across the different groups. These are programming, program planning, testing/running program,

debugging, describing situation, talking about science and other. When students were actually writing code (in MW) or making rules (in SC), we coded that as programming. An activity was coded program planning if students were talking about how to develop a program for their design. Testing/running program was used to code instances where students were trying out their program(s). We coded for describing situation when students were explaining their programs to their team mates or to adults. When students were talking about the science of their design(s), we coded the activity as talking about science. Lastly, because the descriptive nature of this study, we only coded for the most obvious categories, coding all the other activities under “other” for future analysis.

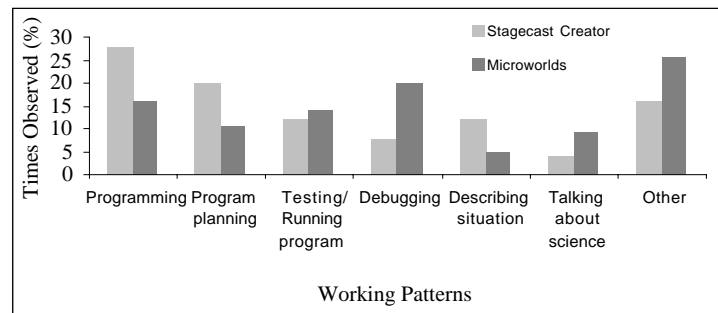


Figure 2: Frequency of students' working patterns using different CPEs

3.1.1. Activity patterns of students working with Microworlds

A majority of their time, students using MW were working as programmers (16%) and debuggers (20%). Another major part of their time was devoted to other activities such as exploring the CPEs' capabilities, and dealing with graphical characteristics of the objects in their designs. Some of their time was also devoted to program planning (10.5%).

Below we provide an excerpt from a conversation between two of the students working with MW. MW has two distinct windows for programming and running programs, and the excerpt below starts while students were typing code in the programming window; their goal was to write a program for the boy to walk on the train.

1. **Alison:** Ok. What is the um (name of the train)...? We'll do that after...
2. **Avery:** (fd)15
3. **Alison:** oh, yea and then... like that?
4. **Avery:** right. Ok.
5. **Alison:** and so now we go back, do we have to make a button?
6. **Avery:** yea.
7. **Alison:** ok. (students go to the output window)
8. **Avery:** and we call it 1.
9. **Alison:** Well, it should be about how it is used. Oh well...
10. **Avery:** It's not capitalized. (Alison starts making the button)
11. **Alison:** It's ok, well, then it's different.
12. **Avery:** Ok. Run it.
13. **Alison:** Run needs more inputs (going back to the programming window).
14. **Avery:** What's that? (pointing on the computer screen)
15. **Alison:** Wait, last time we did that run, setshape, forward, seth 90.
16. **Avery:** But you don't want to go a 90-degree angle.
17. **Alison:** Yes it does!

18. **Avery:** Ok.
19. **Avery:** Why don't we just move forward?
20. **Alison:** Yea, if you move forward, wait no. (several seconds of silence). Remember Loucas said if you do seth 90 you go right. But we want it to go left.
21. **Avery:** No with seth 90 you make a right angle.
22. **Alison:** Right. That would be going right.
23. **Avery:** No a right angle is this.
24. **Alison:** I know, but seth 90 is to go right.
25. **Avery:** No, it's right angle.
26. **Alison:** Yea, I know. When you start, when you push <inaudible> the person goes whenever you want to go up. If you do seth 90 it'll automatically go right. We want to make it go left, not right. Because otherwise he'll be running backward.
27. **Avery:** No it'll automatically go a 90-degree angle.
28. **Alison:** Ok, ok. Do what you want...
(Silence for about 30 seconds, while Avery is typing code)
29. **Avery:** Wait, wait a minute, um we are not um, um is not supposed to be going anywhere, since is supposed to be on the train, right?
30. **Alison:** He is supposed to be going left, but the train is going right?
31. **Avery:** Yea, but, yea ...it (the train) is going forward...(typing code)
32. **Avery:** ...let's see if this works.

In the four first lines of this conversation students were interacting while typing code in the programming window (coded as programming). By line 4 they had typed a simple program that included character animation so that it looked like it was walking (by changing the character's appearance among three different shapes) and code for the motion of the character (forward 15). In lines 5-9 they talked about creating a button in the output window that would correspond to their program (program planning). Buttons enabled students to easily run their programs. In lines 10-11 students created the button (programming). In line 12 students run their program for the first time, and in lines 13 through 18 they were trying to identify the flaws in their program (debugging). In lines 19 through 27 students were trying to reconcile a difference they had about whether the primitive seth 90 (which sets the direction of the "turtle" in 90 degrees) would be appropriate to use (program planning).

