# Exploring Studio-Based Instructional Models for Computing Education

Christopher D. Hundhausen
School of Elec. Eng. and Comp. Sci.
Washington State University
Pullman, WA 99164-2752
+1 509-335-4590
hundhaus@wsu.edu

N. Hari Narayanan
Computer Sci. & Soft. Eng. Dept.
Auburn University
Auburn, AL 36849-5347
+1 334-844-6312
naraynh@auburn.edu

Martha E. Crosby
Dept. of Info. and Computer Sciences
University of Hawaii at Manoa
Honolulu, HI 96822
+1 808-956-3500
crosby@hawaii.edu

## ABSTRACT

While the demand for college graduates with computing skills continues to rise, such skills no longer equate to mere programming skills. Modern day computing jobs demand design, communication, and collaborative work skills as well. Since traditional instructional methods in computing education tend to focus on programming skills, we believe that a fundamental rethinking of computing education is in order. We are exploring a new "studio-based" pedagogy that actively engages undergraduate students in collaborative, design-oriented learning. Adapted from architectural education, the studio-based instructional model emphasizes learning activities in which students (a) construct personalized solutions to assigned computing problems, and (b) present solutions to their instructors and peers for feedback and discussion within the context of "design crits." We describe and motivate the studio-based approach, review previous efforts to apply it to computer science education, and propose an agenda for multi-institutional research into the design and impact of studio-based instructional models. We invite educators to participate in a community of research and practice to advance studio-based learning in computing education.

## Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: *Computer science education.*

## General Terms

Design, Experimentation, Human Factors.

## Keywords

Studio-based learning and instruction, Design crits, CS Ed research, pre-CS1, CS1, CS2, CS3, Student-constructed artifacts.

## 1. INTRODUCTION

The field of computing education is ripe for change and innovation, for a variety of reasons. For one, it is experiencing a significant drop in incoming majors. According to the latest Taulbee Survey, the number of new computer science majors has dropped 39 percent over the past two years [21]. At the same time, demand for college graduates with computing skills is on the rise. Moreover, computing-related employment in the U.S. is experiencing a shift in which rote programming jobs are being outsourced offshore, while new jobs in the IT field emphasize creativity, design, problem solving and interdisciplinary collaborative work. All of these endeavors require communication and teamwork skills as much as programming skills.

We believe that, with few exceptions, university computer science departments have yet to adapt to meet the challenges associated with these changes. Instruction continues to be conducted in the traditions established when computer science emerged from mathematics decades ago. Many courses, especially early core courses on computing fundamentals, data structures and algorithms, focus on teaching programming instead of communication, problem solving and design. Anecdotal evidence suggests that failure and drop out rates in these early courses are significant, as is the failure of students to transfer and apply knowledge and skills from one course to another.

Recent reports on changes in the field of computing and computing education (e.g., [16]) suggest at least two reasons for these failures:

(a) *Computing is too often associated with programming*. Computing degrees are seen as predominantly leading to programming careers, with the associated negative stereotype of a socially challenged person hunched over a monitor all day. This negative stereotype may dissuade prospective students from entering computer science programs.

(b) *Introductory computing education is too closely tied to programming languages*. Many educators have bemoaned the way in which the core courses of undergraduate computing programs (CS1, CS2 and CS3 in the ACM standard curriculum; see [12]) are taught. While these courses address concepts in problem solving, algorithms and data structures, they are typically wrapped in the currently fashionable programming language (e.g., C++, Java). Thus, students are typically exposed to the fundamental concepts of the field while still grappling with syntax and other idiosyncrasies of a programming language. In students' struggle to master a programming language, fundamental concepts of computing may be obscured, and they may become discouraged to enter the field.

