LIVE CODING IN INTRODUCTORY COMPUTER SCIENCE COURSES [*]

Amy Shannon [†]
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15123
amyshann@andrew.cmu.edu

Valerie Summet
Math and Computer Science Dept.
Emory University
Atlanta, GA 30322
vsummet@emory.edu

ABSTRACT

In this paper, we present a study of "live-coding", an active learning technique in which students create code solutions during class through group discussion. We hypothesized this technique might produce greater learning gains than traditional lectures while requiring less time and effort from the instructor. We begin with a discussion of active learning techniques in STEM disciplines and then present a study to evaluate this instructional method in introductory Computer Science courses. We conclude with a discussion of several interesting and positive trends that deserve further investigation.

INTRODUCTION

The traditional introductory computer science course is typically a student's first course in programming. Students in this course learn to write code and to apply basic problem solving principles. Lectures for this course are generally filled with pre-coded examples. However, there may be a better teaching method for students who are learning to program: **live-coding.**

The live-coding technique of problem solving starts with a blank or nearly blank text file. The students generate a code solution, and iteratively test and revise that solution, through guided group discussion during class. This process allows students to practice generating code, testing code, understanding and solving errors, and finding optimal code

---

[†] work done while a student at Emory University

solutions. Live-coding gives students an interactive educational experience, with little additional effort on the part of the professor.

Background

Much research has been done related to computer science pedagogy, very little relates to live-coding. Of the few who have explored live-coding as a teaching practice, most report positive results based on excellent qualitative data, but collect little or no quantitative data.

In 2002, Paxton published a definition of "live programming" as "the process of designing and implementing [code] in front of the class during the lecture period." Paxton used this technique in a non-introductory class to provide an element to lecture that "captures the programming process much more effectively than canned examples ever could"[5]. Paxton's conclusions are unclear in introductory courses, as novices learn differently that those with mastery of a topic [1].

In 2007, the research of Gaspar and Langevin supported "live-coding" in various forms as an effective teaching strategy that helps address two trends in student learning: memorization of code and code tracing for debugging purposes[2]. Live-coding is presented as a way to discourage students from memorizing rather than understanding code, and encourage students to examine code more rigorously, particularly when catching bugs. For both Paxton's and Gaspar and Langevin's work, the primary source of reported data was a student survey, and no data was provided regarding student performance.

In 2012, Nasser Giacaman used extensive live-coding demonstrations to teach parallel computing. Impressively, students in this course are able to "correctly and efficiently multi-thread a sequential desktop application" after only four weeks. Student survey responses indicated live-coding demonstrations and analogies used in class were the most helpful in learning these difficult concepts [3].

In 2013, Rubin studied four sections of an introductory computer science course taught by two professors, where each professor had a control and an experimental section. He compared the overall scores for assignments, exams, and projects and found that the two sections with live-coding examples had significantly higher scores for the coding project and no statistically significant difference for assignments or exams [6]. Rubin's research raises a few questions. For example, live-coding could be more helpful for some introductory topics than others, and the short-term benefits of live-coding (i.e. for in-class quizzes) have not been studied.

METHODOLOGY

In this section, we present a series of experiments designed to explore if live-coding has a greater impact with certain topics more than others, and if live-coding has short-term benefits (i.e. for in-class quizzes) as well as long-term advantages for students' overall learning gains by gathering quantitative data.

The four major topics covered by the experiment were conditional execution (i.e. if statements), iteration (i.e. loops), encapsulation (i.e. methods) and arrays. We split each of these major topics into two subtopics, one of which received the experimental live-coding technique and one of which use pre-coded examples as in a traditional lecture. For example, we split conditional execution into the subtopics of basic `if` statements and `else/else-if`. Iteration was split into the subtopics of `for` loops and `while` loops. Methods were split into two subtopics, one focusing on parameters and arguments while the other focused on return statements and values. Arrays were split into array basics (element access, iteration over arrays) and searching/sorting concepts. We alternated having the control subtopic first since the first subtopic is generally easier than the second subtopic, as they build off of each other. We recognize that there is the possibility of cross-contamination in that instruction on one subtopic could lead to improved learning on the other subtopic. However, this risk was minimized as much as possible through the specific division of subtopics and timing of the quizzes. The only difference between the control subtopics and the experimental subtopics is that the experimental subtopics were explained using live-coding examples and interaction techniques, while the control subtopics were explained using traditional pre-coded examples and explanation only. Live-coding is the only active learning technique used in this course. The subtopics and distribution between control and experimental subtopics are presented in the first two columns of Table 1.

