Personalizing and Discussing Algorithms within CS1 *Studio Experiences*: An Observational Study

Christopher D. Hundhausen and Jonathan Lee Brown
Visualization and End User Programming Lab
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164–2752
+1 (509) 335-6602
{hundhaus, jbrown}@eecs.wsu.edu

ABSTRACT

Pedagogical algorithm visualization technology aims to assist learners in understanding the dynamic behavior of computer algorithms. A key trend in past experimental studies is that learners benefit most when they are actively engaged with algorithm visualization technology. Inspired by this trend, we are exploring the pedagogical value of a novel active learning activity—the *Studio Experience*—within the context of an introductory CS1 unit on algorithmic problem-solving. In a Studio Experience, student pairs are given algorithm design problems, e.g., "design two alternative algorithms that reverse the values in a list." They are tasked both with constructing algorithmic solutions and accompanying visualizations, and with presenting their visualizations for feedback and discussion in a session modeled after an architectural "design crit." Through an observational study of studio experience sessions in which students used two alternative forms of visualization technology—art supplies and a computer-based tool—we gained insight into (a) the processes by which students construct visual presentations of algorithms, (b) the characteristics of their visual presentations; (c) the nature of conversations mediated by visual algorithmic solutions; and (d) the kind of visualization technology that best supports these activities. Based on our results, we suggest improvements to the approach, and propose an agenda for future empirical studies.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Info. Science Education—*computer science education; curriculum*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*animations*, *evaluation/methodology*.

General Terms

Algorithms, Experimentation, Human Factors

Keywords

Algorithm visualization, CS1, algorithmic problem solving, observational studies, studio-based instruction

1. INTRODUCTION

Pedagogical algorithm visualization (AV) technology (see [24] for a review) produces graphical representations of the dynamic behavior of computer algorithms. The goal of the technology is to assist learners in understanding how computer algorithms work. While early research into pedagogical AV focused on developing AV systems, more recent research has focused on empirically evaluating the pedagogical effectiveness of the technology [24]. Focusing on a significant subset of this work, a recent meta-study of 24 experimental studies of AV effectiveness [14] identified an important trend: the more actively learners were involved in activities involving AV technology, the better they performed.

Given this trend, a key focus of recent AV research has been to explore approaches and technology that get learners more actively engaged with AV technology. Naps *et al.* [18] present a framework of five progressively active levels of learner engagement that have been considered by AV research:

Level 1: Viewing a visualization (see, e.g., [22]).

Level 2: Responding to questions concerning a visualization (see, e.g., [3]).

Level 3: Changing a visualization (see, e.g., [16]).

Level 4: Constructing a visualization (see, e.g., [23]).

Level 5: Presenting a visualization for feedback and discussion (see, e.g., [11]).

Inspired by the "studio-based" instructional method used to teach architectural design (see, e.g., [2]), we have been exploring approaches and technology to facilitate levels 4 and 5 in the above framework: *visualization construction* and *presentation*. Our general approach has been to give groups of students a particular algorithm design problem to solve. They are asked to construct an algorithm visualization that illustrates their solution, and then to present that visualization to their peers and instructor for feedback and discussion in a presentation session modeled after an architectural "design crit" [6].

In architectural design education, "design crits" are seen as providing an ideal forum for stimulating educationally beneficial discussions. In addition to eliciting constructive comments about each student's specific design and design rationale, "design crits" typically generate higher-level discussion about the general principles and methods being explored in the course. Likewise, we hypothesize that, in AV presentation sessions, student-constructed visualizations can serve as powerful *mediational resources* [19] that bridge the gap between expert and learner perspectives, ultimately enabling pedagogically beneficial conversations about algorithm design to take place.

In previous studies, we explored this "studio-based" approach within the context of a junior-level algorithms course [11]. In this course, student groups were given "visualization assignments" in which they had several weeks to develop their own visualizations of algorithms under study. They then presented their visualizations for feedback and discussion during presentation sessions scheduled after the assignment due date.

A key observation in these studies was that students benefited from constructing and presenting their own visualizations not only because this exercise increased their motivation and level of interest in algorithms, but also because it stimulated meaningful discussions about algorithms. At the same time, we observed that the *type* of AV technology used by students had a significant impact both on the focus of students' activities during the construction phase of the assignment, and on the focus of the conversations during the presentation sessions. In particular, when students used Samba [23], a "high tech" (computer-based) AV tool, to construct their visualizations, they tended to spend inordinate amounts of time steeped in low-level implementation details; consequently, they tended to share "programming war stories" when they ultimately discussed their visualizations with the class. In contrast, when students used "low tech" materials (pens, paper, scissors) to construct their visualizations, they tended to focus their attention squarely on the algorithm itself both in the construction phase, and in the conversations that took place in the presentation phase.

Given the potential value of "low tech" visualization construction and presentation activities documented in our prior studies of a junior-level course, might learners benefit from engaging in these activities earlier in their computer science careers, when they are first learning to program? To explore this possibility, this paper reports on an observatinal study of AV construction and presentation activities within an "algorithms first" [15] CS1 course. Focusing on a series of "studio experiences" that augmented an introductory unit on algorithmic problem solving within two separate offerings of a CS1 course at Washington State University, this study set out to address three key research questions:

- RQ1. Will the process of constructing *personalized* visual representations that depict solutions to algorithm design problems help CS1 students to better understand their solutions?
- RQ2. Will the process of presenting personalized visual representations engage students and the instructor in pedagogically beneficial conversations about the correctness and procedural behavior of their solutions?
- RQ3. What form of AV technology can best support the above two processes: simple art supplies ("low tech") or ALVIS [ref], a specialized computer-based AV development environment we have developed for this purpose ("high tech")?

In the remainder of this paper, we present the observational study, its key results, and their implications for CS1 pedagogy. After reviewing related work in Section 2, we present key background information on the study in Section 3. In Section 4, we describe the field techniques employed by our study. Section 5 presents our key observations, while Section 6 uses our observations as a basis both for addressing our research questions, and for identifying directions for future empirical research. Finally, we conclude in Section 7 by considering the implications of our findings for CS educators.

2. RELATED WORK

A legacy of previous computer science education research shares our interest in empirically investigating the novel use of AV in computer science courses. In work that greatly inspired our initial ethnographic studies of a junior-level algorithms course [11], Stasko [23] gave visualization construction (but not presentation) assignments to students in his junior-level algorithms course. Through questionnaires administered at the end of the course, he learned that students generally enjoyed the assignments and felt that the assignments helped them to better understand the algorithms they visualized. Likewise, in a second-semester programming course, Gurka [7] relied on student diary keeping to explore students' use of an AV package, as compared to their use of other course resources and activities. She found that student use of AV constituted 35% of their course activity, and that they most often used algorithm visualization to help them complete course programming assignments.

