Is More Active Always Better for Teaching Introductory Programming?

Adalbert Gerald Soosai Raj
Computer Sciences and Education
University of Wisconsin-Madison
Madison, Wisconsin, USA
Email: gerald@cs.wisc.edu

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

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

***Abstract*—Introduction to programming is usually taught using a wide range of instructional techniques. Some common techniques among them are mini-lectures, live-coding and in-class coding. Each of these three techniques require varying level of student activity. In this study, we taught programming to a group of students using these three techniques. We collected data in the form of a survey to understand the students' perceptions on these three instructional techniques. The results suggest that students like techniques that require a moderate level of student activity (e.g., live-coding) more when compared to techniques that require a great deal of student activity (e.g., in-class coding). We believe that our work has the potential to help instructors design their instructional techniques using a student-centric approach.**

I. Introduction

Introduction to programming is usually taught using a variety of instructional techniques like traditional chalkboard based lectures, powerpoint presentations, live-coding (i.e., writing a program on a computer from scratch), in-class coding (asking the students to write some code on their own to solve a programming problem), etc. Among the many techniques that are employed to teach programming to beginners, the most commonly used techniques are mini-lectures [1], live-coding [2] and in-class coding [1].

These three pedagogical strategies require varying levels of student activity. Students usually listen, understand concepts, and take notes during a mini-lecture. During live-coding, students listen, understand code, code along with the instructor, draw some memory diagrams [24], and take some notes in the form of comments in their programs or separately in their notebooks. In an in-class coding activity, students are expected to code by themselves. So, they have to read some specifications about what the code should do, think about the logic for the program, write some pseudo code, look up some references for syntax (e.g., Javadoc), write a program, compile it, and debug it until it works.

Based on the level of student activity needed, these three instructional strategies are ordered as shown below in the order of increasing student activity.

1) Mini-lecture (least active)
2) Live-coding
3) In-class coding (most active)

Although these three instructional strategies are used commonly by instructors for teaching introductory programming, a study of students' perceptions of these three pedagogies in Computer Science is fairly limited. Similarly, even though there are numerous reports that show the effectiveness of the active learning instructional techniques in Computer Science [4], [5], [6], [7], [8], [9], a comparison of students' perceptions on these instructional techniques is missing. In other words, the answer to the following question is still unknown.

"*Do students prefer instructional techniques that require them to be more active during class when compared to techniques that may require only moderate or minimum level of student activity?*"

In this study, we present some preliminary results on students' perceptions of the three instructional techniques (i.e., mini-lectures, live-coding, and in-class coding) that were used for teaching an introductory programming course using an active learning approach. Using the students' perceptions, we rank these three instructional techniques based on students' preferences.

We believe that by using a student-centric approach for understanding students' perceptions on instructional techniques, our work has the potential to spark future research in this direction, where the instructors consider the students' point of view too while designing instructional techniques for teaching introductory programming.

II. Related Work

Byron S. Gottfried in his experience report [1] highlights the importance of *mini-lectures* and *in-class coding* activities for effectively teaching introductory programming. He also discusses the disadvantages of traditional techniques like 50-minute lectures and the use of detailed examples written on a chalkboard and states that they should be avoided. This work shows the effectiveness of *mini-lectures* and *in-class coding* from the instructor's perspective. We consider our study to be an extension to this report, where we mainly focus on students' perceptions on *three* commonly used active learning techniques (including *live-coding*) used to teach introductory programming.

The importance of using both instructor led presentations and in-class student activites in an active learning class for Computer Science are reported by McConnell in his study

on active learning and its use in Computer Science [4]. He suggests instructors to have a mini-lecture component at the beginning of the lecture mainly to discuss the concepts that are critical or may be difficult to understand from the students' perspective. Similarly, in our study, we used a combination of instructor led mini-lectures and in-class student activites. We also added an additional live-coding component and we mainly focused on reporting the students' perceptions on these techniques.

In flipped courses, students use out of class time to watch pre-recorded lectures or read text books and use the time in the class for labs or discussions. Baldwin describes his experience with a flipped class model for teaching an introductory programmimg course to non-major (Maths) students [10]. The students in this study have reported dissatisfaction with active learning since they felt that the flipped classrooms abandoned students to learn on their own from the videos and the readings. Based on this finding, in our work we made sure that we have a mini-lecture component before we start the more active components like live-coding and in-class coding.