We flipped the classroom by creating lecture videos for students to watch before attending class. This flip created enough time for examples and quizzes during class. At the beginning of each 75 minute class, the instructor led a 5-10 minute question and answer period. Students were then shown an example, beginning with a problem statement in English (no code involved). After discussing a problem solving strategy, a basic solution (in Java) was discussed. The basic solution was then iteratively refined to a final solution, and possible errors were discussed. The example would either be a traditional pre-coded example or a live-coding example. For a traditional example, the basic code solution and each refinement was given to the class (pre-coded) and discussed. For a live-coding example, the class would generate the basic solution and each refinement through discussion, leading to a final solution. The example typically took between 20 and 25 minutes to demonstrate.

Students would then immediately be given an in-class quiz lasting no more than 10 minutes on the topic they had just seen. After the quiz was collected, the class would continue as normal, often into a discussion of the quiz questions or with further examples.

Students were given 11 in-class quizzes throughout the semester. The quizzes were intended to be the main criteria used to determine the effectiveness of live-coding examples. Quizzes generally consisted of at least two questions, where the first question asked for basic understanding of a concept and the second asked student to demonstrate the ability to apply that concept.

As much as possible, we made the two quizzes given for a particular topic match in question type and question difficulty. We used the principles of Bloom's Taxonomy to assess the comparability of questions [7].

We were primarily interested in four major topics covered in this course - conditionals, loops, methods and arrays. These four topics were covered in two quizzes

each (one for live-coding examples and one for traditional examples). We collected data primarily from these eight quizzes.

Each semester, at the end of each course, the department provides anonymous surveys for students to comment on the strengths and weaknesses of the course and instructor. These surveys are not available to the instructor until after the final course grades have been submitted. We examined the responses to these surveys for our research.

Thirty-one students were recruited based on their enrollment in a CS 1 course. Informed consent was obtained from all participants, and the course professor did not see the consent forms until after final grades had been submitted. Two students who withdrew from the course and one student with significant prior knowledge of were excluded. Of the remaining twenty-eight students, 12 were female and 16 male. Students were primarily majoring in STEM topics and ranged from freshmen to seniors.

RESULTS

We collected data for eight quizzes covering four major topics of this course. Each quiz was scored out of 50 points. Table 1 lists the mean quiz score, standard deviation, and number of students for each of these quizzes. Using R, we ran the default t-test on our data, comparing the quizzes in each major topic. This test assumes unequal variance and unpaired data. We operated under a null hypothesis that the two quizzes would have no difference in mean score, with an alternative hypothesis that the means were not equal. Live coding did not have a significant effect for conditional topics ($t(42) = 2.3$, $p = .03$), for iteration topics ($t(46) = 2.0$, $p = .06$), or for encapsulation topics ($t(40) = 1.1$, $p = .3$) or arrays($t(50) = 1.68$, $p=.04$).

Table 1. Quiz Scores by Topic

Quiz Topic	Condition	Mean (of 50)	SD	# students
If Statements	Experimental	36	7	28
Else Statements	Control	42	9	25
While Loops	Control	35	14	25
For Loops	Experimental	41	10	26
Parameters	Control	31	14	26
Return Values	Experimental	31	8	24
Array Basics	Experimental	41	9	26

Searching/Sorting	Control	37	9	26

Figure 1 shows the differences between histograms of the scores on 3 traditional condition quizzes (black columns) and 3 live-coding condition quizzes (gray columns). These graphs represent data from all eight quizzes.

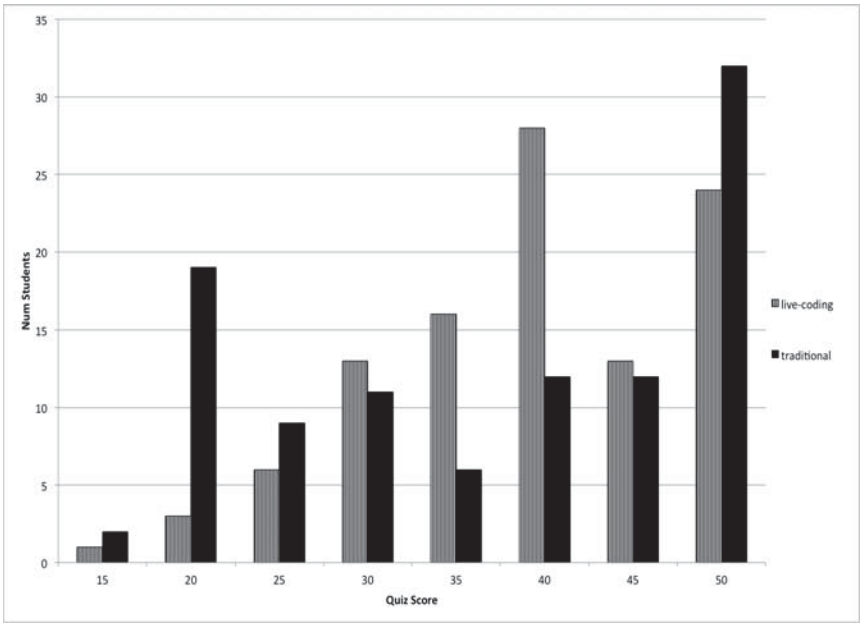

Figure 1 Histograms of Quiz Scores

A departmental end-of-course survey asks three very general questions about the strengths and weaknesses of the course and instructor. The survey did not ask any specific questions related to our changes to the course.