More recently, Hübscher-Younger and Narayanan [9, 10] conducted a series of empirical studies that explored a novel pedagogical approach in which second- and third-semester CS students created and posted their own algorithm visualizations to the web; peers could then view, discuss, and rate each other's visualizations on-line. In addition to analyzing students' ratings of each other's visualizations, the researchers performed a metaphoric analysis of students' visualizations similar to the one we present in Section 5.2.2. Their results indicated that students who constructed their own visualizations learned significantly more than students who only rated and discussed other students' visualizations. Moreover, students' ratings of each other's visualizations varied most widely with respect to the characteristics of pleasure and salience. A follow-up study showed students learned most from student-constructed visualizations that were rated highest with respect to those characteristics.

The work presented here differs from the above studies of AV in three key respects: (a) its focus on a CS1 course as opposed to an upper-division CS course, (b) its focus on AV presentation, in addition to AV construction; and (c) its commitment to using a broad range of ethnographic field techniques to obtain a rich perspective on learning activities involving AV.

A large body of computer science education research shares our goal of increasing the number of students—especially underrepresented students—who successfully complete the CS1 course. Using specialized visually-oriented programming environments and AV technology to enhance CS1 courses is one means to that end, and has been explored by several CS educators, including Naps [17], Dann et al. [5], Ben-Bassat Levy et al. [1], and Carisle et al. [4]. While some of this work shares our commitment to rigorously evaluating the educational benefits of learning exercises involving AV technology, none of it focuses on the use of AV technology as part of a novel, "studio-based" teaching approach.

3. STUDY BACKGROUND

The observational study reported here focused on two consecutive offerings of CptS 121 ("Program Design and Development"), the introductory computer science course at Washington State University. Offered within the School of Electrical Engineering and Computer Science, the course is a requirement for all engineering majors in the School. Because it caters to a broad engineering audience, the instructional language of this semester-long course is C; the textbook used in the course is by Hanly and Koffman [8].

Like many other CS1 courses throughout the country, the CptS 121 course has suffered from a notoriously high drop-out rate that hovers between 30% and 40%. To address this problem, we have been exploring an introductory five-week "algorithms-first" unit that aims to provide a gentler introduction to programming—one free of the baggage of a traditional programming language like C. Supported by the Schneider and Gersting text [20], our "algorithms first" unit uses plain English and pseudocode to explore algorithmic problem solving.

In the "algorithms-first" unit, three weekly lectures motivate, formulate, and solve progressively more complex algorithm design problems. Augmenting these lectures is a series of Studio Experiences in which students create and present their own visual solutions to algorithm design problems related to the ones presented in lectures. These Studio Experiences were the focus of the study presented here.

In the Fall, 2004 offering of the course, we observed a total of two studio experience sessions, each of which contained roughly 22 students. In the remainder of this paper, we will refer to these sessions as the *Art Supply* sessions, because students in these sessions used art supplies to construct their visualizations. In the Spring, 2005 offering of the course, we observed a total of two Studio Experience sessions, each of which contained roughly 22 students. Henceforth, we will refer to these sessions as the *ALVIS* sessions, because students in these sessions used the ALVIS programming and visualization environment [12] to construct their visualizations.

In the remainder of this section, we provide more background on the participants in the study, and the "studio experiences" in which they engaged. We begin by describing the students, instructors, and teaching assistants who participated in the study. Next, we outline the structure and content of the Studio Experiences we investigated. Finally, we describe the "low tech" and "high tech" versions of the programming and AV technology used by students in the two alternative offerings of the course we studied.

3.1 Participants

Approximately 170 students were enrolled in the first offering of CptS 121 considered by our study; roughly 45 of these students participated in the two Art Supply sessions we observed. The enrollment of the second offering of CptS 121 considered by this study was about half that of the first: approximately 90 students. Of these, about 45 students participated in the two ALVIS sessions we observed.

Students who participated in the Studio Experiences we observed ranged in age from 18 to 45, with the vast majority of the students between 18 to 20 years of age. Roughly 90 percent of the students in these courses were males. The students in this course varied greatly with respect to their previous programming experience. Roughly 40 to 50 percent of the students came in to the course with no prior programming experience. The remaining 50 to 60 percent of the students had at least some programming experience, either in a high school course, or on their own.

In addition, five graduate teaching assistants (not including the authors) ran the Studio Experiences that we observed. These teaching assistants took attendance, gave introductory instructions, made themselves available for help while students developed their algorithmic solutions, and helped to facilitate the presentation sessions at the end of each Studio Experience.

3.2 The Studio Experiences

Two Studio Experience sessions were included in each of the CptS 121 courses we considered. The Studio Experience sessions took place during the regularly-scheduled two hour and fifty minute lab periods in the second and fourth weeks of each course. It is important to note that, in this paper, we consider only those observations made in the *second* Studio Experience session of each semester, and that the Studio Experience sessions were identical across semesters, except for the fact that students in each semester used different technology to create their algorithmic solutions and accompanying visualizations (art supplies vs. ALVIS; see next subsection).

In the Studio Experience sessions we observed, students were encouraged, but not required, to work in self-selected pairs. Each pair was given one of twelve different algorithm design problems, which required students to rearrange, merge, or otherwise manipulate the integer elements of arrays. Table 1 presents three examples of the algorithm design problems that were assigned.

Students in *all* Studio Experience sessions wrote their solutions in *SALSA*, the pseudocode-like language that is supported by the ALVIS environment (see Table 2 for a summary of the language). Students had been taught SALSA in the previous three weeks' worth of lectures, and were provided with a one-page SALSA quick reference guide for use in the Studio Experience. Most solutions required fewer than 30 lines of SALSA, and involved nested iteration, and possibly the creation of temporary arrays.

In addition, pairs were required to create visualizations of their solutions, and they were encouraged to personalize those visualizations by developing a storyline to portray the step-by-step behavior of their algorithmic solutions. Finally, during the last 45 minutes of each Studio Experience, student pairs were required to present their visual solutions to their peers and instructor for feedback and discussion. In particular, student presenters were asked to give a brief description of the problem they solved. Following that, each pair was asked to walk through each of their visualizations, providing a "play by play" of the key algorithmic steps. Finally, the pair was asked to provide a big-O analysis of their alternative algorithms, along with a justification for their analysis.

Table 1. Three Sample Algorithm Design Problems

PROBLEM TITLE	PROBLEM DESCRIPTION
Concatenate Lists	Design two alternative algorithms that create an array containing *n* random integers between 1 and 50. Your algorithms should create a new list that is twice as long as the original list. The new list should contain two copies of the sequence of elements that appear in the original array: one full sequence followed by a copy of that sequence.
Reverse Order Inputted Numbers	Design two alternative algorithms that prompt the user to input integer values, one at a time, until a negative integer is entered. After each integer is inputted, your algorithms should insert it into an array. The end result should be that the values appear in descending order from largest to smallest. Note that the negative value should not be inserted into the list.
Merge Lists	Design at least two alternative algorithms that create two lists containing *n* random integers between 1 and 100 that are in ascending order. After that, your algorithms should build a third list of length 2*n* in which the elements of the two source lists are in ascending order.

