Role of Live-coding in Learning Introductory Programming

Adalbert Gerald Soosai Raj
Department of Computer Sciences and Education
University of Wisconsin-Madison
gerald@cs.wisc.edu

Jignesh M. Patel
Department of Computer Sciences
University of Wisconsin-Madison
jignesh@cs.wisc.edu

Richard Halverson
Department of Educational Leadership and Policy Analysis
University of Wisconsin-Madison
rich.halverson@wisc.edu

Erica Rosenfeld Halverson
Department of Curriculum and Instruction
University of Wisconsin-Madison
erica.halverson@wisc.edu

ABSTRACT

Live-coding is an approach to teaching programming by writing actual code during class as part of the lectures. In a live-coding session, the instructor thinks aloud while writing code and the students are able to understand the process of programming by observing the thought processes of the instructor. In our study, we conducted a live-coding session to two groups of students as a part of a teaching intervention that was originally designed for studying the effects of using the native language for learning programming. We analyzed the student feedback data that was collected and found that many students have mentioned about the usefulness of live-coding for learning programming. We conducted a grounded theory analysis of the student feedback data to understand the value of live-coding for learning introductory programming. We found that live-coding (1) makes the process of programming easy to understand for novice programmers, (2) helps students learn the process of debugging, and (3) exposes students to good programming practices. We also found that students prefer to code along with the instructor during a live-coding session rather than being mere observers.

CCS CONCEPTS

• **Social and professional topics → Computer science education**;

KEYWORDS

Live-coding, Students' perceptions, Computer Science Education

ACM Reference format:
Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. 2018. Role of Live-coding in Learning Introductory Programming. In *Proceedings of 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 22–25, 2018 (Koli Calling '18),* 8 pages.
https://doi.org/10.1145/3279720.3279725

1 INTRODUCTION

Live-coding [12] is a pedagogical approach for teaching programming where the instructor writes actual code from scratch (i.e., without using any skeleton or boilerplate code) on a computer connected to a projector during the class. The main purpose of a live-coding session is to teach *programming as a process* [1, 7]. This approach helps students to fully understand the tasks and techniques that are needed to write a fully-working program, after going through multiple iterations of thinking, designing, coding, and testing. At the end of a live-coding session, along with learning the programming process, the students also learn the syntax of the programming construct that was discussed during the coding session. The effectiveness of live-coding for teaching introductory programming is discussed in [14].

Live-coding is an alternative approach for teaching programming compared to the traditional way of using pre-coded working code examples. There are pros and cons of both these approaches. For example, live-coding has the advantage of exposing the thinking processes of the instructor to the students while it takes a lot more time than explaining using pre-coded examples. On the other hand, using pre-coded examples, helps instructors to cover more topics than that can be covered in a live-coding session but tends to create some misconceptions about programming for novice programmers. For example, the importance of *incremental coding* (i.e., writing a few lines of code and immediately testing them before writing the next few lines of code) cannot be emphasized using static code examples since students may get a wrong impression that the code that is shown during the lecture is the final product that they are expected to produce before they compile/run their program (for the first time).

Prior research on live-coding [3, 12, 14] report on the effectiveness of this approach for teaching programming to novice programmers. In these prior works, students' preferences on live-coding vs. traditional (pre-coded examples) are collected and they show that novice programmers prefer live-coding for learning programming. Even though the student preferences of live-coding for teaching introductory programming are well-known, the reasons for students to prefer live-coding is still unknown.

In our study, we try to understand the reasons behind why students may prefer live-coding as a valuable instructional technique for learning programming. In other words, what is the value of live-coding as an instructional strategy for teaching and learning programming from the students' perspective. To answer this question, we use the open-ended feedback data that we collected from one of the experiments we conducted previously [13, 16]. The experiment's main purpose was to understand the effectiveness of using students' native language for learning a programming language. As a part of this experiment, we conducted a live-coding session and in the open-ended feedback we found that students have mentioned many positive things about our live-coding session. We used this data and performed a grounded theory analysis to find the value of live-coding for learning programming from the students' perspective.

Our research aims at addressing the following question: *What is the value of live-coding as a pedagogical tool for teaching and learning programming from the students' perspective?* We believe that our paper may contribute to the valuable literature on live-coding.