Line 28 lasted about 30 seconds, while Avery was typing code in the programming window without any interaction between the two students (programming). In line 29, Avery realized that they did not have to worry about having the boy walking because he "is not supposed to be going anywhere" assuming of course that he is walking at the same speed that the train moves (an issue that that they previously talked about). Alison disagreed with him. He "is supposed to be going left, but the train is going right." It seems that the two students had different views for the same situation. Alison was thinking about the train and the boy independent of each other: they both move in different directions and they both need code that would generate those motions. Avery was rather thinking about the physical system: they both move but, for a person on the ground, the boy would look like not moving; thus, no code is needed for the boy. Meanwhile, during lines 29-31, Avery kept typing code, and in line 32 he ran that program.

As seen in the excerpt above, students were focused in developing programs without flaws. They were writing code, testing and debugging it over and over again until they got a program that ran without any bugs. Because they were trying to write flawless programs, we suggest that it may have gotten in the way of thinking about the physical mechanism that underlies the phenomenon.

We do not suggest that this distraction is a characteristic caused by MW, since we observed similar behaviors with students working with SC (see our discussion later). In an

artifacts analysis (not discussed here) of students' successive models, we found that students moved from Avery's model (only the train had code) to Alison's model (independent codes for each object) to a model with the two characters' speed related.

Overall only 9% of the students' activities were devoted to *talking about science*. Like in the excerpt above, when students started talking about the science related with their programs, programming and debugging would distract them. After line 34, students started looking again for flaws in their programs and didn't revisit the issue they started talking about in lines 29-31. However, we propose that instances like the one in lines 29-31 might suggest that students can use programming media as modeling tools in science.

Lastly, their interactions seemed to be limited to issues related with the programming language itself. Prior to any typing, students talked about the appropriate primitive to use, rather than the appropriate code that would correspond to the physical phenomenon (e.g., use *setheading 90* or *left 90*? vs use *forward 10* or *forward 20*?). The discussion in lines 19-27 above corresponds to a conversation *after* students wrote and tested an initial program. Prior to that, their conversation was about which primitive is appropriate (i.e. lines 5-9). In addition, students used already typed code as the basis of their interactions, which in some cases was simply taking control of the keyboard and correcting each other's code, without any conversations (line 28).

3.2. Activity patterns of students working with Stagecast Creator

Students from both groups were observed similar relative times *testing* their designs (14% for MW and 12% for SC), whereas students working with SC were observed less times *debugging* (8%) their programs. When debugging was needed, they simply created new rules without going back to their original code, as the following example shows.

Drew: Push that button. Now press done and this. (Pointing on the computer screen)

Joe: Oh!... (The rule does not work as they would like.)

Joe: We have to make a new rule, ok?

Michael: Yea

Drew: Oh, wait, I've got an idea. Right click on this (pointing the characters' window on the computer screen) and see if there are any more trains. (There are no more train characters in the characters' window.) Oh, no! (Michael starts making another rule)

Drew: Done. Leave it like that and then <inaudible> (Michael is testing their new rule)

Joe: Cool. Let's leave it like that.

In this excerpt students found out that their rule was not what they wanted. Joe suggested making a new rule, and instead of changing the existing rule, Michael started to make a new rule for the same situation. We coded this activity as programming and not as debugging, which resulted higher frequency of programming (28%) and lower frequency of debugging. Interestingly, if we add the programming and debugging percentages of each group, groups working with different CPEs have in total the same percentage devoted to those two activities. Because students were not debugging their designs, their programs contained multiple rules for the same behavior. This seemed to be another distraction, because it is more difficult to debug when having several rules for the same behavior.

Students working with SC were observed program planning (20%) twice as many times as students working with MW. In their conversations, SC's students used programming language to explain to each other their ideas (program planning) prior to any programming. Unlike students working with MW, they did not spend as much time trying to figure out how to program a particular situation; nor did they spend as much time talking about the

code itself. We are proposing the possibility that there might be a difference in the type of communication between members in the individual groups. Here is an example from SC:

Michael: Walking across the train you mean? Like a super hero?

Drew: Yea.

Michael: Remember how I made that picture (with the four squares), remember how I said that he jumps in the forth square and because of the wind we make him come to the first square?

