# Scaffolded Live Coding: A Hybrid Pedagogical Approach for Enhanced Teaching of Coding Skills

Ankur Chattopadhyay
Computer Science Department
Northern Kentucky University
Highland Heights, Kentucky, USA
chattopada1@nku.edu

Drew Ryan
Computer Science Department
University of Wisconsin
Green Bay, Wisconsin, USA
ryandp18@uwgb.edu

James Pockrandt
Computer Science Department
University of Wisconsin
Green Bay, Wisconsin, USA
pockjm06@uwgb.edu

***Abstract* - This Full paper in the Innovative Practice Category describes a novel research study that is based upon an experimental scaffolding of live coding techniques used towards teaching a traditional face to face undergrad computer programming class. It discusses our hybrid pedagogical model, which comprises of a scaffolded collection of our class instructional methods that include fine blend of live coding-based teaching strategies, and traditional lectures. Our combination of live coding styles, as used in this study, consists of the standard live coding technique and our live secure coding demonstrations, which lead to a uniquely blended and integrated live coding approach for teaching coding along with software security concepts. To our knowledge, this is a new research study based upon a hybrid, integrated live coding approach that represents a scaffolding of distinct teaching styles, which combines coding instructions with teaching of secure coding components. We demonstrate an improvised teaching model that enhances the classical live coding pedagogy by adding the live secure coding components, which holistically represent a new variant in a traditional undergrad coding class, for an engaged and enhanced learning experience. Existing literature indicates that there is limited number of prior educational research studies on the usage of a hybrid, non-traditional live secure coding approach blended with the traditional live coding style. Thus, this paper discusses a fresh, nifty teaching strategy that involves new variants of the standard live coding instructional method and forms a hybrid pedagogical model for a more effective and relatable learning of coding skills. We discuss the results of our experimental study in the form of data obtained over a couple of semesters from an upper-level undergrad coding class through learning assessments and collected survey data on learner experiences. We have analyzed the overall gathered learner data to evaluate the performance of our new hybrid model of scaffolded live coding techniques. This paper discusses our overall hybrid pedagogical approach through a scaffolding of traditional and non-traditional live coding styles for teaching coding plus software security topics. It analyzes our obtained learner assessment data plus enhanced student learning experiences, based upon the inclusion of secure code design lessons and code vulnerability hacking plus exploitation demos.**

***Keywords - hybrid teaching pedagogy model, live hacking, scaffolded live-coding, secure coding***

## I. INTRODUCTION

A great deal of research has been done towards improving the student learning experience within computing education. Traditional lectures have been used for years to introduce students to programming concepts through visuals and explanations. However, the nature of computer science aligns inherently well with hands-on, active approaches to teaching. A common way of achieving this is using the technique of live coding within the classroom.

The idea of live coding, or live performance, has been successfully utilized in fine arts and musical performance studies, where live coding refers to the real-time writing or editing of source code to create music, all while in front of a live audience [1, 2, 3]. The topic has become common in research and has seen its popularity rise so much so that there are live coding music festivals held across the globe [3]. An area of computer science educational research stems around the use of live coding in learning environments. The primary focus of this live coding technique is exploring the possible benefits of shifting the class from its more usual environment of listening to a more unique and active environment of doing. These differences, as well as several others, are shown in Table 1.

While live coding by itself is not an unprecedented educational methodology, this study considers the unique strategy of a hybrid teaching model and scaffolded approach to live coding. More specifically, this study aims to blend traditional lectures with live coding sessions leading to live hacking. In order to do so, our hybrid teaching model strives

| Traditional Lecture | Live Coding |
|---|---|
| • Passive (listening, observing, memorizing) | • Active (doing, participating, applying) |
| • Discussion of concepts/syntax | • Application of concepts/syntax |
| • Taught at pace of instructor | • Taught at pace of classroom |
| • Direct knowledge transfer | • Direct and indirect (unintended) knowledge transfer |
| • Delayed feedback / use of learned concepts | • Immediate feedback / use of learned concepts |
| • Mistakes rarely/never encountered | • Mistakes encountered and solved |
| • Impersonal (lacks connection to material) | • Personal (direct connection to material) |
| • Minimal/no collaboration | • High collaboration |
| • Higher structure/routine | • Lower structure/routine |

TABLE I. TRADITIONAL LECTURE VERSUS LIVE CODING

to incorporate important elements of secure coding concepts within traditional coding topics that can be more thoroughly explained and applied through a live coding approach.