Machemer and Crawford conducted a study [11] in a single cross discipilinary class to assess the students' value about traditional, active, and co-operative classes. The study found that students valued the lectures the most. Co-operative learning was not a favourite for the students as they did not want to be responsible for their group's learning. Also, they found that whichever method yielded the best results in exams was most valued by the students.

In this study [12] by Lumpkin et. al., college students from 5 college courses (related to Sport Management, Human Sexuality, Sport Finance, Exercise, and Ethics in Sports) were taught using short lectures, various in-class and out-of-class exploratory assignments, group discussions and other engaging activities. They found that students felt that active learning positively impacted their learning.

Smith and Cardaciotto compared students' perceptions of active and passive activities [13]. In an introductory psychology class, students were divided into two groups for completing out-of-class group exercises for learning the course materials. One group was given active learning exercises while the other group performed content review activities (passive learning). Though the students in the active learning group reported greater retention of and better engagement with the course material, they did not report greater enjoyment while doing those activities when compared to the passive learning group. In our study, we rank three instructional techniques used in Computer Science, that require varying levels of student activity based on students' perceptions.

Prior works have also considered exploring alternate pedagogical approaches. Van Gorp and Grissom [14] explore constructive and collaborative learning in introdcutory programming classrooms, and empirically evaluate these techniques. Further, Rathika [15] studies a games-based approach to teaching and Simon et al. [16] report experiences about peer instruction in introductory computing. In contrast, our work focuses primarily on understanding student's perception of the three commonly used instuctional techniques used in an active learning Computer Science classroom.

III. Research Question

The following are our research questions:

1) What difference (if any) is there in the way computing students perceive instructional techniques that require varying levels of student activity? Or in other words, "Is more active always better for learning introductory programming?"
2) How do students perceive an active learning classroom that uses a combination of mini-lectures, live coding, and in-class coding for teaching introductory programming when compared to a traditional lecture-based classroom?

IV. Methodology

In this section, we describe the course details and three techniques (mini-lectures, live-coding, and in-class coding) used to teach introdcutory programming in detail. We also describe the survey that was used to understand students' perceptions of these three instructional techniques individually and our active learning methodology as a whole.

A. Course Details

The introductory programming course in this study was taught at a large, research-intensive, public university in Summer 2016 for 8 weeks using mini-lectures, live-coding, and in-class coding. There were a total of 76 students in the class from various disciplines. The class comprised of 20 graduate students and 56 undergraduate students. There were 4 class meetings every week from Monday to Thursday, each 75 minutes duration. There were no separate lab sessions. The regular class meetings were organized in a way that they served the purpose of both the lectures and the lab sessions. A typical active-learning classroom similar to the ones used in the SCALE-UP project [17] was used for teaching this course. Even though the classroom had laptops for every student, most of the students preferred to use their own laptop.

Each class meeting was roughly split among the following three components as follows:

1) Mini-lecture: 20 to 30 minutes
2) Live-coding: 30 to 40 minutes
3) In-class coding: 15 to 20 minutes

The same structure as described above was used for teaching all the classes throughout the semester except three classes that were used for review sessions and two classes that were used for in-class exams. In the following section, we describe the first technique that we used for teaching, namely mini-lectures.

B. Mini-lectures

The initial 20 to 30 minutes of the class time was spent for lecturing. The instructor introduced a sample problem with the intention to help students understand that it may be difficult or even impossible, to implement a solution for this problem using the programming constructs that they had studied till

then. For example, before introducing arrays, the instructor started the lecture with the following problem:

"In last class, we wrote a program for finding the maximum of three numbers. What should we do if the problem is to find the maximum of ten numbers?"

This made the students to think about this problem and come up with solutions like: *"We can create 10 variables for each number and then find the maximum of them using multiple if-else statements to find the maximum of these 10 numbers."* The instructor tried to show the students how difficult or cumbersome it may be to write code to find the maximum of 10 numbers using 10 separate variables.

Some students who had some programming background already may said, *"We should use arrays to solve this problem!"* At this point, using this initial discussion as a motivation, the idea of *arrays* was discussed. We acknowledge that we really do not need arrays to find the maximum of 10 numbers since a loop will do. Although a problem like finding all the numbers larger than the average is more relevent for arrays, we preferred to start with simple problems and then move on to more complicated ones.