Of the 16 surveys we received, 5 expressed positive opinions about the lecture videos. These students found the videos to be "helpful for studying" and even complained about the lack of videos for a few later topics in the course. Two students mentioned the additional quizzes, saying the constant testing forced them to stay on top of the material. However, they would have preferred advance notification of quizzes. One student made an ambiguous comment that could be directed toward live-coding: "I found the days we actually worked with code on the computer to be particularly helpful".

Our results were largely inconclusive in determining the positive effects of live-coding. However, it is important to note that our changes to the course did no harm. In fact, some students reported benefits from the lecture videos and frequent quizzing. There was no statistical difference in quiz scores between subtopics taught with live-coding examples and those taught with traditional examples. Despite our lack of statistical significance, several of our results raise interesting questions and observations.

ANALYSIS

Some of the changes to this course caused additional work for students. Primarily, we expected everyone to watch a lecture video before attending each class. While the

amount of time was minimal (an average of just under 20 minutes per video), we expected students to complain. We also assumed they would complain about frequent, unannounced quizzes during class that served as an encouragement to attend more classes.

Contrary to our assumptions, students found the lecture videos useful and helpful for studying. Some even complained about the lack of videos for topics towards the end of the course. Students also acknowledged that the frequent quizzing benefited them by encouraging students to stay on top of new information. We found these responses encouraging given that the only prompt provided was, "Comment on the strengths and weaknesses of the course and instructor".

Although we did not have enough data to determine statistical significance, students consistently performed slightly better on the second quiz within each topic (see Table 1). One possible explanation could be that because the second subtopic builds on the first, instruction on the first subtopic improves learning for the second subtopic. Also, the first quiz always tests a completely new topic and concept, while the second quiz tests a new topic, but not a new concept. For example, while loops are the first loops these students learn. for loops are different, but the basic iteration concept still applies. In a study where every topic was taught with traditional examples for one group and live-coding examples for a different group, this phenomenon may not occur or may occur to a lesser degree.

When examining the histograms in Figure 1, notice that live-coding quiz scores are closer to normal distribution and the traditional example quizzes exhibit a bimodal distribution. While we recognize that the small number of students in our study limits the amount of certainty available from these graphs, we believe further investigation into this area could reveal interesting findings regarding the effectiveness of live-coding as the bi-modal distribution of grades has long been a topic of discussion in introductory computer science education (e.g. [4]).

CONCLUSIONS

Based on active learning literature and prior live-coding research, we believe the practice of live-coding deserves further investigation. We would like to conduct a similar study with multiple sections of the introductory computer science course taught by the same professor. This alteration would allow us to have separate classrooms for our control group and experimental group. In addition to increasing the number of students involved in the study, it would also ensure every topic could be taught with live-coding and with traditional examples. Finally, this change would eliminate the challenge of trying to create comparable quizzes on different subtopics. We could also use multiple methods to evaluate student learning gains. In addition to in-class quizzes, the grades from homework, projects, and labs could also be considered. Overall, using multiple sections of the same course, or one section taught across multiple semesters, would greatly improve the investigative power of this research. Given the positive trends our study found regarding student preferences for live-coding, we believe further investigation based on our pilot study as described above is a valuable research endeavor.

REFERENCES

[1] Bransford, J. D., Brown, A. L. and Cocking, R. R. *How People Learn: Brain, Mind, Experience and School.* National Research Council, Washington DC, 2000.

[2] Gaspar, A. and Langevin, S. Restoring "coding with intention" in introductory programming courses. In *Proc. of the SIGITE* (2007), 91-98.

[3] Giacaman, N. Teaching by Example: Using Analogies and Live Coding Demonstrations to Teach Parallel Computing Concepts to Undergraduate Students. In *Proc. of the 26th Intl IEEE Parallel and Distributed Processing Symposium* (2012), 1295-1298.

[4] McCracken et al. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.*33, 4 (Dec. 2001), 125-180.

[5] Paxton, J. Live programming as a lecture technique. *J. Comput. Sci. Coll.*, 18, 2 (2002), 51-56.

[6] Rubin, M. J. The effectiveness of live-coding to teach introductory programming. In *Proc. of SIGCSE*, (2013), 651-656.

[7] Starr, C. W., Manaris, B. and Stalvey, R. H. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin* 40(1), 261-265.