### A. Related Works

Existing research literature shows that there is limited work involving educational assessments on the usage of a hybrid, non-traditional live secure coding approach blended with the traditional live coding techniques. Typically, cybersecurity-focused courses revolve around offensive and defensive security techniques, while programming courses generally exclude software security topics altogether. Security is a fundamental piece of any program, though, and software security related knowledge plus skills can be extremely useful and handy for programmers. Our study incorporates elements of secure coding and live hacking centered around the prevention of software security issues, allowing students to gain exposure to core security concepts, like data tampering plus data protection, while working within the confines of a coding course. Hacking has been shown to be engaging and performative, which blends particularly well with the concept of live coding [4]. This is something that traditional, lecture-based coding classes may not cover and touch upon.

The idea of secure coding, and hacking, is often misunderstood. The typical "Hollywood" portrayal of hacking, which is commonly seen in movies and television, often depicts malicious intent and unauthorized access of a system. Hacking can be more realistically exemplified through the unconventional use of software or a system, regardless of the nature of intent. Additionally, the term does not necessarily have to pertain to cybersecurity either, as hacking can also be used to simply represent unorthodox, or outside-the-box, thinking [5]. These definitions were focal centerpieces for the live coding techniques used in this study. The research discussed in this paper is intended for computing educators, especially the ones, who teach coding, as a learning pathway for students to achieve a strong level of comprehension and understanding through a multifaceted blend of teaching strategies, which includes traditional lectures and scaffolded live coding techniques.

## II. METHODOLOGY

### A. The Hybrid Model

Computer programming course lectures in higher education often include a combination of concept explanations plus examination of static code. The goal behind live coding is to teach coding concepts via visual demo by typing, compiling, and testing code examples in class. Live coding has been found to be as good, if not better than teaching with static code [6]. It exemplifies a more active style of learning that students have previously found more useful than traditional lectures [7, 8, 9, 10]. It is considered to provide many pedagogical benefits to the learning process, including unintended knowledge transfer, being a medium through which students learn important details, which were not consciously planned, such as coding styles, debugging, and editor shortcuts [7, 11]. These intricate details can potentially make a major difference to the aspiring developer. There is so much more to computer programming than concepts and code syntax. One of the most significant aspects of programming is problem solving, which needs to be taught effectively. Struggling students tend to jump into the code first, failing to conceptually think about the task at hand, without considering the program design, and using pseudocode [12]. It is conceivable that students, who are subjected to a lecture-only learning model in a coding class, may lose out on these key practical learning components and face similar struggles. The focus of coding lectures is often syntax and theory, but the actual application of the theory is where students have trouble [6, 13].

The concept of continuous learning is also becoming increasingly important in the computing industry. Professionals across the globe are often required to keep up with an ever-changing landscape that expects individuals to remain active as lifelong students, even outside of the formal classroom. Because of this, it is certainly not uncommon for professional developers to self-learn new languages, frameworks, tools, concepts, and paradigms. Surveys have shown that an overwhelming majority of professional developers have used the self-taught method, which is nearly twice as common of an educational method as taking an online course [14, 15]. Students and professionals alike are choosing to turn to YouTube and other online tutorial avenues instead of textbooks, especially among the younger demographic of individuals [15]. With this information known, it seems greatly beneficial for academia members to question why this may be the case, and how can it be utilized to improve the way education is approached, particularly in the field of computer science.