Table 2. Summary of the SALSA Pseudocode Language

COMMAND	MEANING
`create`	Creates new variables, arrays, and array indexes (special variables that reference array cells)
`set`	Creates and sets the value of new variables and array indexes on the fly, or changes the values of existing variables, arrays, and array indexes
`input`	Prompts the user for a variable's value; the variable is created if it does not already exist
`populate`	Fills the empty cells of an array with new values
`print`	Outputs a value to the user
`if…elseif… else…endif`	Specifies blocks of code that execute conditionally based on the results of Boolean tests
`while… endwhile`	Specifies a loop that executes as long as the Boolean test evaluates to true
`move`	Moves a variable, array, or array index to a new location
`swap`	Causes two variables or array indexes to change positions
`say`	Causes a variable or array index to "speak" a text string through an animated speech bubble

3.3 Supporting Technology: Art Supplies and ALVIS

As mentioned above, the problems and format of the Studio Experiences we observed were identical across the two semesters of our study in all respects except one: the technology used by students to create their algorithmic solutions and accompanying visualizations.

In the Art Supply sessions that took place in the first semester of the study, students were asked to use a text editor (Notepad or Word) to develop pseudocode solutions to their algorithm design problems. To make their solutions more concrete, they were then asked to use transparencies, pens, and scissors to create homemade animations illustrating the operation of their algorithms on sample input data sets (see Figure x for a sample visualization). Finally, students used an overhead projector to present their homemade animations to the class.

In contrast, in the ALVIS sessions of the following semester, students used a new version of the original ALVIS software [13] called ALVIS Live! [12] to code their algorithms and customize visualizations of those algorithms. Pictured in Figure 1, the ALVIS Live! environment enables novice programmers to develop single-procedure algorithms in a pseudocode-like language called SALSA. On every edit, the line of algorithm code being edited in the "Script Window" is reevaluated, leading to the dynamic update of an accompanying visualization in the "Animation Window." Alternatively, the user can use the Toolbox Tools on the right to directly lay out and animate program objects (variables and arrays) in the Animation Window. Through such direct manipulation, SALSA code is dynamically inserted into the Script Window on the right.

In order to facilitate "radically dynamic" editing, the execution arrow, which can also be moved around with the Execution Controls, follows the editing caret around in the Script Window. Thus, whenever the programmer moves the caret to a new line, the script is automatically executed (either forwards or backwards) to that point, and the visual representation of the program displayed in the Animation Window is updated accordingly.

ALVIS Live! generates standard visual representations of variables, arrays, and array indexes (see Figure 3). In addition, ALVIS Live! supports three customization features designed to support storytelling. First, users can customize a variable's representation simply by clicking on the variable and choosing a pre-defined picture from a gallery, or sketching out a new picture with a sketchpad editor. Second, users can create a custom background to serve as a backdrop for a story simply by choosing a pen tool and directly sketching on the background. Third, users can introduce "dialog" into their animations through the use of a "say" command, which allows a variable to "speak" through a cartoon-like speech bubble. Students in the ALVIS sessions were encouraged to use these features to build story-based visualizations, which they then presented on a big screen using an LCD projector (see Figure x for a sample ALVIS custom visualization).

4. Ethnographic Field Techniques

Our study employed five different ethnographic field techniques. First, we employed *participant observation* as a primary field technique in all sessions. In all of the sessions we observed, we assumed the role of volunteer teaching assistants. In this capacity, we both observed student pairs, and assisted them on request. In addition, the first author assisted the teaching assistant in facilitating the presentation sessions; the second author passively observed the presentation sessions.

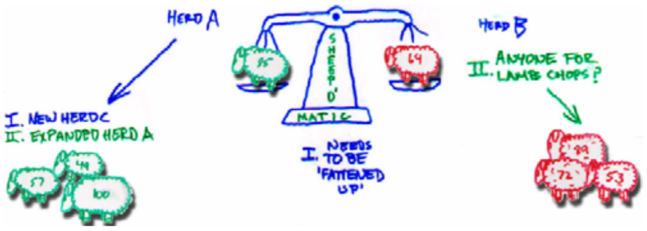
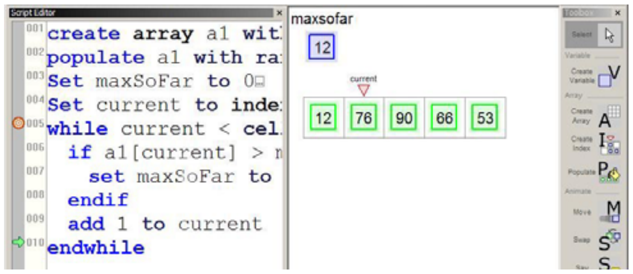


Figure 1. Sample Art Supply Visualization



```
create array a1 wit
populate a1 with ra
Set maxSoFar to 0
Set current to inde
while current < cel
  if a1[current] > n
    set maxSoFar to
  endif
  add 1 to current
endwhile
```

Figure 3. Sample ALVIS Standard Visualization

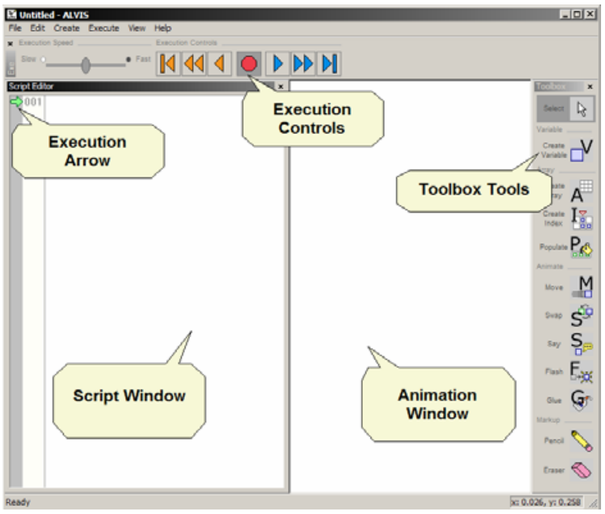
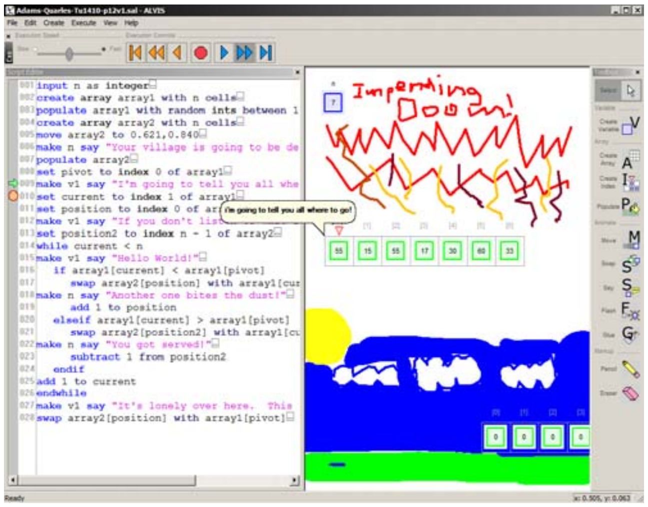