These observations motivate a rethinking of undergraduate computing education. To that end, we explore the potential for *studio-based* instructional methods to focus computing instruction more intently on design, communication, and problem solving. Adapted from architectural education, the studio-based instructional model emphasizes learning activities in which students (a) construct personalized solutions to assigned computing problems, and (b) present solutions to their instructors and peers for feedback and discussion within the context of "design crits" (design critiques). Thus, students participate in studio-based courses in ways typically reserved for instructors. Rather than emphasizing individual problem solving in isolation, studio-based courses regard group discussions mediated by student-constructed solutions as educationally valuable.

In this paper, we introduce studio-based instruction as a pedagogical approach for computing education, and we describe a multi-institutional research project in which we aim to refine, empirically evaluate, and build a community of educators around the approach. We begin, in Section 2, by placing our research within the context of related computing education research and our own prior research into the use of studio-based approaches in computing education. Section 3 identifies the key features of the studio-based pedagogy, while Section 4 describes how the approach might be applied in core computing courses. Section 5 outlines a multi-institutional research project to refine and evaluate the approach. Finally, Section 6 discusses the current status of the research, and issues a call for collaborators.

## 2. RELATED AND BACKGROUND WORK

Over the past two decades, researchers have explored a variety of approaches to improving computing education. These have included innovations both in pedagogy (e.g., [1, 20]), and in the programming languages and environments used to introduce computing to novices (e.g., [4, 5, 14]).

Embracing the concern that introductory computing education is too closely tied to programming languages, some of these efforts have aimed to decouple computing concepts from programming languages. One promising line of research has explored programming and visualization environments that enable learners to construct, view, and interact with concrete visual representations of computing processes (e.g., [3, 10]). These novice programming and visualization environments proceed from the idea that learning is a process in which one constructs one's own understandings through active engagement with one's environment, especially through constructing and exploring one's own artifacts.

While several previous empirical studies have demonstrated the value of novice programming and visualization environments in promoting both positive learning outcomes (e.g., [11, 15]) and increased retention (e.g., [15, 19]), little research has focused on explicating the particulars of innovative pedagogical approaches that amplify the benefits of such programming and visualization environments.

In one of the few lines of research to explore a novel pedagogical approach integrating programming and visualization into computing courses, we have developed a studio-based instructional technique in which students construct and discuss their own visual solutions to assigned problems. We found that when students construct their own visual solutions, discussions mediated by these learner-constructed solutions are educationally beneficial. Such discussions lead not only to improved learning outcomes, but also to a stronger sense of community, improved critical thinking skills, and prevention of premature convergence to incomplete knowledge [6-9]. We believe these benefits accrue because students' motivation and attention increase when they discuss artifacts of their own creation, and because such artifacts serve as powerful *mediational resources* [17] that bridge the gap between expert and novice understandings.

The studio-based approach explored in our research fills a gap in computing education research literature by developing and empirically evaluating a pedagogical approach in which students use programming and visualization environments to construct and present their own representations of computing processes, as well as personalized solutions to computing problems. Unlike the efforts cited above, our approach emphasizes the educational value of students' not only constructing personalized solutions, but also presenting those solutions to the rest of the class for peer review and discussion.

## 3. KEY FEATURES OF THE STUDIO-BASED PEDAGOGY

While the details of their implementation can differ widely, two key learning activities define the studio-based approach:

(a) *representation construction*—students construct their own visual and verbal representations of the computing concepts, algorithms, or processes under study; and

(b) *representation presentation and discussion*—in "design crits" (design critiques), students present their representations for discussion and feedback from their peers as well as the instructor.

While activity (a) is often part of more traditional "open" or "closed" laboratory approaches [13] , activity (b) sets the studio-based approach apart from these more traditional approaches. In particular, in a studio-based course, several of the regularly scheduled laboratory periods are dedicated to "design crits" in which students present their work to the other students and instructor, and discuss, critique, and evaluate each other's work.