Methods used in prior research studies in existing literature [1, 2, 3, 6, 7, 10, 13, 17, 18, 20, 21] were reference points for our experimental case study, as presented and discussed in this paper. Through our work, we attempt an innovative, hybrid approach that blends traditional lecture with live coding sessions. In a way, our efforts try to replicate the impactful results of online tutorials, students are becoming accustomed to utilizing. These tutorials, whether they come in the form of a blog post or YouTube video, often introduce new programming concepts by considering a problem or task at hand, and then by discussing how that issue can be solved via code. Our hybrid teaching model follows a similar outline throughout this experimental study.

The focus of the traditional lecture portion of our instructional delivery is to provide students with both basic and in-depth understanding of coding concepts plus topics. The lectures created a habitat for discussion, especially with the consideration that most, if not all, topics were new to students. Our lectures used standard practices, such as case studies and content slides, while providing some brief overviews of code snippets and application examples.

The live coding sessions were designed as hands-on workshops, intended at reiterating the concepts covered throughout the lectures in real-time. New programming concepts can often be difficult for students to grasp if not worked through. Keeping that in mind, specific coding applications and modules on newly covered topics were taught with the goal of effectively tying together the different learning approaches together. Students were often presented with a specific programming problem or project during a class session, and students were tasked to code a

working solution with the aid of a peer lead. Additionally, students followed along with the instructor to write, run, and test interactive code. Examples of two such programs are shown in Figures 1 and 2, which are code snippets that we used in class to demonstrate the concepts of variable scope and smart pointers, respectively.

```
/*
SCOPE CASE STUDY 1
-Can we declare "x" again inside this if statement?
-What prints first inside this statement?
-What prints second?
-What prints third, outside the statement?
*/

int x = 5;

if (x > 0) {
    std::cout << x << std::endl;

    int x = 4;

    std::cout << x << std::endl;
}

std::cout << x << std::endl;
```

Fig. 1. Example of A Live-Coding Case Study On Variable Scope

```
/*
SMART POINTER CASE STUDY
*/

if (true) {
    // this pointer persists outside this statement/scope
    // (need to add "delete vehicle;" to prevent this)
    Vehicle *vehicle = new Vehicle();
}

if (true) {
    // this smart pointer is auto-destroyed at end of this statement
    unique_ptr<Vehicle> vehicleSmartPtr(new Vehicle());
}
```

Fig. 2. Example of Live-Coding Case Study on Smart Pointers

We performed our experimental study with students of an upper-level coding course over two semesters with a total of 50 students (class size per semester). Our live coding examples allow learners to witness the coding process and interpret how the code works plus behaves when run. The instructor and students run code examples and go through them together with a debugger, allowing students to see the whole process in real-time, and find out whether their expectations were correct. These live coding case studies are difficult to replicate via lecture slides or static code snippets. Our course lectures and the live coding sessions focused on important C++ topics plus coding concepts, including conditions, loops, data types, inputs, outputs, pointers, memory management, and graphical user interfaces. Additionally, we covered core object-oriented programming topics, including encapsulation, polymorphism, and inheritance.

*B. Scaffolded Live Coding*

In addition to our hybrid teaching model, our study also introduces a unique approach of scaffolding to the experimental live coding sessions. To utilize a multi-dimensional pedagogy, our sessions included live secure coding and live hacking techniques, as depicted in Figure 3.