In each mini-lecture, the following pedagogical tools were used to make the lectures as interesting as possible:

1) Analogies: Whenever possible, the instructor used an analogy to help students to understand a programming concept during the lecture. For example, to help students understand the idea that the actual array elements that are created on the heap are actually different from the reference variable for the array that is created on the stack, the following analogy was given:

"Let us assume that my house is located at the following address: 206 Eagle Heights, Apt J, Madison, Wisconsin 53705. If I invite my friend to my house, then I'll write the address of my house on a sheet of paper and give it to him/her. Now, using that paper as a reference, my friend would be able to locate my house and come and visit me."

In a similar way, the actual array (i.e., the house) is different than the array reference variable (i.e., the sheet of paper with the house's address on it) that is used to access the array. This analogy was inspired from a stack overflow question[1]. The usefulness of analogies and metaphors for teaching programming is also discussed in the literature [18], [19]. Although analogies are helpful for short term learning gains, there is no evidence that they are helpful for long term learning gains [20].

2) Diagrams: "A picture is worth a thousand words". Visual representations in the form of memory diagrams, flow charts, class diagrams, state diagrams, etc., were used in almost every lecture to help students visualize the ideas of programming.

For example, when explaining the idea of array creation, the diagram in Figure 1 was drawn in order to explain what each part of the array creation statement meant and to visually show what happened in the computer's main memory when each part of that statement was executed. The order of execution of the different parts of this statement was shown using steps numbered 1 to 3. The usefulness of these memory diagrams for teaching programming can be found in [24].

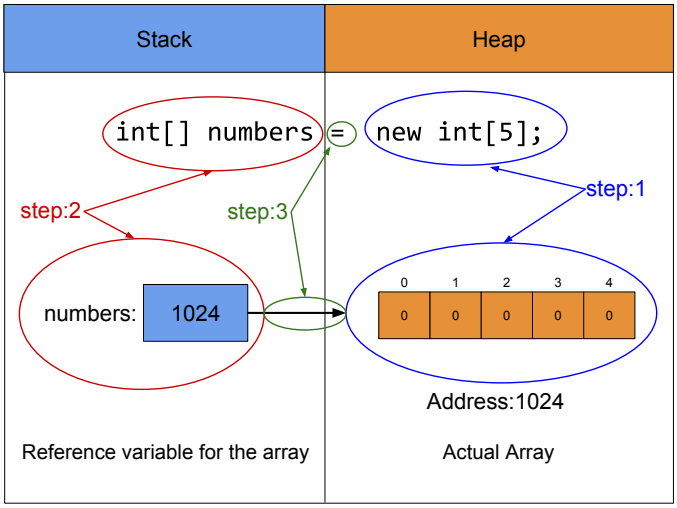

Fig. 1. The step-wise memory diagram for the creation of an array in Java.

The mini-lecture component usually lasted for no more than 30 minutes. It was essential to ensure that all the students had some basic understanding of the programming concept(s) before we wrote some code using those concepts in the next two parts of the class (i.e., live-coding and in-class coding).

C. Live-coding Session

Live-coding [2] 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* [21], [22]. 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 would 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 [23].

The instructor spent around 30 to 40 minutes during every class for live-coding. At the beginning of every coding session, the instructor spent some time to make the students feel comfortable with the syntax of the programming construct that was taught (e.g., while teaching arrays, the instructor wrote code for creating an array in all possible ways in Java). Then, the instructor revisited the problem that was discussed during the start of the lecture (e.g., To find the maximum element from a list of 10 numbers) and discussed one possible (and simple) approach to solve the problem. Immediately after this discussion, the Java code to solve this problem was written in a step-by-step manner by interacting with the students, for each

[1]http://bit.do/pointer_analogy

and every step in the process. Throughout this entire session, the instructor *thought aloud* so that the students would be able to understand the thought process of the instructor while he was writing code. 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) was emphasized throughout the live-coding session. Our live-coding session followed an approach similar to the one explained in Bruhn and Burton's approach for teaching Java using computers [**?**].