The contributions of our work are the following:

(1) Generate theories on the value of live-coding for learning programming that is grounded on student feedback data.
(2) Provide evidence that supports the prior work on the effectiveness of live-coding as a pedagogical approach for teaching programming.

2 RELATED WORK

Marc J. Rubin conducted a semester-long experiment to understand the effectivess of live-coding vs. static-coding [14] in an introductory programming course. In this study, programming was taught to four groups of students (two control groups and two experimental groups) in a C++ course using two different methodologies. Static-coding was used to teach the control group and live-coding was used to teach the experimental group. He used four surveys and the final grades to assess the effectiveness of live- vs. static-coding. He found that teaching via live-coding is as good as if not better than static-code examples. He also found that students in the live-coding group did significantly (statistically) better than the students in the static-coding group in their final project. He concluded that live-coding may better prepare students to tackle large coding assignments by helping them with good programming practices like iterative testing.

John Paxton taught a Java programming language course by using live-programming as the primary lecture technique [12]. He found that at the end of the course 24 students (80%) preferred the live-programming approach, one student (3%) preferred the traditional approach, and five students (17%) preferred a combined approach. He also collected a survey on what students liked and didn't like about using live-coding for teaching programming. He reported from his experience that live-programming is an effective and fun pedagogical technique for teaching programming. It is important to note that the students had previous programming experience in Ada or C++ and hence these results may not apply directly when live-coding is used for beginning programmers with no prior programming experience.

Soosai Raj et. al. conducted a study in an introduction to programming course in Java. The instructor used three active learning teaching methods, namely mini-lectures, live-coding and in-class coding [17]. The authors ranked these three techniques based on the amount of student activity required. The authors focused on finding out which of these three active learning techniques (that require different levels of student activity) were most preferred by students who learnt programming for the first time. The authors found that students preferred live-coding the most and in-class coding the least.

Russel E. Bruhn and Philip J. Burton taught Java using a studio style teaching method [3]. They used computers within the classroom so that students could try out the programming concepts that were presented during the lecture immediately. In their approach, the instructor presents a concept and then writes some code from scratch (without any skeleton code) and the students follow along with the instructor by writing the program on their assigned computers. They have reported that this style of teaching programming by actually programming using the computers within the classroom helps the average-to-poor scholastic achievers the most, while high-achieving students seemed to do just as well with the typical lecture-style format. They concluded that by combining the theory and practice of programming within the classroom, students learn programming easier and faster. The authors found that lecturing alone, without any hands-on programming, produces lower grades on homeworks and exams.

Amy Shannon and Valerie Summet conducted a quantitative study [15] comparing student learning in live-coding lectures vs. traditional lectures (which used pre-coded examples). They conducted a within-subjects design where four topics (conditionals, loops, methods, and arrays) were taught to the same group of students. All these topics were divided into two parts (e.g., loops were divided as while loops and for loops) and one part was taught using pre-coded examples while the other part was taught using live-coding. They found that there was no statistically significant difference between the test scores of participants using these two types of teaching methods.

In 2012, Nasser Giacaman taught parallel programming concepts using Java to undergraduate students [9]. He took a practical hands-on approach to teaching this course where he used live-coding for teaching parallel programming. He started from scratch in some of the demonstrations and in others he used some form of base code to start with so that he may focus on new programming features that is being taught. From the students' survey, he reported that the students found the analogies and the live-coding sessions to be most useful for their learning of parallel programming.

Alessio Gasper and Sarah Langevin studied the benefits of instructor led live coding sessions and student led live coding sessions [8]. They reported that the instructor led live-coding sessions for teaching programming helps students to focus on understanding the program in a systematic way rather than memorizing it. They also reported that the student led live-coding sessions helps the students to examine their code more rigorously, especially when catching bugs. The main focus of their work was to help students in introductory programming course to code with intention rather than hacking their way to make their program work to pass the test cases. They collected survey about students' preferences about live-coding and found that most of the students preferred live-coding.