Figure 2. The ALVIS Live! Environment



```
input n as integer
create array array1 with n cells
populate array1 with random ints between 1
create array array2 with n cells
move array2 to 0.621,0.840
make n say "Your village is going to be de
populate array2
set pivot to index 0 of array1
make v1 say "I'm going to tell you all whe
set current to index 1 of array1
set position to index 0 of ar
make v1 say "If you don't list
set position2 to index n - 1 of array2
while current < n
make v1 say "Hello World!"
   if array1[current] < array1[pivot]
      swap array2[position] with array1[cur
make n say "Another one bites the dust!"
      add 1 to position
   elseif array1[current] > array1[pivot]
      swap array2[position2] with array1[cu
make n say "You got served!"
      subtract 1 from position2
   endif
add 1 to current
endwhile
make v1 say "It's lonely over here.  This
swap array2[position] with array1[pivot]
```

Adams-Quarles-Tu1410-p12v1.sal - ALVIS
File Edit Create Execute View Help
Impending Doom!
I'm going to tell you all where to go!
55
15
55
17
30
60
33
Ready

Figure 4. Sample ALVIS Custom Visualization

With respect to the first author's role as a participant observer in the study, it is important to point out that students in one of the Art Supply sessions came from a lecture section taught by the first author. We openly acknowledge the conflict of interest posed by the first author's presence as an observer in these two sessions; indeed, it could have been the case that his status as the course instructor influenced student participation. At the same time, we would like to underscore that we observed one additional Art Supply session that included students who came from a lecture section taught by an instructor unaffiliated with this research. Moreover, all of the ALVIS sessions consisted of students who came from an independent instructor's lecture section. Thus, we believe that, on balance, we were able to obtain a set of observations that were relatively free of the potential bias brought about by having an instructor observe his own students.

Second, we used *videotaping* as a primary field technique in all sessions. We videotaped the algorithm and visualization development activities of selected pairs of participants in the Art Supply sessions. In the ALVIS sessions, we instead used specialized screen recording software to produce audio and video recordings of all students' coding activities. In both the Art Supply and ALVIS sessions, we videotaped all of the visualization presentation sessions. We performed various post-hoc analyses of all recordings in order to identify patterns of activity and interesting events. To gain further insight, we elected to transcribe some of the recordings that included episodes we deemed interesting.

Third, we used *artifact collection* as a secondary field technique in all sessions. We collected all of the code and homemade visualizations developed by students in the Studio Experience sessions. We subsequently evaluated students' code to determine its semantic correctness, and we subsequently analyzed students' visualizations with respect to their use of stories and visual elements.

Fourth, we used *interviewing* as a secondary field technique in the Art Supply sessions. In order to pursue themes that emerged from our observations of these sessions, we audiotaped, and subsequently transcribed, brief interviews with selected students and teaching assistants.

Finally, we used *questionnaires* as a secondary field technique in all sessions. We administered a written exit questionnaire containing open-ended questions that asked students to reflect on how they approached the activities they undertook in Studio Experiences, and what they thought of them.

5. OBSERVATIONS

In 88% of the student pairs whose coding and visualization construction activities we considered (n = 19), we observed a clear division between (a) the task of coding the algorithm in pseudocode, and (b) the task of developing an accompanying visualization and story. This meant that students engaged in three primary activities in the Studio Experiences we observed:

- algorithm development,
- visualization and story development, and

- visualization presentation and discussion.

In the subsections that follow, we organize our observations around these three primary activities. Within each subsection, we have three aims: (a) to provide a high-level feel, grounded in our empirical record, for how students went about the activity; (b) to provide a high-level analysis of the products (code, visualizations, stories) of the activity; and (c) to highlight the ways in which students' activities and products differed between the Art Supply and ALVIS sessions.

5.1 Algorithm Development

In the first phase of the Studio Experiences we observed, student pairs used either a text editor (Art Supply sessions) or ALVIS (ALVIS sessions) to construct SALSA pseudocode solutions to the algorithm design problems assigned to them. Although, in the ALVIS sessions, the ALVIS tool itself also produced a generic visual representation at the level of variables, arrays, and array indices; (see Figure 3 above for an example), we observed that both Art Supply and ALVIS consistently deferred the task of developing a visual story until after the algorithm was initially developed. The rationale for separating algorithm development from story development is expressed in the following quote from a student participating in one of the Art Supply sessions:

> "Do we really have to have the story? I mean like the first thing we're gonna do is.,…we have to get the [algorithm] out of the way right? Well, I mean, like, if we don't finish the algorithm first, [we're in trouble]."

Thus, because the algorithm itself was generally perceived as the most important product of the Studio Experience, students tended to work it out prior to developing their visualizations and stories.

In the remainder of this section, we first focus on what students did as they coded their algorithms. We then consider the semantic correctness of their algorithms.

5.1.1 Coding Activities

Figure 5 presents a comparison of how students in the Art Supply and ALVIS sessions spent their time on algorithm development. Because of the large volume of video data we gathered, we chose to compute the percentages presented in this figure based on random video snapshots. In particular, we analyzed 21 two-minute snapshots taken from a random sample of 10 Art Supply pairs (roughly half of the Art Supply pairs we observed), and 15 two-minute snapshots taken from a random sample of 9 ALVIS pairs (again, roughly 50% of the ALVIS student pairs we observed). We chose this sampling strategy in order to cover a representative subset of participants' coding activities; in future work, we hope to increase the number of snapshots in our sample.

Three noteworthy observations can be made from the breakdowns presented in Figure 5. First, pairs in both sessions spent the majority of their time (50 to 60 percent) coding their solutions, and talking with each other about their solutions. This is not surprising, given that student pairs were tasked with collaboratively generating algorithms.

Second, it appears that student pairs in the Art Supply sessions relied much more heavily on discussions with others than did student pairs in the ALVIS sessions (20 percent vs. 8 percent). Our observations suggest that most of these conversations in the Art Supply sessions took place with teaching assistants, as students struggled to code their solutions. In addition to being

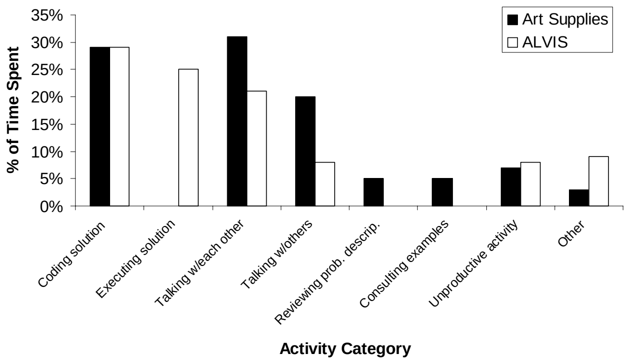


Figure 5. Comparative Breakdown of How Students Spent Their Time on Algorithm Development

unsure about correct SALSA pseudocode language syntax, Art Supply students commonly complained about the lack of execution feedback provided by the text editor they were using. This lack of feedback led to a low level of confidence that their code was correct. As one student put it in an interview,

> [I was] not very confident [that my algorithm was correct as written], because when you write it, it's hard to conceptualize in your mind what it does. But if you actually run it, it would be easier; you can see if it's right or wrong instantly.