During live-coding, sometimes the instructor purposefully introduced some bugs into the program. e.g., Using the assignment operator (=) instead of the equality operator (==). A few times some unexpected bugs showed up in the instructor's code. These scenarios created ample opportunities for the students to observe and learn how an expert would fix these issues using techniques like print statements for simple bugs and tools like the debugger for more complex bugs.

The instructor wrote code by following the indentation guidelines that were required for the course during the live-coding session. Since the students were coding along with the instructor, most of them started indenting their code after every few minutes. Also, in the first few lectures, the instructor wrote descriptive comments for his code, so that the students understood the importance and the usefulness of commenting their code. In later lectures, the instructor wrote very few comments in his code as a measure to save time during the live-coding session.

Whenever the instructor used some methods from the Java library, he explained how to search the Javadoc or Google to find the method that we need. e.g., When the task was to extract some specific part(s) of a String, the instructor opened Google and typed "how to get a substring from a string in java" and taught the students how to make use of Stack Overflow when they program on their own. Also, the usefulness of the Javadoc for understanding the purpose of all the methods in a class was discussed. This approach of learning programming based on the task at hand is termed as Just-In-Time Learning [**?**].

Live-coding session helped to communicate some good coding practices (e.g., debugging, indentation, commenting, reading Javadoc, using Eclipse shortcuts, etc.) to the students in almost every class so that they too used these good coding practices while they worked on their programming assignments and projects.

In the next section, we describe the in-class coding activity that the students do immediately after the live-coding session.

D. In-class Coding

The in-class coding was the time during the lecture where the students were required to write programs on their own, compile, run, and test them for correctness. This gave the opportunity for the students to immediately apply the concepts that they learnt during the mini-lecture and the live-coding components of the class. The students were usually encouraged to discuss with their classmates while they work towards a solution for solving the programming problem. Most of the students worked in pairs during this time while a few of them preferred to work on their own.

The instructor and the teaching assistants (TAs) were present during this time to answer students' questions. The TAs were required to attend only the in-class coding activity component of the lecture (approximately 20 minutes). Three teachers (1 instructor + 2 TAs) were enough for handling all the questions that were asked by the students during this activity time.

The coding problem that was given during this in-class activity time usually required approximately 5 to 10 minutes to complete for an average student. Usually, more than 50% of the students were able to complete the activity within the given time. The in-class coding made many students to realize how difficult it was to write code by themselves even though they may have felt that the concepts discussed during the lecture and the live-coding were pretty straight forward. Usually, the students who were not able to complete the in-class coding activity stay in the classroom for an additional 10 to 15 minutes to complete it although they were not required to do so.

In the last 5 minutes, the instructor discussed a sample solution for the coding activity by actually coding it in front of the students. This discussion was intended to make the students to understand the process needed to come up with the solution. The instructor usually mentioned that his solution was only one of the many possible solutions for solving that problem.

a) A sample in-class coding activity: The following coding activity was given on the day on which arrays were taught:

1) Create an integer array of 100 elements.
2) Initialize the array with random values from 0 to 100.
3) Find the minimum element and its index in the array.
4) If the minimum element appears multiple times in the array, then print the index of its first occurrence only.

E. Student Survey

To understand the students' perceptions on the three instructional practices that we used for teaching introductory programming, we collected a 7-point likert scale survey from the students at the end of the semester. This survey was prepared with the intention to find out the degree to which the students liked/disliked these three instructional practices. The survey was anonymous and 75% of the students completed the survey. The rating scale was defined as: [1 - Extremely Useless, 7 - Extremely Useful].

1) **mini-lectures:** How useful were *mini-lectures* for learning programming?
2) **live-coding:** How useful was *live-coding* for learning programming?
3) **in-class coding:** How useful was *in-class coding* for learning programming?

We also asked the following question to understand the combined effectiveness of these three active learning instructional techniques. The rating scale for this question was: [1 - Not at all, 7 - Completely].

"How much did you like the active learning approach used

in this class when compared to a traditional lecture based class?"

V. Results

In this section, we present the results of our student survey about the three instructional techniques that we used for teaching programming. We also present the survey results about how much students liked the active learning approach used in this course when compared to a traditional lecture based approach.