Most of these previous works evaluate the usefulness of live-coding either using student surveys which asks about their preferences towards live-coding or using student responses to questions like 'What do you like/dislike about live-coding'. Even though all these prior works report the effectiveness of live-coding as a pedagogical strategy for teaching introductory programming, none of these prior works tells us what makes live-coding a useful pedagogical strategy for learning programming from the students' perspective. In our work, we try to answer this question by digging deep into the open-ended feedback we got from students about our live-coding sessions. We use a grounded theory approach to create our theories about the role of live-coding in teaching and learning programming.

3 THEORETICAL BACKGROUND

Cognitive apprenticeship [5, 6] is pedagogical model that focuses on making thinking visible. This model is based on the ancient model of apprenticeship where a student learns a skill from a master by working under the master's guidance. This model of education was mainly used in fields like medicine, shoe-making, painting, agriculture, etc. In modern times, formal schooling replaced this apprenticeship model but still this model is used whenever it's essential (e.g., how new surgeons learn about surgery from experienced surgeons). The emphasis of cognitive apprenticeship is on acquiring cognitive skills (e.g., programming and problem-solving).

The cognitive apprenticeship model focuses mainly on the process rather than focusing only on the end product. With respect to programming, the process is the multiple iterations of writing code, compiling the code, correcting syntax errors, running, debugging, etc that are needed to write a program and the end product is the final program (code). When pre-coded fully-working code examples are presented to students who are learning programming for the first time, they may have a misconception that the program was written from start to end without any mistakes in a single pass. This misconception causes some students to write code from start to end without breaking down the program into multiple smaller parts. When they follow this approach as opposed to performing incremental coding they may be overwhelmed looking at the error messages that pop up when they compile their code at the end. We can correct this poor practice of writing the whole program at once and compiling it at the end by explicitly teaching the students how to program by teaching programming using live-coding.

The three stages of instruction in cognitive apprenticeship are: (1) Modeling, (2) Scaffolding, and (3) Fading. In the modeling phase, the students learn a conceptual model of the process from the teacher. This step is primarily instructor driven. In a live-coding session, this is the part where the instructor teaches how to write a program for a particular task by actually writing a program from scratch using his/her laptop (that is projected to the entire class). The students may optionally follow along with the instructor by writing the same program on their computer or they may just observe how the instructor is writing code and take notes of some things that they need to remember.

The next stage is the scaffolding stage where the teacher gives some task to the students and provides the needed scaffolds for achieving this task. In a live-programming classroom, during the scaffolding phase, the instructor gives a problem to the students based on the topic that was just taught during the live-coding session and asks them to write code for this problem. The instructor provides the needed scaffolds by writing some skeleton code or algorithm or data structure that is needed to solve the problem. The instructor also walks around the classroom (sometimes the teaching assistants are also present) to help the students as and when needed.

In the last stage, named as the fading stage, the teacher removes the scaffolds so that students can learn to do the task by themselves. In an apprenticeship model, this is the phase when the students try to master the skills on their own without any support from the teacher. In a live-coding classroom, this can be compared to the instructor giving some take-home programming assignment without any skeleton code (i.e., without any scaffolds) so that the students may try to work on their own and develop the skill of programming.

Cognitive apprenticeship model has been widely applied in teaching programming to novices with positive effects [2, 4, 11, 18]. These successes suggest that cognitive apprenticeship model is a suitable model for teaching and learning computer programming for novices. Live-coding is a pedagogical approach for teaching programming which is based on this model of cognitive apprenticeship and so we believe that it is important to understand the learning benefits of live-coding from the students' perspective.

4 METHODOLOGY

In this section, we explain the methodology that we used to conduct the experiment and to collect the data. We would like to remind the reader that this experiment was originally designed for evaluating the effectiveness of using a students' native language along with English for teaching programming and that live-coding was one of the methods that were used for teaching programming during this intervention. We begin by describing the participants in our research.

4.1 Participants

The experiment was conducted in a well reputed Engineering college in Chennai, Tamil Nadu, India. Two groups of first-year students, enrolled in two different sections of a data structures course were selected for the study. One group was treated as the control group and the other group was treated as the experimental group. The total number of students in the control and the experimental group were 52 and 51 respectively. All these students have previously taken a programming course in C [10] and data structures was their second programming course.