Taken together, these representation construction, presentation, and discussion activities form a unique approach to learning computing. In particular, they provide a "way in" to computing as a *design science*, which encompasses not only conceptual and procedural knowledge, but also an ability to apply the vernacular (e.g., Big-O notation), representations (e.g., data structure diagrams), and tacit heuristics (e.g., time-space tradeoffs) of the computing community of practice to design solutions to problems.

The studio-based approach has three key features that make it particularly well suited to the varying needs of CS educators:

1. *Scalability*. The approach scales well to different computing courses and class sizes. Any computing course includes concepts and processes that learners can represent, present, and discuss. Moreover, these activities can be implemented in weekly studios that replace labs. Ideally, studio work would be done in sections of 20 students or less, to encourage collaboration, personal attention from a studio instructor, and participation in "design crits." Large classes

(e.g., CS1) can accommodate the approach by transforming smaller lab sections into studios. Alternately, studio sessions can be implemented in the classroom, while the bulk of the work is conducted outside the class asynchronously through an on-line collaborative learning system.

2. *Adaptability*. The approach can be easily adapted to meet the local needs of specific institutions and instructors. Rather than replacing lectures, the core activities of the approach—representation construction, presentation, and discussion—are intended to complement the lectures. Instructors can choose to implement a weekly studio if time allows, or to implement studios that cover only a subset of the topics in a course. Moreover, the specific ways in which instructors facilitate (a) peer collaboration within representation construction activities, and (b) discussions mediated by student-constructed representations, can vary according to the needs of the instructor. For instance, an instructor might require students to develop representations individually or in pairs, opt to give presenters written feedback, or elect to keep all feedback informal and verbal.
3. *Technology independence*. While implementations of the studio-based approach will clearly benefit from visualization or concept representation software, the approach is designed to be completely independent of any specific technology. Instructors can, in fact, choose to have students develop their representations using a variety of technologies, ranging from art supplies (as in [9]) to algorithm visualization technology that works with standard programming languages (e.g., [2, 18]), to standalone program development and visualization technology (e.g., [3, 4, 10]), to online learning technology that allows students to share and critique artifacts independently of how the artifacts are produced (e.g., [7]). Such technology independence gives instructors broad latitude in implementing the approach, enabling them to make use of the specific technologies with which they are familiar and comfortable.

## 4. EXAMPLE COURSES

As argued above, the studio-based approach lends itself to a broad range of computing courses. For example, two upper-division courses to which the approach is particularly well suited are (a) a course on human-computer interaction that emphasizes user interface design, analysis, and evaluation skills, and (b) a course on software engineering that stresses high-level software architecture design and software development practices. Below, we highlight the pedagogical features of the approach by sketching out how the approach could be implemented in four core computing courses: pre-CS1, CS1, CS2, and CS3. We are presently implementing and evaluating these four courses in the studio-based format at our three universities.

### 4.1 Pre-CS1

At the pre-CS1 level, we believe a key objective should be to make problem solving through programming fun, inspiring, and personally meaningful, in order to capture the interest of students. To that end, we propose a pre-CS1 course that consists of two weekly lectures that introduce the week's programming concepts (e.g., conditionals, loops), together with a weekly studio session in which students either (a) use a novice programming environment (such as Alice [4]) to work through a series of exercises that give them practice with the concepts, or (b) present to the class for feedback and discussion their progress toward each of a series of programming projects in which they develop, program, and narrate progressively more complex stories and scenarios. We believe that the fact that student projects are actually stories of students' own choosing would motivate the students not only as program authors, but also as audience members. Moreover, through presenting and discussing their project milestones, students would gain valuable communication skills that are important in the computing profession—most notably, an ability to present, rationalize, debug, and critique programmed solutions to design problems.