Drew: He is not actually moving, though...

Since students working with SC did not spend time talking about what code to use in their programs, we propose that it is possible that SC's programming language might be easier for young programmers (Gilmore, Pheasey, Underwood and Underwood, 1995; Smith, Cypher and Tesler, 2000). This suggests that SC might be a better programming environment for providing students with means of communicating their ideas: its programming language seemed to serve this function quite often.

The structural design of SC seemed to interfere with of debugging programs. Learning through developing models of physical phenomena is an iterative process of developing, testing and revising models. Revisions are usually based on the existing models; students need to identify the part of their code that has a flaw and replace it. However, students using SC preferred to make new rules instead of debugging already existing ones. We suggest that this may interfere with the process of modeling, as developing, comparing models with reality and debugging.

Finally, talking about science was only 4% of their activities for students in the SC group. They seemed to be more concerned about how their designs looked rather than how they functioned. Having dealt with the visual details, however, students could easily have been prompted to evaluate their models and start thinking about the physical mechanisms underlying the phenomenon.

4. DISCUSSION

The study we report in this paper, reveals three areas of interest that are related with the ways that young learners use CPEs as modeling tools in science. We focus our discussion here below on the possible effects of CPEs' characteristics on students' use.

4.1. Students' use of CPEs

One issue that stands out from our results is the apparent few conversations about science that occurred during group work. It is possible that this was a result of the way that students perceived the use of CPEs. CPEs can also be used as visualization tools and is likely that students would use them as tools to show how reality looks and not how physical systems function, which would explain the relatively few science discussions in the study.

Our results are consistent with a number of studies (diSessa, Hammer, Sherin and Kolapakowski ,1991; Penner, Giles, Lehrer and Schauble, 1997) in which students were initially inclined to use different types of modeling media as visualization tools that depict how reality looks rather than how it functions. In these studies, however, after prompted by researchers, students were able to use modeling media to show functionality. Rather than seeing students have to develop such abilities, we propose that they need to develop more reliable access to nascent abilities they already have, following a number of studies that argue that students have a variety of inquiry skills that might be context dependent (Louca and Hammer, 2001; Samarapungavan, 1992). There were instances when students would

talk about physical mechanism, even though they were relatively low. Thus, we suggest that we need to further study the ways that students use CPEs in the context of learning science.

4.2. Different CPEs promote different communication patterns?

Preliminary analyses of within group conversations indicate different ways that students used programming languages as communication media. SC's language was used for communicating ideas to be/that had been programmed. Students were using the programming language to explain their ideas to each other. On the other hand, students were using MW's programming language as a means for writing and reading each other's ideas (rather than orally communicating them). In both situations, different types of programming languages seem to promote different types of communication. We need, however, to identify the different elements of these CPEs that may support these different types of communication among students. For instance, it might be possible that SC's language is easier for students and as such it was used to communicate ideas in science and the mechanism that was underlying them. On the other hand, it is equally possible that *reading* rules (code) in SC is just more difficult than in MW, where written code seemed to be a meaningful means of communication.

4.3. Different "design" characteristics may result different use of the CPEs

SC is an environment specifically designed to facilitate students' use of programming (Gilmore, et al, 1995; Smith, Cypher and Tesler, 2000). Programming in SC is object-oriented. Code is written for particular objects and it is physically linked with the objects. Because of this, the activity of programming uses a "sub-application" within SC, independent of the place where the code is stored, and as importantly, in order to program the user does not need to go to the window where code is stored. However, for debugging, the user needs to go to the appropriate character's window to locate the existing code. This is quite different from MW, where all the code for every object is located in the same window, where new code can be written or existing code can be changed.

We suggest that this particular design may have caused the relatively few number of debugging activities that were observed from the SC group. When flaws were identified, students decided to write new code instead of correcting their existing work. It is possible we argue, if they had to go to the window where the initial code was in order to create new rules, it might have helped them to debug instead of writing new code. We use this example to make a more general point: there is a need to identify the characteristics of CPEs that support student thinking and learning in science. We argue that it is possible that CPEs' different design characteristics may promote or hamper learning with modeling in science.

5. CONCLUSION

This study identifies how young learners use different CPEs in science. Limitations include the small number of students that participated. It is possible that some results are due to students' individual characteristics rather than to the programming environment's differences. Also limitations include the fact that percentages do not represent the amount of time that students spent on each activity. Rather percentages show the times that students actually did this activity in the 10-minute period. Our future research focus on what characteristics should future CPEs have, including those that might tap into students' nascent abilities for using CPEs to show functionality in science.