4.2 Experimental Procedure

The following activities were performed with both the control group and the experimental group as a part of our intervention. There was a pre-test, three in-class lectures, a live-coding session, and a post-test. The programming (coding) was done in C [10] (a high-level programming language) and the questions in the pre-test and the post-test were in English for both groups. We collected an open-ended feedback from the students in both groups to understand what they felt about our intervention.

We intentionally omit the details of the pre-test, three classroom-based lectures, and the post-test from this paper since they are not relevant to the topic of this paper. We refer the reader to [16] to understand more details about these steps in the experimental procedure. We mainly focus on the live-coding session we conducted and the open-ended feedback we collected from the students.

4.2.1 Live coding session. Following the three classroom-based lectures on linked lists, a live-coding session [14] was conducted for about 90 minutes. The instructor projected his laptop on a screen, and wrote C code for the following linked list functions from scratch:

(1) adding a node at the beginning of the linked list
(2) printing all the elements in the linked list
(3) deleting all the nodes from the linked list

The instructor was thinking aloud throughout the live-coding session. He showed the students how he would go about writing the code for these three functions. He also showed them some common sources of errors while writing code for linked lists. The content taught during the live-coding session was the same for both the control group and the experimental group.

The main differences between the live-coding session for the two groups were the following: The live-coding session was conducted *only in English* for the students in the control group. Also, the students in the control group were required to communicate with the instructor and their classmates during the live-coding session only in English.

On the other hand, the live-coding session was conducted using both *English and Tamil* in the experimental group, and the students were free to communicate in any of those two languages, whichever they felt more comfortable with, during the live-coding session.

4.2.2 Open-ended feedback. We collected open-ended feedback from the students about our teaching methodologies. All the students in both groups wrote their feedback in English, although the choice of language for the feedback was left to the students.

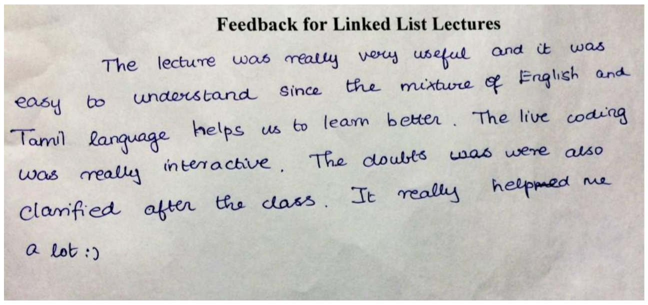
The data we used for our analysis in this paper is this open-ended feedback that was collected from the students after our intervention. It was interesting to find that even though our intervention mainly focused on finding the effectiveness of students' native language for learning programming, many students have explicitly mentioned the usefulness of the live-coding session that we used as a part of our intervention, in their open-ended feedback. This made us interested in analyzing this student feedback data with the intent to find the value of live-coding for teaching introductory programming.

A sample open-ended feedback from a student is shown in Figure 1. In this sample feedback, we can see that the student has mentioned the usefulness of both the mixed language instruction and live-coding.

The complete open-ended feedback from the students in the experimental group and the control group can be found at http://bit.do/feedback_experimental and http://bit.do/feedback_control respectively.

5 DATA ANALYSIS

In this section, we describe our data and the analytic methods that we used to analyze the student feedback data to answer our research questions. We also explain our rationale behind choosing these analytic methods.



Feedback for Linked List Lectures

The lecture was really very useful and it was easy to understand since the mixture of English and Tamil language helps us to learn better. The live coding was really interactive. The doubts was were also Clarified after the class. It really helpped me a lot :)

Figure 1: A sample open-ended feedback.

The type of data (i.e., open-ended student feedback) we had and the research question that we were trying to answer made us choose Grounded Theory for our data analysis. We were interested in analyzing every student feedback that mentiond live-coding and develop a theory around the various themes, concepts, and categories that evolve from the data. We believe that building this theory grounded on our data would enable us to answer our research question about the value of live-coding as an instructional strategy from the students' perspective.

The different steps that we performed in our data analysis are: (1) Data filtering, (2) Coding, (3) Categorizing, and (4) Theory building.

5.1 Data Filtering

The first step in our data analysis process was to filter out the student feedback data that explicitly mentiond live-coding. This was a necessary step in our data analysis since not all students mentiond live-coding in their feedback. This enabled us to focus only on the data of interest for answering our research questions. This step also helped us in reducing the amount of data that we'll be coding as a part of our next step.