As was the case for many students, this Art Supply student concluded that most challenging part of the coding exercise was "to see if [the code] was right."

It was not surprising, then, that a key role played by the teaching assistants who oversaw the Art Supply sessions was that of "code checker." During roughly 20 percent of their time, students would ask teaching assistants to step through their code for them and tell them whether it was correct.

A third observation to made from the data in Figure 5 is that both the Art Supply and ALVIS students spent similar proportions of time (seven to eight percent) engaged in activities that did not lead to visible coding progress. Through a follow-up analysis of students' unproductive time chunks, we found that 80% of the Art Supply students in our sample experienced one or more "stuck" periods of at least 2.5 minutes during which they made no visible progress in coding their solutions. In their effort to generate correct solutions, several of these Art Supply students simply gave up until they could summon a teaching assistant for help. In contrast, no student pairs in the ALVIS sessions had even one "stuck" period of at least 2.5 minutes. In fact, the longest "stuck" period we could find in our ALVIS session video samples was only 30 seconds. Indeed, we observed that, even when ALVIS pairs became "stuck," they were still able to engage in constructive activities, such as reviewing the execution of their code so far.

5.1.2 Code Products

Given the differences between the Art Supply and ALVIS students with respect to how they spent their algorithm development time, a key question arises: Did these *process* differences lead to notable *product* differences—that is, did one group of students develop better code than the other? Table 3 presents the results of analyzing the semantic correctness of the code written by a random sample of 50% of the student pairs who participated in the sessions we observed (n = 44 Art Supply code samples and 42 ALVIS code samples).

As the table indicates, ALVIS students committed roughly half as many semantic errors per algorithmic solution as the Art Supply students, which translates to a four-fold advantage if one averages the errors on a per-line basis. According to a non-parametric Kruskall-Wallis[1] test, the difference between the number of semantic errors per line in the Art Supply and ALVIS sessions is statistically significant (df = 1, H = 4.53, p = 0.03). Thus, as might have been predicted, the edit-time visual feedback provided by the ALVIS environment resulted not only in students' needing fewer interventions by a teaching assistant, but also in students' developing code that was significantly more correct.

5.2 Visualization and Story Development

In the second phase of the Studio sessions, students turned their attention to the visual representation of their algorithms, and to how that visual representation might depict the algorithm in terms of a story. One general observation was that students in the Art Supply sessions had less time to concentrate on visualization and story development. As we saw in the previous section, this most likely stemmed from the fact that the text editors they used for algorithm development made it difficult for them to determine if their coding efforts were on track.

In contrast, we observed many ALVIS pairs who wrote and verified the correctness of their algorithms well before the presentations started. This gave them ample time to explore the story-telling features of ALVIS (custom variable pictures, custom backgrounds, and speech bubbles; see Section 3.3), and to embellish their generic ALVIS visualizations as they saw fit.

A post hoc analysis of a sample of visualizations produced by a random sample of 20 Art Supply pairs and 20 ALVIS pairs suggests that students' visualizations varied along two key dimensions:

- *Level of personalization*—the extent to which students customized their representations with their own unique visual elements; and
- *Level of story content*—the extent to which students grounded their presentations in stories that portrayed the behavior of the underlying algorithms.

In the remainder of this section, we present analyses of our sample of visual presentations with respect to these dimensions.

5.2.1 Levels of Personalization

Students often took great pleasure in creating and personalizing visual presentations of their algorithmic solutions in a variety of ways:

- choosing pre-defined pictures for algorithm variables (in the case of the ALVIS sessions);
- creating custom pictures for algorithm variables;
- creating background doodles;

[1] Because our data points are ratios with varying denominators, they violate the assumptions of parametric statistics; we therefore use non-parametric Kruskal-Wallis tests in the statistical analyses presented in this paper.

Table 3. Average Number of Semantic Errors in the Algorithmic Solutions of Art Supply and ALVIS Students

	Lines of code per solution		Semantic errors per solution		Semantic errors per line of code	
Group	*M*	*SD*	*M*	*SD*	*M*	*SD*
Art Supply	19.07	7.77	0.77	0.95	0.04	0.01
ALVIS	21.55	6.56	0.36	0.78	0.01	0.01

- creating a more sophisticated background themes; and
- creating "cutouts" that can be animated (in the case of the Art Supply presentations).

In fact, 55 percent of the Art Supply pairs in our sample, and 65 percent of the ALVIS pairs in our sample, chose to personalize their visual presentations in one of the above ways.

As our analysis suggests, the ALVIS students appeared to perform more personalization than did the Art Supply students. We suspect that this greater level of personalization was a result of two main factors: (a) ALVIS students tended to have more time to invest in personalizing their visual presentations, because they tended to finish coding faster; and (b) the ALVIS tool made personalization easier than did art supplies, because specific personalization features were built in.

5.2.2 Levels of Story Content

In addition to personalizing their visual presentations, many students aimed to develop scenarios and stories to portray their algorithms. Our analysis suggests that the level of sophistication of students' story content varied widely. The following is a list of what we regard as six progressively more sophisticated levels of story content that were employed by student presentations:

1. *no story at all* (i.e., purely geometric representations);
2. *scenario not grounded in algorithm behavio*r (i.e., a background scenario that is unrelated to the algorithm);
3. *scenario grounded in algorithm behavior* (i.e., a background scenario that serves as an analogy for the algorithm's behavior);
4. *character dialog grounded in algorithm behavior* (i.e., variables and data structures talk to each other so as to emphasize how the algorithm works);
5. *plot grounded in algorithm behavior* (i.e., the visualization tells a story that serves as an analogy for the algorithm's behavior); and
6. *plot and character dialog grounded in algorithm behavior* (i.e., same as 5, except that the plot unfolds through character dialog between variables and data structures).

Figure 6 presents a comparison of the level of story content we observed in the Art Supply and ALVIS presentations. Overall, it appears that, although a substantial proportion (42 to 50 percent) of both the Art Supply and ALVIS pairs failed to construct any kind of story at all, the ALVIS presentations with story content tended to be more sophisticated than those Art Supply

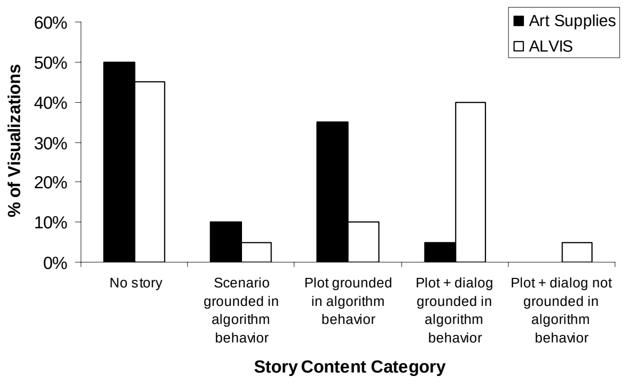


Figure 6. Comparison of Visualization Story Content

presentations that had story content. Indeed, 38 percent of the ALVIS presentations in our sample achieved the highest level of story content on our scale, whereas only 10 percent of the Art Supply presentations achieved that level.