A. Students' perceptions on three techniques

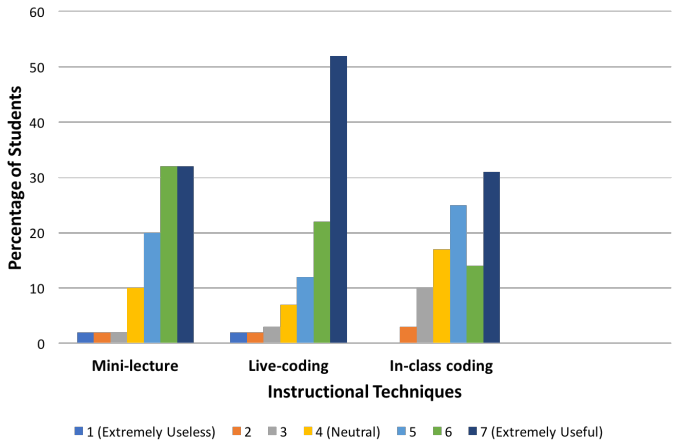

Fig. 2. The students' perceptions on the usefulness of mini-lectures, live-coding, and in-class coding for learning programming.

Figure 2 shows the summary of the students' responses for the three 7-point likert scale questions on the three instructional techniques shown in Section IV-E. The x-axis in this figure represents the three instructional techniques, and the y-axis represents the percentage of student responses for each category in the likert scale questions. The graph shows that live-coding is the technique that is most preferred among the three techniques.

To better compare these three techniques based on students' preferences, we group the responses into the following three categories:

- **Useless:** 1 (Extremely useless) to 3 (Useless)
- **Neutral**: 4 (Neither useless nor useful)
- **Useful:** 5 (Useful) to 7 (Extremely useful)

Figure 3 shows a stack plot of students' responses based on these three categories. From Figure 3, we can observe that *live-coding* is the most preferred and *in-class coding* is the least preferred among the three tehniques.

B. Students' perceptions on active learning

Figure 4 shows the students' responses for the question *"How much did you like the active learning approach used in this class when compared to a traditional lecture based class?"* From the students' responses, we see that 87% of the students have liked the active learning approach when

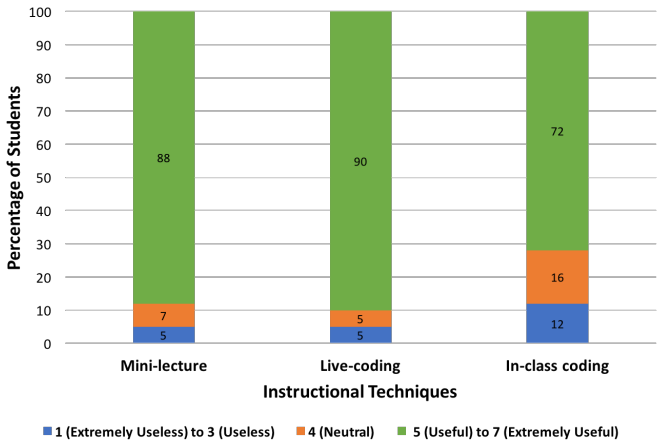

Fig. 3. A stack plot comparing the students' perception on mini-lectures, live-coding, and in-class coding.

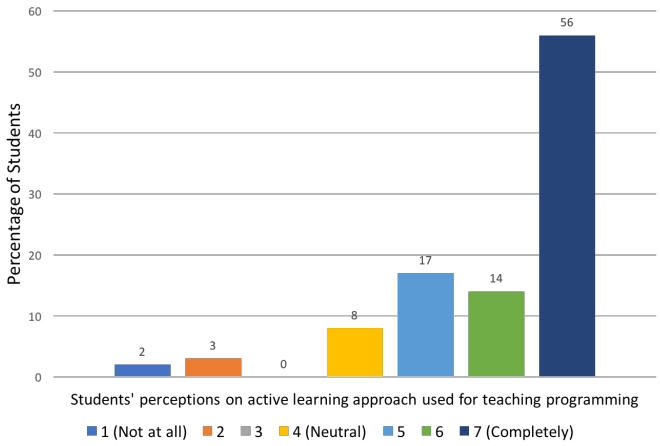

Fig. 4. The students' perceptions on the usefulness of active learning when compared to traditional lectures for learning programming.

compared to a traditional lecture based class for learning introductory programming.

VI. Discussion

In this section, we provide an interpretation of our results, discuss some of the limitations with our work, and provide some directions for future work in this area.