We took a print of the student feedback and marked all the feedback that mentioned something about live-coding using an asterisk (*) sign. This was done so that we may focus only on this filtered feedback in the next stage of our data processing (i.e., coding). The criteria for a student feedback to pass this filter is by having any one of the following terms: live-coding, live-programming, coding session, programming session, in-class programming, coding demo, programming demo, and some variants of these terms (e.g., interactive coding).

A sample feedback that passed our filtering test is shown below: *"This lecture is really interesting and interactive. That practical live coding is the best part in it. Actually, the most convenient way for teaching computer programming is to have a board at one end to explain the logical view of the problem and at the other end there should be live coding to implement the logic using code. I always prefer this. Maybe this could help in better understanding."*

5.2 Coding

In this stage of our data analysis, we read all the feedback that were filtered from the previous stage and underlined the sentence(s) in these feedback that mentioned something about live-coding. While underlining the different sentences, we also added a code that we thought best summarized the idea described in that particular sentence.

Coding was done at the granularity of a sentence. In other words, code(s) were assigned to individual sentences. Usually only one code was assigned for each sentence but if needed multiple codes were assigned to the same sentence. For example, as shown in sample_coding[1], the codes "interesting" and "helpful" were assigned to the following sentence: "The coding session was really interesting and helpful."

If a single student feedback contained more than one sentence that mentioned live-coding, we used different color codes to clearly identify which code corresponds to which sentence in the feedback. See the first students' feedback in sample coding (footnote 1 below) to understand how this was done while coding.

Note that the color codes weren't created to group the codes into categories but rather to distinctly differentiate the different codes that were assigned to multiple sentences that were nearby (e.g., within the feedback from a single student).

Codes were generated using the keywords in the first few feedback. For coding the rest of the feedback, if a sentence matched the idea represented by a particular code, then the same code was reused but if a sentence did not fit into any of the codes that were already present, then a new code was created.

5.3 Categorizing

Once our codes were created and finalized, we were interested in grouping them into related categories. In this section, we explain the process that we followed to group the related codes into different categories.

We went through the generated codes multiple times so that we may get some high-level idea about the different categories these codes may be grouped into. After going through the codes multiple times, we came up with the following four categories:

(1) **Positives:** Benefits of live-coding
(2) **Debugging:** Live-coding as a means to learn debugging
(3) **Negatives:** Things to improve during live-coding
(4) **Long-term benefits:** Learning useful skills from live-coding

The different codes that are part of these four high-level categories are shown in categories[2]. If the same code appeared multiple times, then those codes were marked with additional vertical lines near them. The scheme we followed here was that the first four lines will be vertical and the fifth line will be crossing over these first four lines diagonally so that we may group these lines as groups of five. For example, the code "better understanding" in the "positives" category had occurred 7 times (5 + 2) in the feedback data (see the Figure in footnote 2).

The categories we created gave us a better understanding of the different ways in which live-coding is useful as an instructional technique for students learning programming. For example, before we started categorizing the codes into categories, we didn't know that live-coding helped students understand the process of debugging. Similarly, we also didn't realize that live-coding taught some useful skills that students may use in the long run (e.g., indenting their code, thinking like a programmer, connecting the theory and practice of programming, etc). The process of categorization paved the foundation for developing some useful theories about the value of live-coding for learning programming.

[1]http://bit.do/sample_coding
[2]http://bit.do/sample_categories

Some sample lines from the student feedback that were categorized under these categories are shown below:

(1) **Positives:** Benefits of live-coding
"The live coding demo shown increased a little of my programming skills. It was very good and in this way I can develop my programming skills."; "I actually found the live coding program very helpful, it was like very easy to understand other than just teaching us the theoretical concepts of programming during class hours."; "Live coding demos were great and helped us. Hope to become a better programmer from today."
(2) **Debugging:** Live-coding as a means to learn debugging
"The live coding session helped to understand the possible bugs that occurs while coding."; "The live coding session was really useful. In that session, I came to know about the errors what we do while coding." "We felt good to have a live coding classes. Though it takes time, it is very useful in understanding the code and rectifying the errors and mistakes thereby itself."
(3) **Negatives:** Things to improve in live-coding
"Live coding session was good. It could have been more useful if each and every student can type the code as they see on the project so that they too can participate in it.";
(4) **Long-term benefits:** Learning useful skills from live-coding
"It is the first time I have been in a class of interactive coding session where you learn while you code. This way of approach has definitely helped me (a beginner in data structures) understand linked list easily. I guess this lecture has sown the seed for me as well as my classmate to do programming like how a programmer would do."