5.3 Visualization Presentation and Discussion

In the final phase of the Studio sessions, which was allotted 45 minutes, student pairs presented their visualizations to the class for feedback and discussion. The 29 Art Supply presentations we observed lasted an average of four minutes and 29 seconds each (*sd* = 2:23). The 19 ALVIS presentations we observed were slightly shorter: four minutes on average (*sd* = 2:36).

A lab instructor (either the first author or one of the teaching assistants) facilitated each of the presentation sessions. To that end, he or she followed this four-step protocol, which served to give the presentation sessions a similar structure:

1. Call on a student group to come up to the overhead projector (in the Art Supply sessions) or the computer connected to the LCD projector (in the ALVIS sessions) to give their presentation.
2. Ask the student group to explain the algorithm design problem that they solved.
3. Ask the student group to provide a "play by play" as they walk through each of their two visualizations;
4. Ask the student group to perform Big-O analyses of their alternative algorithms, and to justify their analyses.

Even though the presentation sessions were specifically designed to promote audience participation and discussion, we discovered that student presenters contributed by far the most to these sessions. In the remainder of this section, we first present a high-level analysis of instructor and audience contributions. We then turn to an analysis of the specific content of the presentations: what was presented and talked about.

5.3.1 Audience and Instructor Contributions

Table 4 presents a high-level analysis of the verbal contributions made by student audience members and the instructor in the Art Supply and ALVIS sessions. (For the purposes of this analysis, a "contribution" is defined as at least one sentence of verbal dialog.) As the table indicates, neither the student audience nor the instructor contributed significantly to the presentation discussions, with average contributions ranging from only 0.01 to 1.69 per minute. In addition, note that at least some humor tended to be present in the presentations, with the number of laughs per minute ranging from 0.04 to 0.32. [In this analysis, a "laugh" is defined as an instance of (simultaneous) laughter by one or more audience members.]

Table 4. Average Number of Contributions from Student Audience and Instructor in Art Supply and ALVIS Sessions

	# Student audience contributions per minute		# Instructor contributions per minute		# Laughs per minute	
Group	*M*	*SD*	*M*	*SD*	*M*	*SD*
Art Supply	0.01	0.06	1.69	2.80	*0.04*	*0.12*
ALVIS	0.30	0.50	1.07	0.70	0.32	0.51

Even though these rates of student audience contributions, instructor contributions, and audience laughter are relatively low overall, it is notable that there exist significant differences between the Art Supply and ALVIS sessions with respect to two of these three measures. According to non-parametric Kruskal-Wallis tests, while no statistically significant difference in instructor contributions per minute exists between the two groups (df = 1, H = 0.06 , p = 0.81), the ALVIS presentations had significantly more student audience contributions per minute (df = 1, H = 11.28 , p = 0.001), and significantly more laughs per minute (df = 1, H = 7.17, p = 0.007).

Our observations suggest three potential reasons for the increased rate of student audience contributions in the ALVIS sessions. First, we observed that the instructors who facilitated the ALVIS sessions did a better job of eliciting student participation. Whereas in the Art Supply sessions, the instructors rarely directed their questions to anyone but the student presenters, instructors did so at least occasionally in the ALVIS sessions. Consider, for example, the following exchange, in which the instructor (I) encourages the audience (A) to help the presenters (P1) with a Big-O analysis of their partition algorithm:

I:	What is the big-O efficiency of this algorithm?
P1:	It does make just enough comparisons so it makes, what, n comparisons for every size n of your list with the pivot. It doesn't have to run through the entire list more than once.
…	
I:	So its efficiency then, is what? O of?
[3 sec pause]	
I:	Someone help him out.
A::	N

A second possible reason for the higher rate of student audience contributions in the ALVIS sessions relates to the differing levels of error visibility in the Art Supply and ALVIS presentations. In the Art Supply sessions, students tended to present their algorithms at a relatively high level; indeed, because the presenters themselves were executing their algorithms, they could gloss over, or even avoid, specific execution details through "hand waving" maneuvers. In contrast, in the ALVIS sessions, the ALVIS environment executed presenters' algorithms. As a result, presenters could not gloss over execution details. In many cases, semantic errors, of which the presenters were already aware, became plainly visible. Thus, in the ALVIS sessions, the presentation sessions served as forums for collaboratively fixing semantic errors in presenters' algorithms, leading to increased opportunities for audience participation.

We observed a third possible reason for the higher rate of audience participation in the ALVIS sessions: Students in the ALVIS sessions seemed to finish their coding activities more quickly than did students in the Art Supply sessions. Thus, when it came time for the presentations, we observed that nearly all pairs of students in the ALVIS sessions had at least one of their two algorithms working, enabling them to give their full attention to the presentation sessions. In contrast, we observed that many students in the Art Supply sessions failed to finish coding their algorithms in the two hours they were allotted. This meant that they continued to work on their presentations *while* other students were presenting—an endeavor that effectively prevented them from participating in other students' presentations.

An observation mentioned in the previous paragraph—that ALVIS students finished coding their algorithms more quickly—might also explain why there was more laughter in the ALVIS sessions. Because students in the ALVIS sessions finished coding their algorithms more quickly, they had more time to develop the storylines and visual elements of their presentations. This meant that they could put more effort into crafting humorous storylines, visual characters, and dialog (in the form of speech bubbles). Indeed, the speech bubble animation supported by ALVIS seemed to evoke laughter almost every time it was used in a presentation.

5.3.2 Presentation Content

Having presented a high-level analysis of *who* contributed to the presentations, we now consider *what* was actually talked about. As discussed above, contributions by student audience members were relatively infrequent in both the Art Supply and ALVIS sessions; most of the talk that took place in the presentations was either presenter monologue, or presenter-instructor dialog. To explore this talk in greater detail, we performed a content analysis of the transcripts of 20 randomly selected presentations—10 from the Art Supply sessions, and 10 from the ALVIS sessions. We divided the transcripts into segments, which we defined as single sentences or utterances. Two independent coders then classified 20 percent of the segments into one of the mutually-exclusive content categories presented in Table 5, and achieved 95% agreement. Having established sufficient reliability, we had single analyst code the remaining 80% of the segments.

Figure 7 presents a comparison of the content composition of the Art Supply and ALVIS presentations. As the figure suggests, there were remarkable similarities between the Art Supply and ALVIS presentations. In line with the learning objectives of the presentations sessions, Algorithm Behavior and Algorithm Eficiency were the dominant topics in both the Art Supply and ALVIS sessions. At first glance, two differences stand out:

- Whereas 20 percent of talk in the ALVIS presentations focused on Errors and Error fixes, less than 1 percent of the talk in the Art Supply presentations focused on these topics.
- Algorithm Behavior appears to have been talked about much more frequently in the Art Supply presentations.