### 4.2 CS1

At the CS1 level, we anticipate that students will already have an interest in computers as tools for problem solving. Our key objective in this course would be to solidify that interest by having students complete, share, and discuss a series of progressively more challenging programming projects that solve personally-meaningful problems. As in the pre-CS1 course, we could structure the CS1 course around two weekly lectures, a weekly studio session, and a set of projects to be worked on and presented in the studio. In addition, students could be asked to use an algorithm visualization environment such as ALVIS Live! [10] to build illustrative visual representations of their solutions, which they would then present to the class for feedback and discussion during "design crits" scheduled every other week. As in the previous course, such "design crits" would serve to grow and refine key communication and analysis skills, including the ability to present, rationalize, debug, and critique solution designs (algorithms) and their implementations (programs).

### 4.3 CS2

At the CS2 level, students will have obtained a background in structured programming and top-down design. Our main goal is to strengthen students' programming and design skills by (a) introducing more advanced data structures (e.g., red-black trees) and problem solving techniques (e.g., graph representations); (b) challenging students to create and present solutions to progressively more difficult design problems; and (c) requiring collaborative teamwork for some of the exercises. The CS2 class that we propose would maintain the same lecture-studio structure used in CS1, and continue the use of a standard programming language. In order to complete studio exercises, students would work together in teams of two or three, with each student being responsible for constructing a separate module of a complex software design problem. Because they would be required to construct multi-part solutions collaboratively, each student would need a clear understanding of (a) the overall task, and (b) how each student's module fits into the design of the overall solution. Each student would also be required to construct an explanatory representation or visualization of his or her module, and to present it to other team members. In so doing, students would negotiate with their teammates their understanding of the entire design task. The team would then present their solution to the class for feedback during "design crits." Students' negotiation process in smaller groups, coupled with their class presentations, would reinforce their understandings of complex problem solutions and develop their communication and analysis skills.

## 4.4 CS3

At the CS3 level, we expect that students will work in a more independent and self-directed fashion than they do in prior courses. Our key objective in this course would be not only to provide students with a solid background in advanced algorithmic problem solving and analytic techniques, but also to challenge students to understand, develop, and mathematically analyze efficient algorithmic solutions to interesting problems. We propose a series of homework assignments that supplement traditional lectures. For each assignment, the instructor would explain a meaningful problem, but initially omit an explanation of its algorithmic solution. Students would have a week to consider the problem and research solutions individually, and to develop expository representations that explain both the mechanics and mathematical analyses of their solution algorithms to a fellow student. Students' representations might be in the form of a story, an animation, or another kind of visualization mixing text, equations and graphics. Students would then share their representations with the class. In the second week of the assignment, students would discuss each other's representations, critique them, and rate them on a multi-dimensional scale provided by the instructor. The assignment would conclude with the instructor discussing the algorithms, student representations and ratings in class. This implementation of the studio approach stresses understanding existing algorithms through a process of self-learning and collaborative critiquing, coupled with individual construction of solutions, analyses and explanations.

## 5. RESEARCH PROGRAM

We have recently begun a two-year multi-institutional project at Washington State University (WSU), Auburn University (AU), and the University of Hawaii (UH) to refine, empirically evaluate, and build a community of computing educators around studio-based instructional approaches.

### 5.1 Project Activities

The project encompasses four main activities:

*Stage 1a: Implementation and evaluation of the approach at the local level.* We have begun systematic implementation and evaluation of the studio-based approach in consecutive courses at WSU (pre-CS1, CS1), the University of Hawaii (pre-CS1, CS1, CS2), and Auburn University (CS2, CS3) in fall 2007. At each university, the courses will be offered in alternating studio and traditional formats. This will facilitate replicated comparisons between (a) student performance in studio-based and traditional offerings of the same course in a given year, and (b) performance in subsequent classes of students who took previous computing courses in studio-based vs. traditional formats.

*Stage 1b: Dissemination and community building at the regional level.* Concurrent to Stage 1a, we plan to undertake outreach activities to recruit computing educators from regional institutions. To enable the community to communicate and collaborate regularly, we are building a web-based support system that facilitates on-line communication and provides a repository for studio-based curricular materials. This community of educators from Alabama, Hawaii, and Washington will meet at a regional workshop to be held in the summer of 2008.