5.4 Theory Building

In this section, we explain the data and the processes we used for creating the theories on live-coding and our rationale behind why live-coding is a useful pedagogical technique for teaching programming to novice programmers.

5.4.1 *How does live-coding help understand the process of programming?*

When students begin to learn programming, usually they don't have a good idea about where to start, how to break down a program into multiple smaller parts, when to define functions, the order in which the functions should be implemented, etc. So, if the only thing that the students see in class are pre-coded working examples of code, they might get a wrong impression that their code should look similar when they write it in their first try. When they find out that they are not able to produce working code for their programming assignments similar to the ones they saw in class, they might develop inferiority complexes about their programming skills.

Learning Benefits. Linked lists was the first data structure that was taught to these students that used dynamic memory allocation. Although the learning curve is very high when beginning to learn linked lists since it involves difficult concepts like self-referential structures, the students felt that the live-coding session helped them to understand the concept of linked lists in a better way than the traditional approach of using pre-coded code examples that these students were previously used to. Sample student comments include: *"Linked list is the new concept for me and hence working out this lively in the live section of coding helped us to avoid the errors and helped us to understand more clearly." "The live coding session was helpful to understand the concepts that were theoretically taught in the class."*

5.4.2 How does live-coding help students learn the art of debugging? Debugging is an important skill in programming which is mostly ignored when programming is taught only using pre-coded examples. This is mainly because the instructor writes the pre-coded examples while preparing for the lectures and makes sure that there are no bugs in the code so that (s)he can use it for explaining a particular programming concept. In live-coding, even though the instructor may have tried coding the example beforehand, it's usually the case that some bugs may occur in the instructor's code while performing a live-coding session. This gives the opportunity for explaining the process of debugging when it is most needed. For example, if the bug is easy to find, the instructor may use simple print statements to find the bug or in case of more difficult bugs the instructor may use a debugger to find the bugs. Even though this process may be time-consuming sometimes, it's one of the best opportunities for the students to learn the art of debugging as they can see an expert debug his/her code right in front of them. Even if an instructor writes code without any errors, (s)he may intentionally introduce the errors in their code with the intention to teach students this useful skill.

Many students mentioned that live-coding helped them understand the process of debugging. *"The online (live) coding session was very useful as we could witness the most commonly faced errors."; "The live coding session helped to understand the possible bugs that occurs while coding."; "Linked list is the new concept for me and hence working out this lively in the live section of coding helped us to avoid the errors and helped us to understand more clearly."; "The live coding session was really useful. In that session, I came to know about the errors what we do while coding." "Though it takes time, it is very useful in understanding the code and rectifying the errors and mistakes thereby itself."; "Coding and teaching simultaneously gave us an idea about frequently occurring bugs."*

5.4.3 Things to improve during a live-coding session. Since our intervention was mainly designed to understand the benefits of bilingual CS education, we didn't try to conduct our live-coding session in a lab where the students too would have access to computers. Usually, live-coding is more effective if students can follow along with the instructor by writing their own code on their computers. Even though the instruction is usually led by the instructor, if students are able to follow along they will be more engaged in the class. Some students expressed this concern as shown in the category on the negatives of live-coding in Section 5.3.

Some students expressed concerns about learning a lot of content within a limited amount of time. We taught the topics of creating a node structure, adding an element at the end of the linked list, printing a linked list, finding the length of the linked list, deleting an element from the linked list, and freeing all the nodes in a linked list during the 90-minute live-coding session. We acknowledge that this may have been overwhelming to some students as seen from these sample student comments: *"The live coding session was understandable but it is not fair to teach the whole code in a short period. Other than this it was an awesome class."; "Very useful lectures. Understood almost everything regarding the topic. And also how to actually write code in a professional way, how to start a problem. The only small issue was time. As a biology student, it was a bit too overwhelming for me. Maybe if it was stretched over more classes, it would have been better.".*

5.4.4 What are the long-term benefits of live-coding? Some students expressed their opinions on how the things they learnt during the live-coding session may have some long-term benefits for them as future programmers. Live-coding also helped students in understanding the link between the theory and practice of programming concepts.