Table 5. Presentation Topic Categories

CODE	DESCRIPTION
Problem	Talk focused on the problem being solved by the presenters' algorithms
Algorithm Behavior	Talk focused on the step-by-step behavior of the presenters' algorithms
Algorithm Efficiency	Talk focused on the efficiency of the presenters' algorithms
Story & Visual Elements	Talk focused on the story and visual elements that presenters are using to describe their algorithm.
Errors	Talk focused on errors in the presenters' code.
Error Fixes	Talk focused on fixing errors in the presenters' code
Tool	Talk focused on aspects of the ALVIS environment or the art supply materials
On Task	Other talk that relates to the presentation task being performed
Off Task	Any other talk that does not relate to the presentation task being performed.

Upon closer inspection, it turns out that these two differences are closely related, and provide substantial insight into the pedagogical tradeoffs between art supplies and a computer-based tool like ALVIS within the context of interactive presentations. In the Art Supply presentations, we observed that talk about algorithm behavior tended to be at a higher level of abstraction. Student presenters typically presented their pseudocode, and then "handwaved" their way through a sample execution of that code. For example, consider the following Art Supply presentation segment , which illustrates the handwaving that took place:

P: For ours we had to allow the input of integers between 1 and 100. And we created an array however large, and put them in ascending order from highest to lowest. On one of them we were supposed to use 2 arrays. So we set up our first array. Our while loop scans through one time and it goes to the highest number. After it's done, it moves down to the second array and it moves the array iterator for the second array to the right. The next time it scans and finds the highest number it'll synchronize.

Notice that the presenter covered the algorithm at such a high level of abstraction that it was difficult for the audience to follow the algorithm, let alone detect errors.

In stark contrast, owing to their basis in "live" code, ALVIS presentations tended to be much more grounded in the detailed execution of the algorithm. This allowed the discussion to focus more intently on the specific details of the algorithms being presented, as the following segment of one ALVIS group's presentation of the partition algorithm illustrates (P1 and P2 are the presenters, A is an audience member, and I is the lab instructor):

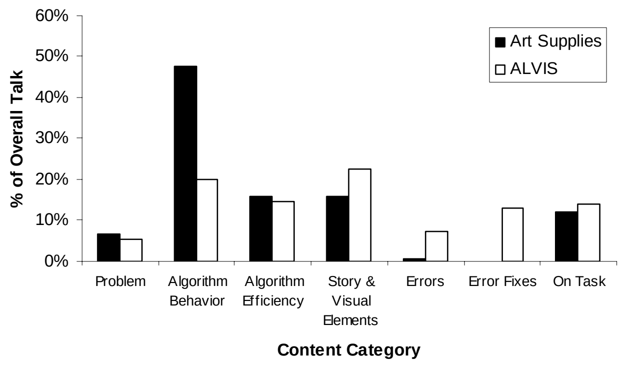


Figure 7. Comparison of Presentation Session Content

P2:	So when get here to the while loop, it's just going to set current to that position, instead of swapping.
…	
I:	Is there an error in your program. [Noticing the visual display]
…	
P2:	Yeah, there is a problem.
I:	Oh, you can't deal with duplicates.
P1:	Yeah, we forgot to put that in.
P2:	We just didn't think the odds of duplicates were too high.
I:	Well, it does happen from time to time.
A:	There's two 97's there. [Noticing the duplicates in the visual display)
P1:	It doesn't matter in this case, because if the pivot is matched by another, then there's trouble because it doesn't know what to do when it equals instead of greater or less than.
I:	Ah, well that's easy to fix. How would you fix that?
P1:	You put in an else at the very end that says else equals to
I:	Or you could change the if here to less than or equal to, and then just add one else.

In sum, it appears, upon closer inspection, that algorithm behavior was talked about equally frequently in the Art Supply and ALVIS sessions—around 60 percent of the time. The key difference was in the level of detail. In the Art Supply presentations, presenters tended to talk about algorithms at a high level of abstraction, glossing over the (potentially erroneous) low level details of their algorithms. In contrast, presenters in the ALVIS sessions were forced to embrace the errors in their algorithms. As a result, presenters tended to stimulate discussion about the specific errors in their algorithms, and about how to go about fixing them.

6. DISCUSSION

Given the observations and analyses presented in the previous section, what insights have we gained into our three original research questions, and what research remains for the future? In this section, we discuss our results vis-à-vis those three questions, and propose directions for future research.

6.1 Does the Construction of Personalized AVs Aid Algorithm Understanding?

We have presented a diverse set of observations on how novices go about the process of constructing algorithmic solutions and accompanying visual representations with differing tools. We observed that the construction of visualizations engaged students actively in the course, and that students appeared to derive pleasure from constructing story content. In addition, we have analyzed the code and visualizations that resulted from that process. However, aside from the result that students who used ALVIS to construct their algorithmic solutions had significantly fewer errors than students who used art supplies, we failed to gather any concrete evidence that students actually *learned* through the process of algorithm construction. Thus, we have to question whether the time students spent on this endeavor actually benefited their understanding of the algorithm. Perhaps it did not benefit their understanding per se, but rather increased the general level of engagement and enjoyment of students in the course? In future research, we would like to use an attitude questionnaire in order to see if this is the case. In addition, we would like to gather further quantitative evidence that students actually *learned* through the algorithm construction process.

6.2 Does the Presentation of Personalized AVs Lead to Educationally Beneficial Conversations?

We gathered a rich set of data on *who* contributed in the presentation discussions, and *what* was talked about. In the transcripts we analyzed, we found many examples of educationally beneficial conversations, like the one presented in the Section 5.3.2. Because they were firmly grounded in students' personal representations of algorithms, such conversations appeared to enable student presenters and instructors to bridge the gap in their understandings of algorithms. However, future research will need to take care to link the conversations that take place in presentation sessions to concrete learning outcomes.

In addition, although we observed minimal audience participation in the presentation sessions, we cannot say at this point that these sessions benefited anyone except the presenters themselves. Indeed, several questions for future research remain:

- Did students in the audience actually learn each other's algorithms through the presentations?
- Did constructing their own algorithms somehow "prime" them to learn other algorithms?
- Does a student have to make a contribution in order to learn?

6.3 What Form of AV Technology Best Supports the Above Processes?

In presenting our observations and analyses, we took great care to note process and product differences between the Art Supply and ALVIS students. In fact, based on our results, we were able to identify three key differences:

- The ALVIS tool appeared to promote a faster coding process, with fewer "stuck" periods, and less reliance on expert help.
- The ALVIS tool appeared to promote the creation of visualizations with richer storylines and more elaborate visual elements.
- The ALVIS tool appeared to promote conversations with a sharper focus on the specific details of algorithm behavior, leading to the collaborative identification and repair of semantic errors.