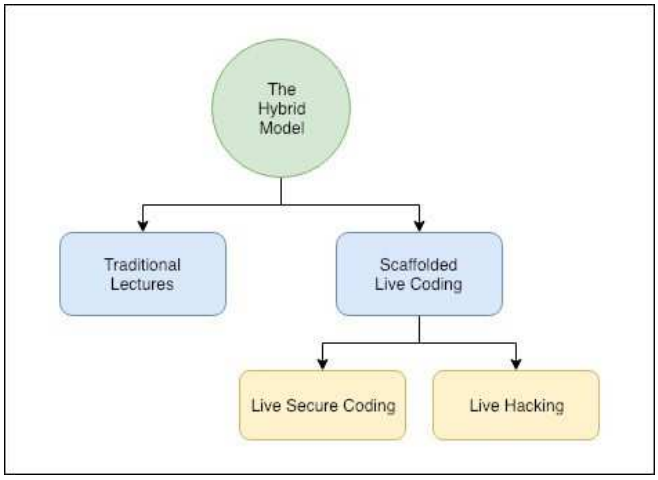


Fig. 3. The Hybrid Learning Model with Scaffolded Live Coding

Secure coding modules were constructed to expose students to a number of important s o f t w a r e development related security concepts, including input validation, buffer overflow, and integer error. These topics tie in well with live hacking case studies that were formulated to engage students and improve their overall learning experience.

```
/*
INPUT BUFFER MISBEHAVIOR CASE STUDY
*/

string name = "";
int age;
string major = "";
string homeTown = "";

cout << "Please, input your name: " << endl;
getline(cin, name);
cin.clear();
cin.ignore(2000, '\n');

cout << "Please, input your age: " << endl;
cin >> age;
cin.clear();
cin.ignore(2000, '\n');

cout << "Please, input you major: " << endl;
getline(cin, major);

cout << "Please, input you hometown: " << endl;
getline(cin, homeTown);

cout << "Name: " << name << end;
cout << "Age: " << age << endl;
cout << "Major: " << major << end;
cout << "Hometown: " << homeTown << endl;
```

Fig. 4. Input Buffer Misbehavior Case Study

An additional goal of scaffolded live coding is simply to ensure the students remain interested and have fun. While live coding activities are already considered an improvement on a traditional lecture, the extra blend of secure coding and hacking tries to push these benefits even further. We observed that the students enjoyed these finely blended mix of hands-on activities, leading to greater engagement and understanding, which overall motivates learners to continue studying the coding topics [16]. One of the secure

Coding lab modules, which we designed for the students, focused on the theme of input buffer misbehavior.

Our live coding sessions did adapt and utilize secure coding lessons and questions from the Towson University's 'Security Injections' project lab modules [17]. Additionally, a program was written to help students practice, test, and understand the different options available for collecting input within C++. Students were taught how to properly handle input and avoid common buffer issues that can pose serious security risks within a given program. Students were also tasked with running the code, observing the initial outcomes, manipulating the code to change its outcomes, and see if they can apply different techniques to solve the vulnerability issues with the program. A sample of this program is shown in Figure 4.

Other topics covered in our secure coding lessons include integer error, data tampering, and exception handling. For instance, one code example on data tampering (refer to Figure 5), demonstrates to students the potential danger of passing variables by reference without utilizing the ability of the constant ("const") keyword. With guidance from the course instructor and the peer team leads, students were able to create this code and then run and debug multiple test cases in real-time.

```
/*
DATA TAMPERING CASE STUDY
-Here, we see what happens when a variable is passed by reference
and changed inside of another function.

-Try to prevent this by making the variable a const.
-Do you see the value in using constants?
*/

void passByReference(int& num) {
        hackInt = 987654321;
        std::cout << "Your int is mine now." << std::endl;
        num = hackInt;
};
```

Fig. 5. Data-Tampering Example - Dangers of Passing by Reference

```
/*
HACKING CASE STUDY: Disk Jam
-Uses "dir" system command to get file directory contents
-Stores the output to a .txt file (could be any file, such as .exe)
-An infinite loop can make this output grow indefinitely and jam the disk
*/
for (int i = 0; i < 10; i++) {
    system("dir >> TestFile.txt");
}
```

Fig. 6. A Simple Hacking Case Study - Disk Jam Example

Some sessions were more closely related with hacking and sought to branch out into unique examples that students would find interesting and thought-provoking. These case studies ranged from the simple, traditional issues of an infinite loop to more complex, classic examples of software security. One such program included how to utilize a system command to write into the memory program registry and disable system functionalities. Another example, which can be seen in Figure 6, showcased how to repeatedly overwrite file contents, causing an eventual disk jam.

### *C. Peer Mentoring and Instruction*

Beyond the hybrid model and scaffolded techniques, this study also utilized a multi-layered peer-led mentoring approach. The traditional lecture portion of class was taught by the course instructor, while three teaching assistants led the coding and hacking sessions. Additionally, there were peer mentors within the class, who were group/team leads for some of the course activities.