Increased interest in programming. Students mentioned that the live-coding session increased their interest in learning programming and problem solving. Sample comments include: *"The class was super fun and I was able to concentrate without any distractions because of the way you explained linked lists. The live coding session was very helpful and I would miss your classes. I always wanted to run out of the class as soon as the time is up. But your classes made me want more of such explanation making me wanting more to learn programming and the way of solving problems using code more precisely so as to emerge as a tech fellow." "The live coding demo shown increased a little of my programming skills. I would prefer an extra classes of these teaching methods. It was very good and in this way I can develop my programming skills."*

Connecting theory with practice. The live-coding session gave the students some idea about how the theoretical concepts taught during the in-class lectures (e.g., diagrammatically explaining the process of adding or deleting a node in a linked list) is actually related to the practice of programming (i.e., how to actually implement them in code).

Sample student comments include: *"I actually found the live coding program very helpful, it was like very easy to understand other than just teaching us the theoretical concepts of programming during class hours. There were no cons actually. I really could not believe I was able to focus one full hour in class without being distracted even for a minute. Though linked list looked like a vague topic, through various teaching strategies taken by you, I was able to get an overall clear idea on the concept. I also learnt the different ways of programming on how to approach problem etc."; "The idea of live coding is very useful as we get to see how the theoretical concepts work."*

6 DISCUSSION

In this section, we provide an interpretation of our results, state the limitations of our study, and provide some directions for future work.

6.1 Interpretation of Results

The major findings of our study are the following theories on the value of live-coding from the students' perspective:

(1) Live-coding is a useful pedagogical tool to help students understand the process of programming.
(2) Live-coding helps students to understand the process of debugging (i.e., identifying and correcting the bugs in a program).

(3) It may be better to involve students in the process of live-coding by making them code along with the instructor on their own laptop as opposed to the instructor writing the code and the students being mere observers.
(4) Live-coding may have some long-term benefits for the students (e.g., learning program design and indentation, increased confidence in their programming skills, etc.) that may help the students in their future programming careers.

Our findings suggest that students value live-coding as an effective pedagogical tool for learning programming. The reasons for this may be because live-coding makes the thinking process of the instructor visible to the students and they are able to learn the process of programming as opposed to seeing the final product (pre-coded code examples). Our findings may be valuable for an instructor who would like to use live-coding in his/her introductory programming class but is unsure of its benefits from the perspective of the students.

Our results show that live-coding which is based on the cognitive apprenticeship model is a viable method to teach programming. They also suggest that while teaching programming, focusing on the process of writing a program is as important as it is to focus on the end product (i.e., a working program). This shows that teaching methods based on cognitive apprenticeship should be explored more in the area of Computer Science Education.

Majority of the results from the previous studies on live-coding (shown in Section 2) show that students prefer live-coding over traditional lectures for learning programming. Our study builds on top of these prior work and adds value to the literature on live-coding by explicitly dissecting the major reasons for the students to consider live-coding as a valuable instructional technique for learning programming. Our results also show some ways that live-coding could be improved.

An alternative explanation of the results could be that since our students experienced a live-coding session for the first time in their lives, they may have become overly enthusiastic about this teaching method and this may have motivated most of the students to write great things about live-coding. But some of the previous results on students' preferences on live-coding [12, 15] were collected at the end of the semester. By this time the students were used to this approach and still they seemed to have preferred live-coding for learning programming.

6.2 Limitations

Our study was primarily conducted to find the effectiveness of using the students' native language for learning programming. We used live-coding just as a component of our teaching methodology and initially weren't interested in gathering data on live-coding. Therefore, we didn't collect any specific data on the effectiveness of live-coding at the end of our intervention. All the data that we presented in our study was part of the overall feedback that students provided for the following prompt: 'Feedback for linked list lectures' (also shown in Figure 1). Because of this the data that we had for our analysis was very limited. Only if we had asked some specific question about live-coding at the end of our intervention, we believe that we would have been able to collect more valuable data than what we currently have.