Given these differences, the ALVIS tool appears to have clear advantages over simple art supplies within the context of the CS1 Studio Experiences explored here. Nonetheless, we found that ALVIS is far from optimal. This is especially true of its coding interface, which caused some students considerable difficulties; it will need to refined in future research.

7. SUMMARY AND IMPLICATIONS

In this paper, we have presented an observational study of CS1 *studio experiences*—novel pedagogical activities in which students construct algorithmic solutions to algorithm design problems, personalize those solutions with visual elements and storylines, and finally present their solutions to their peers and instructor for feedback and discussion. Through a diversity of ethnographic field techniques, we have gathered a rich set of data that has allowed us to gain preliminary insights into both the potential pedagogical value of these activities, and the kind of technology that can best support the activities.

Based on the results and analyses presented here, we can make the following specific recommendations to computer science educators interested in exploring the use of studio experiences in their own courses:

1. *Make sure that novice students have consistent access to knowledgeable experts during the coding process.* We were surprised by the difficulties that students in our study had as they developed algorithms in pseudocode. It is not enough simply to sit pairs of novice students in front of a computer and ask them to write pseudocode Even when a novice programming environment like ALVIS is enlisted, students will still need consistent access to skilled teaching assistants.
2. *Make sure to give students enough time to develop their algorithmic solutions.* As we found in our Art Supply sessions, two hours may not have been enough for novice students to fully develop two alternative algorithmic solutions and accompanying visual presentations. Our observations suggest that, if students do not have enough time to complete the coding process prior to their presentations, the level of participation in the presentations will suffer.
3. *Make sure to have skilled facilitators for the presentation sessions.* Our study found that novice students will be reluctant to participate in discussions about algorithms unless they are specifically drawn in to them. While the humor associated with storytelling and visual presentations can help in that regard, we believe that the pedagogical value of presentation discussions, as well as the level of participation in such discussions, hinges on skilled facilitators who ask the "right" questions. The "right" questions, based on our findings, are those that use students' own representations as a basis for posing questions about algorithmic behavior and efficiency.

The ultimate aim of the Studio Experiences explored here is to engage students in a process of *reflective practice* described by Schon [21]: to get students to think about the process of coding an algorithm, to share their insights with others, and ultimately to become participants in a vibrant community of "schooled algorithmaticians." Our observations suggest that a studio-based pedagogy is a step in the right direction, but will require more refinement in order to achieve our pedagogical goals.

8. ACKNOWLEDGMENTS

We are grateful to our informants: the students and instructors who agreed to allow us conduct this study within two offerings of CptS 121 at Washington State University. Laura Girardeau transcribed and helped to analyze the visualization presentation sessions considered in this paper. This research is funded by the National Science Foundation under grant no. 0406485. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF.

9. REFERENCES

[1] Ben-Bassat Levy, R., Ben-Ari, M. and Uronen, P. The Jeliot 2000 program animation system. *Computers & Education*, *40*, *1* (2003), 1-15.

[2] Boyer, E.L. and Mitgang, L.D. *Building Community: A New Future for Architecture Education and Practice*. The Carnegie Foundation for the Advancement of Teaching, Princeton, NJ, 1996.

[3] Byrne, M.D., Catrambone, R. and Stasko, J.T. Evaluating animations as student aids in learning computer algorithms. *Computers & Education*, *33*, *4* (1999), 253-278.

[4] Carlisle, M., Wilson, T., Humphrieis, J. and Hadfield, S. RAPTOR: A visual programming environment for teaching algorithmic problem solving. In *Proc. ACM SIGCSE 2005 Symposium*, ACM Press, New York, 2005.

[5] Dann, W., Cooper, S. and Pausch, R. Making the connection: Programming with animated small world. In *Proc. ITiCSE 2000*, ACM Press, New York, 2000, 41-44.

[6] Doidge, C., Sara, R., Parnell, R. and Parsons, M. *Crit - An Architectural Student's Handbook*. Architectural Press, Boston, 2000.

[7] Gurka, J.S. *Pedagogic Aspects of Algorithm Animation*. Unpublished Ph.D. Dissertation, Computer Science, University of Colorado, 1996.

[8] Hanly, J.R. and Koffman, E.B. *Problem Solving and Program Design in C*. Addison Wesley, Boston, 2004.

[9] Hübscher-Younger, T. and Narayanan, N.H. Constructive and collaborative learning of algorithms. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ACM Press, New York, 2003, 6-10.

[10] Hübscher-Younger, T. and Narayanan, N.H. Dancing hamsters and marble statues: characterizing student visualizations of algorithms. In *Proceedings of the 2003 ACM symposium on Software visualization*, ACM Press, New York, 2003, 95-104.

[11] Hundhausen, C.D. Integrating Algorithm Visualization Technology into an Undergraduate Algorithms Course: Ethnographic Studies of a Social Constructivist Approach. *Computers & Education*, *39*, *3* (2002), 237-260.

[12] Hundhausen, C.D. and Brown, J.L. What You See Is What You Code: A radically dynamic algorithm visualization development model for novice learners. In *Proc. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society Press, Los Alamitos, In press.

[13] Hundhausen, C.D. and Douglas, S.A. Low fidelity algorithm visualization. *Journal of Visual Languages and Computing*, *13*, *5* (2002), 449-470.

[14] Hundhausen, C.D., Douglas, S.A. and Stasko, J.T. A meta-study of software visualization effectiveness. *Journal of Visual Languages and Computing*, *13*, *3* (2002), 259-290.

[15] Joint Task Force on Computing Curricula. Computing curricula 2001. *Journal on Educational Resources in Computing (JERIC)*, *1*, *3es* (2001).

[16] Lawrence, A.W., Badre, A.N. and Stasko, J.T. Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Los Alamitos, CA, 1994, 48-54.

[17] Naps, T. Algorithm visualization in computer science laboratories. In *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education*, ACM Press, New York, 1990, 105-110.

[18] Naps, T.L., Roessling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., Mchally, M., Rodger, S. and Valazquez-Iturbide, J.A. ITiCSE 2002 working group report: Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, *35*, *2* (2003), 131-152.

[19] Roschelle, J., Designing for conversations. in *AAAI Symposium on Knowledge-Based Environments for Learning and Teaching*, (Stanford, CA, 1990).

[20] Schneider, G.M. and Gersting, J.L. *An Invitation to Computer Science*. Thomson Course Technology, Boston, 2004.

[21] Schon, D. *The reflective practitioner: How professionals think in action*. Basic Books, New York, 1983.

[22] Stasko, J., Badre, A. and Lewis, C. Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, ACM Press, New York, 1993, 61-66.

[23] Stasko, J.T. Using student-built animations as learning aids. In *Proceedings of the ACM Technical Symposium on Computer Science Education*, ACM Press, New York, 1997, 25-29.

[24] Stasko, J.T. and Hundhausen, C.D. Algorithm visualization. In Fincher, S. and Petre, M. (eds.), *Computer Science Education Research*, Taylor & Francis, Lisse, The Netherlands, 2004, 199-228.