This approach perfectly suited the nature of the live coding sessions, which were designed and developed to be a low-pressure and light-stakes setting for guided learning. In a peer-to-peer environment, students can openly learn, work, and discuss with less worry for embarrassment and less anxiety over the knowledge gap that can often exist between the instructor, teaching assistants and students. This also helped to foster an environment conducive to active collaboration, which is important, because professional developers often spend most of their time working in teams [18]. Additionally, students typically improve their academic performance when learning with and from peers [19, 20, 21].

### *D. Split-Class Study*

Another aspect of our experimental setup is that we did additional experimentation to collect data through the adaption of different teaching methodologies, using a split-class study. This approach divided the classroom into two student groups, both randomly assigned. The first group attended a traditional lecture session, while the second group attended a live coding session. Both sessions were focused specifically on the concept of C++ operator overloading. Like all other class periods, these two sessions utilized the same mixed-instruction approach, as the lecture portion was taught by the instructor and the live coding parts were led by teaching assistants. Students participated in both pre- and post-session surveys, as well as quizzes on covered topics.

Another example of a program designed and developed for one of our live coding sessions is shown in Figure 7. The students, led by the class peer mentors, created custom code from scratch that used operator overloading to manipulate the balance of bank accounts - an example that well suits the usefulness of the topic at hand, and the live demo theme.

```
/*
OPERATOR OVERLOADING EXAMPLES
*/

// create bank accounts
BankAccount account1(250.00);
BankAccount account2(500.00);
BankAccount account3(0.00);

// display initial account balances
std::cout << "Account 1: $" << account1.getBalance() << std::endl;
std::cout << "Account 2: $" << account2.getBalance() << std::endl;
std::cout << "Account 3: $" << account3.getBalance() << std::endl;

// operator overload case 1 (+ operator)
account3 = account1 + account2;
std::cout << "Account 3: $" << account3.getBalance() << std::endl;

// operator overload case 2 (< operator)
if (account1 < account2) {
    std::cout << "Account 1 is less." << std::endl;
} else {
    std::cout << "Account 2 is less." << std::endl;
}

// operator overload case 3 (= operator)
account1 = account3;
std::cout << "Account 1: $" << account1.getBalance() << std::endl;

// operator overload case 4 (++ operator)
++account1;
std::cout << "Account 1: $" << account1.getBalance() << std::endl;
```

Fig. 7. Example of Live-Coded Operator Overloading Program

After the described split sessions were completed and students finished the survey participation, the two groups were switched. This was not done for research purposes, but rather to allow each student the opportunity to get the full experience (both lecture and live coding) for the topic taught, thus ensuring that all bases were covered.

### *E. Student Base*

While this study was performed in an in-person upper-level computer programming course, the student base consisted of individuals of varied coding skills and at different knowledge levels, plus learning capacities. The primary focus of the course was advanced object-oriented software design, specifically using C++. Therefore, the classroom included students, who had little programming knowledge, other than introductory work with the Java programming language. These students had little-to-no experience with the Object-Oriented Programming (OOP) paradigm, or its concepts. Other individuals within the classroom did have some exposure to OOP, albeit with the Java language. Most students in this study did not have prior knowledge or experience with C++. Individuals within the student base held varying levels of understanding in major coding concepts, as well as limited exposure to programming languages. Hence, our study offers unique perspectives and potentially meaningful insights to aid prospective pedagogical methodology for an effective form of differentiated learning.

### *F. Engagement and Feedback Techniques*

Among all facets of our course proceedings, students were persistently encouraged to embrace discussion and collaboration. Live coding case studies were formulated with the intent of demystifying programming related granularity and the involved problem-solving steps. This allowed the class peer mentors to focus on valuable lessons within coding, including how to work through projects step-by-step, how to approach the design of a program, how to avoid common programming pitfalls, and how to successfully debug a program when issues, including security ones, arise.

Throughout the course, students were continually instructed to provide all types of feedback, regardless of medium, to facilitate the collection of learner assessment data. Students were asked to participate in multiple surveys at varying parts of the semester. All these measures helped our goals, which included successful implementation of our hybrid teaching model and determination of its effectiveness.