*Stage 2a: Refinement, implementation and evaluation of the approach at the regional level.* During 2008-2009, we will refine the studio-based pedagogy based on results from our first year of implementation and evaluation, and feedback received at the regional workshop. In addition, we will expand the implementation and evaluation efforts of Stage 1a to regional institutions other than our own. Through videoconferences, the on-line community support system and site visits, we will provide these institutions with technical and logistical help for this effort.

*Stage 2b: Dissemination and community-building at the national level.* In 2008, we will embark on a national recruitment effort to attract more computing educators to the community. We will expand the web-based community support system to build and sustain this group. These activities will culminate in a national workshop, co-located with SIGCSE 2009, at which the results and experiences of those who have implemented and evaluated the studio-based approach in various computing courses will be reported, and input from participants regarding how to further refine the approach will be sought. The ultimate of this workshop will be to initiate a follow-on project to implement and evaluate the studio-based approach nationally and more widely across the undergraduate computing curriculum.

### 5.2 Evaluation of the Pedagogy

Our current and future evaluations of the pedagogy will address four key questions:

RQ1. Do students learn better in studio-based instruction than in traditional instruction?

RQ2. Are students able to better transfer and apply the knowledge learned through the studio-based approach in future courses?

RQ3. Are students more engaged, invested and motivated in studio-based classrooms?

RQ4. What are the long term impacts of this approach in terms of persistence in the undergraduate program and future plans for computing careers?

To address these questions, we are offering two or three consecutive courses from a set of four undergraduate computing courses (pre-CS1, CS1, CS2 and CS3) in alternating studio and traditional formats over the next two years at each participating university, and undertaking the following data collection, comparison and evaluation activities.

To address RQ1, we have developed a common pre- and post-test instrument, specific to each course, to assess the learning of key computing concepts and skills in each course. We will also use course-specific assignment and exam scores to track student progress in each consecutive course sequence. Collection of these data will allow us to systematically compare student learning in traditional and studio-based courses within and across institutions.

To address RQ2, we will compare the performance in subsequent courses of students who take previous courses in the traditional and studio-based formats. We will also compare failure and dropout rates in each course vis-à-vis current and previous exposure to studio-based and traditional formats.

To address RQ3, we will track the attitudes, engagement, motivation and future plans of students in the consecutive courses at each university through both pre and post surveys (at the start and end of each course in each semester), and interviews of selected students. These data should reveal immediate and long-term affective impact of traditional vs. studio-based instruction.

To address RQ4, we will track retention by looking at the number of students who take all courses in a sequence, and the number of students who subsequently remain in the computing program, at each of the participating institutions.

## 6. SUMMARY AND CALL FOR PARTICIPATION

In this paper, we have argued that traditional instructional methods in computer science tend to focus too intently on programming and individuals working in isolation. This not only runs the risk of turning students away from the field; it also inadequately prepares students for today's computing careers, which increasingly require skills in design, communication, problem solving and collaboration, in addition to programming skills.

We believe there is a need for a fundamental shift in computing education pedagogy. Rather than rely on traditional instructional methods, computing educators could greatly benefit from adopting the instructional methods of another discipline—architecture. As a well-established design science, architecture has a strong track record of producing skilled practitioners through collaborative learning methods in which learners create and discuss their own problem solutions. Design sciences like architecture view solutions not as cut-and-dried, but as arising out of careful, collaborative consideration of tradeoffs and heuristics. Such a design science orientation, we argue, could deeply engage students in computing as a discipline and prepare them for careers in the modern workforce.

We have presented a possible adaptation of the studio-based instructional model for computing education, along with a research program that is currently underway aimed at refining, evaluating, and building a community of computing educators around studio-based instructional methods. We invite interested members of the SIGCSE community to contact us for more information on how they can collaborate in this project by participating in the community and its activities.