A. Interpretation of results

We found that the students perceive the three components of our course that required varying level of student activity as shown in Table I. The three techniques are ranked based on the amount of student activity required from 1 (least active) to 3 (most active) and also based on students' preferences from 1 (most preferred) to 3 (least preferred).

Based on these results, we found that the students' perceptions of the three instructional techniques that require varying level of student activity is not directly related to the amount of student activity that is required by the technique. In other words, if a technique requires more student activity, then it

does not necessarily mean that it is a better way for learning programming from the students' perspective.

TABLE I
RANKING OF THE THREE TECHNIQUES THAT REQUIRE VARYING LEVELS OF STUDENT ACTIVITY BASED ON STUDENTS' PREFERENCES.

Technique	Student activity	Students' preference
Mini-lecture	1 (least active)	2
Live-coding	2	1 (most preferred)
In-class coding	3 (most active)	3 (least preferred)

The results from Figure 4 suggest that, in general, most of the computing students prefer active learning techniques when compared to traditional lecture based classes. These findings provide evidence that active learning is not only preferred by computing instructors but also by computing students. We acknowledge that our results only suggest that students like active learning that involves mini-lectures, live-coding, and in-class coding. The results may be totally different in an active learning computing classroom that uses other instructional techniques.

We believe that our findings would help computing instructors to choose the instructional techniques to be used in their classrooms not only based on the instructor's preferences but also considering the students' perceptions of these various techniques. By doing so, we would be able to design and develop student-centric courses.

B. Limitations

There were some limitations with our study which we acknowledge here. The students' perceptions of the three instructional techniques in Computer Science reported in our study are from a single group of students who learnt programming from the same instructor. The students' perceptions might change when a different instructor teaches the same course using the same instructional techniques. Therefore students' perceptions from similar computing courses taught by different instructors are needed in the future to really understand what students think about these three instructional techniques for teaching programming.

The survey and the feedback collected were answered only by 75% of the students in the class. We did not make the participation compulsory and the reviews were anonymous since we felt that the student perceptions may be biased, if the reviews were not anonymized. If the 25% of the students who did not voice their opinions had any negative experiences with these instructional strategies, then they were not captured in this study. It would be interesting to find out if the students' perceptions of these three instructional strategies are any different, if all the students in the course shared their opinions.

C. Future Work

Our current work only focuses on students' perceptions of instructional strategies. It is also important to correlate students' peformance with their preferences since even though in-class coding seems to be the least preferred instructional technique by the students as a whole when compared to other techniques, it may well be that the students who preferred in-class coding may be the ones who performed better in the course. So, one of the future work in this area is to correlate students' preferences with their performances to choose the instructional strategies for teaching introductory programming.

Another suggestion for future work is to study the instructors' perceptions on these three active learning instructional techniques and compare them with the students' perceptions. By doing so, we would be able to understand these three instructional techniques from the perspective of both the students and the instructors. From this study, we may also be able to find out why computing instructors prefer certain instructional techniques when compared to others.

VII. CONCLUSION

This work evaluates the students' perceptions of the effectiveness of a trio of instructional practices commonly used for teaching Computer Science. We conclude that students' perceptions on mini-lectures, live-coding, and in-class coding are mostly positive. Analysis of the student survey showed that, majority of the students have reported live-coding and mini-lectures to be most useful for their learning, while in-class coding was not as useful as the other two techniques. We also conclude that, computing students prefer active learning instructional techniques when compared to traditional lecture based classrooms for learning introductory programming.

REFERENCES

[1] Gottfried, Byron S. "*Teaching computer programming effectively using active learning.*" American Society of Engineering Education Annual Conference, June 1997.

[2] Paxton, J. "*Live programming as a lecture technique.*" Journal of Computing Sciences in Colleges, 18(2), 51-56. 2002.

[3] Holliday, M., and David Luginbuhl. "*Using memory diagrams when teaching a Java-based CS1.*" In Proc. of the 41st Annual ACM Southeast Conference, pp. 376-381. 2003.

[4] McConnell, Jeffrey J. "*Active learning and its use in computer science.*" ACM SIGCSE Bulletin 28, no. SI (1996): 52-54.