6. NOTES

This work was partially supported by the “Case Studies for K-8 Student Inquiry in Physical Science” project funded by the NSF at the University of Maryland, and partially funded by a NSF Career Award for Allison Druin’s work on the “Classroom of the Future”.

7. REFERENCES

- Ball, L., D. (1993). With an eye on the mathematical horizon: dilemmas of teaching elementary school mathematics. *The elementary school journal*, 93 (4), 373-397.
- Constantinou, C., P. (1996). The Cocoa microworld as an environment for modeling physical phenomena. *International Journal of Continuing Education and Life-Long Learning*, 8 (2), 65 - 83.
- diSessa, A. A., Abelson, H., and Ploger, D. (1991). An overview of Boxer. *Journal of Mathematical Behavior*, 10, 3-15.
- diSessa, A., A., Hammer, D., Sherin, Br., and Kolapakowski, T. (1991). Inventing graphing: Meta-representational Expertise in Children. *Journal of Mathematical Behavior*, 10, 117-160.
- De Corte, E., Verschaffel, L., Schrooten, H., Olive, H. and Vansina, A. (1993). A logo based tool-kit and computer coach to support the developments of general thinking skills. In T. M. Duffy, J. Lowyck, D. H. Jonassen (Eds.), *Designing Environments for Constructive Learning* (p. 109-124). Germany: Springer-Berlin Heidelberg.
- Druin, A. (1999) Cooperative Inquiry: Developing New Technologies for Children with Children. *CHI99 Conference Proceedings* ACM Press, 223-230.
- Druin, A. (2002) The Role of Children in the Design of New Technology. *Behavior and Information Technology (BIT)* 21(1), 1-25.
- Enkenberg, J. (1990). Computer programming, Logo and development of thinking. In G. Shuyten, M. Valcke (Eds.), *Proceedings of the Eurologo '89 Conference* (p. 150-165). Washington, DC: IOS.
- Gallas, K. (1995). *Talking their way into science: hearing children's questions and theories, responding with curricula*. NY: Teachers College Press.
- Gilmore, D., J., Pheasey, K., Underwood, J., and Underwood, G. (1995). Learning graphical programming: An evaluation of KidSim. In k. Nordby, P. H. Helmersen, D. J. Gilmore and S. A. Arnesen (Eds.), *Human-Computer Interaction: Interact '95* (p. 145-150). London: Chapman and Hall.
- Golin, G. (1997). Structure of scientific knowledge and curriculum design. *Interchange*, 28 (2,3), 159-169.
- Louca, L. and Constantinou, C. *Using computer-based microworlds for constructing modeling skills in physical science: an example from light*. Manuscript submitted for publication.
- Louca, L. and Hammer, D. (2002, April). Elementary student inquiry in physical science: Answers, explanations, and arguments in a 5-6th grade discussion about a dropped pendulum. Paper presented in *AERA 2002 conference*, New Orleans, LA.
- National Research Council. (1990). *National Science Education Standards*. Washington, DC: National Academy.
- 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.
- Redish, E. F. and Wilson, J. M. (1993). Student programming in the introductory physics course: M.U.P.P.E.T. *American Journal of Physics*, 61 (3), 222-232.
- Samarapungavan, A. (1992). Children’s judgments in theory choice tasks: Scientific rationality in childhood. *Cognition*, 45, p. 1-32.
- Sherin, Br. (1996). *The Symbolic Basis of Physical Intuition. A Study of Two Symbol Systems in Physics Instruction*. Unpublished dissertation Thesis.
- Sherin, Br., diSessa, A. A., and Hammer, D. (1993). Dynaturtle revisited: Learning physics through collaborative design of a computer model. *Interactive Learning Environments*, 3 (2), 91-118.
- Smith, D., C. and Cypher, Al. (1999). Making programming easier for children. In A. Druin (Ed.), *The design of children's technology*. San Francisco: Morgan Kaufmann Publishers, Inc.
- Smith, D., C., Cypher, A. and Tesler, L. (2000). Novice programming comes of age. *Communications of Association for Computer Machinery (ACM)*, 43(3), 75-81.
- White, B. Y. and Frederiksen, J. R. (1998). Inquiry, modeling and metacognition: Making science accessible to all students. *Cognition and Instruction*, 16 (11), 3-118.
- Wilensky, Ur., and Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. *Journal of Science Education and Technology*, 8 (1), 3-19.