## 7. ACKNOWLEDGMENTS

This project is funded by a National Science Foundation CPATH Award (CNS-0721927). Contributions to this work by other members of the research team—Margaret Ross, Michael Trevisan and Rita Vick—are gratefully acknowledged.

## 8. REFERENCES

[1] Anderson, R. and Bendix, L. eXtreme teaching: A framework for continuous improvement. *Computer Science Education*, *16*, *3*, 2006, 175-184.

[2] Astrachan, O. and Rodger, S.H. Animation, visualization, and interaction in CS 1 assignments. In *Proc. ACM SIGCSE Symposium*, ACM Press, New York, 1998, 317-321.

[3] Carlisle, M., Wilson, T., Humphries, 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, 176-180.

[4] 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.

[5] Guzdial, M. *Introduction to computing and programming in Python: A multimedia approach*. Prentice Hall, Upper Saddle River, NJ, 2004.

[6] Hübscher-Younger, T. and Narayanan, N.H. Authority and convergence in collaborative learning. *Computers & Education*, *41*, *4*, 2003, 313-334.

[7] Hübscher-Younger, T. and Narayanan, N.H. Constructive and collaborative learning of algorithms. In *Proc. ACM SIGCSE Symposium*, ACM Press, New York, 2003, 6-10.

[8] 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.

[9] Hundhausen, C.D. and Brown, J.L. Designing, visualizing, and discussing algorithms within a CS 1 studio experience: an empirical study. *Computers & Education 50, 1*, 2008, 301-326.

[10] Hundhausen, C.D. and Brown, J.L. What You See Is What You Code: A 'Live' Algorithm Development and Visualization Environment for Novice Learners. *Journal of Visual Languages and Computing*, *18*, *1*, 2007, 22-47.

[11] Hundhausen, C.D., Farley, S. and Brown, J.L. Can direct manipulation lower the barriers to programming and promote positive transfer to textual programming? An experimental study. In *Proceedings IEEE 2006 Symposium on Visual Languages and Human-Centric Computing*, IEEE, Piscataway, NJ, 2006, 157-164.

[12] Joint Task Force for Computing Curricula. *Computing curricula 2005: The overview report*. Association for Computing Machinery, New York, 2005.

[13] Knox, D. et al. Use of laboratories in computer science: Guidelines for good practice (Report of the Working Group on Computing Laboratories). *SIGCSE Bulletin*, *28*, 1996, 167-181.

[14] McIver, L. and Conway, D. GRAIL: a zeroth programming language. In *Proc. Seventh Int. Conf. on Computers in Education (ICEE '99)*, IOS Press, The Netherlands, 1999, 43-50.

[15] Moskal, B., Lurie, D. and Cooper, S. Evaluating the effectiveness of a new instructional approach. In *Proceedings 35th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, New York, 2004, 75-79.

[16] Patterson, D.A. President's letter: Computer science education in the 21st century. *Communications of the ACM*, *49*, *3*, 2006, 27-30.

[17] Roschelle, J. Designing for cognitive communication: Epistemic fidelity or mediating collaborative inquiry? *The Arachnet Electronic Journal on Virtual Culture*, *2*, *2*, 1994.

[18] Stasko, J.T. Using student-built animations as learning aids. In *Proc. ACM SIGCSE Symposium*, ACM Press, New York, 1997, 25-29.

[19] Tew, A.E., Fowler, C. and Guzdial, M. Tracking an innovation in introductory CS education from a research university to a two-year college. In *Proc. ACM SIGCSE Symposium*, ACM Press, New York, 2005, 416-420.

[20] Williams, L. and Kessler, R.R. Experimenting with industry's "pair-programming" model in the computer science classroom. *Computer Science Education*, *11*, *1*, 2001, 7-20.

[21] Zweben, S. Taulbee survey: Ph.D. production at an all-time high with more new graduates going abroad; undergraduate enrollments again drop significantly. *Computing Research News*, *18*, *3*, 2006, 7-17.