[5] Cordes, David, and Allen Parrish. "*Active learning in computer science: impacting student behavior.*" In Frontiers in Education, 2002. FIE 2002. 32nd Annual, vol. 1, pp. T2A-T2A. IEEE, 2002.

[6] Cordes, David, and Allen Parrish. "*Active Learning in Technical Courses.*" (1996).

[7] Astrachan, Owen L., Robert C. Duvall, Jeff Forbes, and Susan H. Rodger. "*Active learning in small to large courses.*" In Frontiers in Education, 2002. FIE 2002. 32nd Annual, vol. 1, pp. T2A-T2A. IEEE, 2002.

[8] Smith, Peter D. "*A Process Education Approach To Teaching Computer Science.*" (1996).

[9] Porter, Leo, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. "*A multi-institutional study of peer instruction in introductory computing.*" ACM Inroads 7, no. 2 (2016): 76-81.

[10] Baldwin, Douglas. "*Can we flip non-major programming courses yet?.*" In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 563-568. ACM, 2015.

[11] Machemer, Patricia L., and Pat Crawford. "*Student perceptions of active learning in a large cross-disciplinary classroom.*" Active Learning in Higher Education 8, no. 1 (2007): 9-30.

[12] Lumpkin, Angela, Rebecca M. Achen, and Regan K. Dodd. "*Student perceptions of active learning.*" College Student Journal 49, no. 1 (2015): 121-133.

[13] Smith, C. Veronica, and LeAnn Cardaciotto. "*Is active learning like broccoli? Student perceptions of active learning in large lecture classes.*" Journal of the Scholarship of Teaching and Learning 11, no. 1 (2012): 53-61.

[14] Van Gorp, Mark J., and Scott Grissom. "*An empirical evaluation of using constructive classroom activities to teach introductory programming.*" Computer Science Education 11, no. 3 (2001): 247-260.

[15] Rajaravivarma, Rathika. "*A games-based approach for teaching the introductory programming course.*" ACM SIGCSE Bulletin 37, no. 4 (2005): 98-102.

[16] Simon, Beth, Michael Kohanfars, Jeff Lee, Karen Tamayo, and Quintin Cutts. "*Experience report: peer instruction in introductory computing.*" In Proceedings of the 41st ACM technical symposium on Computer science education, pp. 341-345. ACM, 2010.

[17] Beichner, Robert J., Jeffery M. Saul, David S. Abbott, Jeanne J. Morse, Duane Deardorff, Rhett J. Allain, Scott W. Bonham, Melissa H. Dancy, and John S. Risley. "*The student-centered activities for large enrollment undergraduate programs (SCALE-UP) project.*" Research-based reform of university physics 1, no. 1 (2007): 2-39.

[18] Foriek, Michal, and Monika Steinov. "Metaphors and analogies for teaching algorithms." In Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 15-20. ACM, 2012.

[19] Waguespack Jr, Leslie J. "*Visual metaphors for teaching programming concepts.*" Vol. 21, no. 1. ACM, 1989.

[20] Cao, Yingjun, Leo Porter, and Daniel Zingaro. "*Examining the Value of Analogies in Introductory Computing.*" In Proceedings of the 2016 ACM Conference on International Computing Education Research, pp. 231-239. ACM, 2016.

[21] Holliday, M., and Luginbuhl, David "*Using memory diagrams when teaching a Java-based CS1.*" Proc. of the 41st Annual ACM Southeast Conference, 2003.

[22] Bennedsen, Jens, and Michael E. Caspersen. "*Revealing the programming process.*" In ACM SIGCSE Bulletin, vol. 37, no. 1, pp. 186-190. ACM, 2005.

[23] Gantenbein, Rex E. "Programming as process: a novel approach to teaching programming." In ACM SIGCSE Bulletin, vol. 21, no. 1, pp. 22-26. ACM, 1989.

[24] Rubin, Marc J. "*The effectiveness of live-coding to teach introductory programming.*" Proc. of the 44th ACM technical symposium on Computer Science Education, 2013.

[25] Bruhn, Russel E., and Philip J. Burton. "*An approach to teaching Java using computers.*" ACM SIGCSE Bulletin 35, no. 4 (2003): 94-99.

[26] Boese, Elizabeth. "Just-In-Time learning for the just Google it era." In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 341-345. ACM, 2016.