6.3 Future Work

Based on our findings, we know that live-coding is an invaluable tool for teaching programming to novices. Live-coding may not be a valuable tool for people who already know programming. For example, students who already know a programming language (e.g., Java) and who are learning a new programming language (e.g., C++) may not appreciate the value of live-coding since they may have already learnt the process of programming while learning their first programming language. It would be interesting to conduct a study to find out the value of live-coding as a pedagogical tool for learning programming from the perspective of people who already know a programming language.

In our current study, we didn't gather any specific data on live-coding using surveys/questionnaires since our original intervention was not geared towards studying the role of live-coding. Although our intervention was not targeted towards live-coding, based on the student feedback, it seems that live-coding is a valuable technique for teaching introductory programming. Therefore, we propose that, in order to know more about live-coding, we need more controlled experiments mainly targeted towards this teaching technique. By doing so, we would be able to understand the pros and cons of live-coding in much more detail.

7 CONCLUSION

We conclude that live-coding is a valuable pedagogical technique for learning programming as it helps students to understand the process of programming by making the instructor's thinking visible, helps students learn valuable debugging skills, and helps students by introducing them to good coding practices. We also found that it may be better to involve students during the process of live-coding by making them code along with the instructor.

ACKNOWLEDGMENTS

We thank Prof. Saswati Mukerjee and Prof. Arockia Xavier Annie for helping us conduct experiments in India.

REFERENCES

[1] Jens Bennedsen and Michael E Caspersen. 2005. Revealing the programming process. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 186–190.
[2] Toni R Black. 2006. Helping novice programming students succeed. *Journal of Computing Sciences in Colleges* 22, 2 (2006), 109–114.
[3] Russel E Bruhn and Philip J Burton. 2003. An approach to teaching Java using computers. *ACM SIGCSE Bulletin* 35, 4 (2003), 94–99.
[4] Michael E Caspersen and Jens Bennedsen. 2007. Instructional design of a programming course: a learning theoretic approach. In *Proceedings of the third international workshop on Computing education research*. ACM, 111–122.
[5] Allan Collins, John Seely Brown, and Ann Holum. 1991. Cognitive apprenticeship: Making thinking visible. *American educator* 15, 3 (1991), 6–11.
[6] Allan Collins, John Seely Brown, and Susan E Newman. 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. *Knowing, learning, and instruction: Essays in honor of Robert Glaser* 18 (1989), 32–42.
[7] Rex E Gantenbein. 1989. Programming as process: a "novel" approach to teaching programming. In *ACM SIGCSE Bulletin*, Vol. 21. ACM, 22–26.
[8] Alessio Gaspar and Sarah Langevin. 2007. Restoring coding with intention in introductory programming courses. In *Proceedings of the 8th ACM SIGITE conference on Information technology education*. ACM, 91–98.
[9] Nasser Giacaman. 2012. Teaching by example: using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 1295–1298.
[10] Brian W Kernighan and Dennis M Ritchie. 2006. The C Programming Language. (2006).

[11] Michael Kölling and David J Barnes. 2004. Enhancing apprentice-based learning of Java. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 286–290.
[12] John Paxton. 2002. Live programming as a lecture technique. *Journal of Computing Sciences in Colleges* 18, 2 (2002), 51–56.
[13] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M Patel, and Richard Halverson. 2017. What Do Students Feel about Learning Programming Using Both English and Their Native Language?. In *Learning and Teaching in Computing and Engineering (LaTICE), 2017*. IEEE, 1–8.
[14] Marc J Rubin. 2013. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 651–656.
[15] Amy Shannon and Valerie Summet. 2015. Live coding in introductory computer science courses. *Journal of Computing Sciences in Colleges* 31, 2 (2015), 158–164.
[16] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M Patel, and Richard Halverson. 2018. Does Native Language Play a Role in Learning a Programming Language?. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 417–422.
[17] Adalbert Gerald Soosai Raj, Jignesh M Patel, and Richard Halverson. 2018. Is More Active Always Better for Teaching Introductory Programming?. In *Learning and Teaching in Computing and Engineering (LaTICE), 2018*. IEEE.
[18] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. 2011. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 93–98.