Additionally, our targeted learner assessment data was gathered through hosting of Kahoot-based quizzes, which allowed the class instructor and the teaching assistants to create custom questions on covered topics and offer them online in a classroom setting via the Kahoot website or a mobile app. The quizzes provide a fun way to engage students in a competition-like environment, where participants gained points for not only choosing the correct answer, but also for fast answers. One such quiz on C++ operator overloading, which was used in the split-class study, was meant to gauge the overall understanding levels.

## III. Data And Results

### *A. Quantitative Data*

An initial survey was provided to students at the start of the semester, asking how they preferred to learn. Participants were asked to choose a number on a scale of 0-10, with 0 representing a more traditional lecture approach, and 10 representing a hands-on, live coding approach. Results of the survey are displayed in Figure 8. No student responded lower than 5, which means that all students in this sample preferred to learn with at least a mix of the two teaching styles. Most responses were 9 or 10, accounting for over 55 percent of the participants. This suggests that students overwhelmingly opted for more active, hands-on learning.

Students were also asked to participate in pre and post surveys during the split-class study on operator overloading. Both surveys included a question that asked the individual to rate their overall knowledge of C++ operator overloading on a scale of 1-5. It can be observed that the average student rating increased after both sessions; however, the differences were surprising. The overall knowledge rating showed a slight, small increase for students participating in the traditional lecture, while the corresponding rating more than doubled for students in the live coding session; although, it is notable that the students in the live coding sessions initially rated their knowledge quite lower than the lecture session student cohorts. These results are seen in Figure 9.

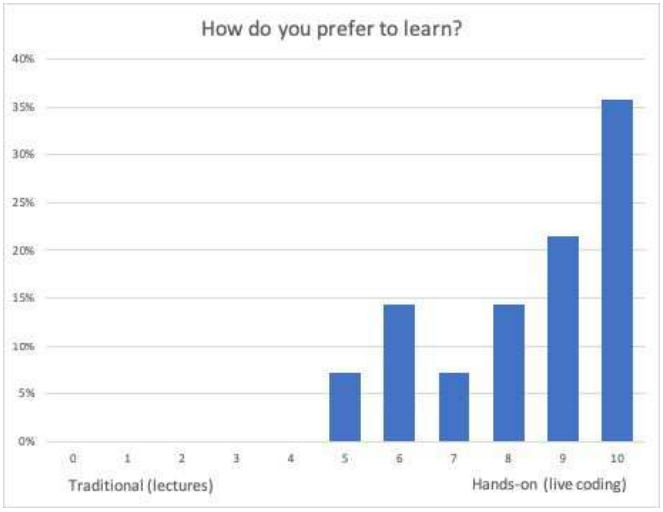


Fig. 8. Student Survey Data: Traditional Lectures Versus Live Coding

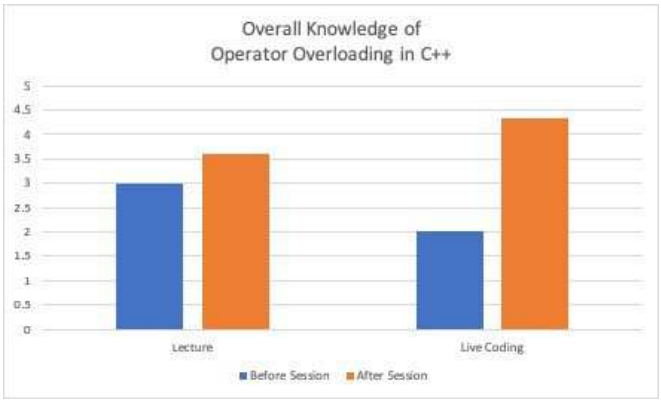


Fig. 9. Split-Class Study Survey Data: Knowledge of Operator Overload

In the post-session surveys, students were asked to rate the teaching style or content-delivery method they experienced. For students that participated in the traditional lecture sessions, the average overall rating was 3.6. The live coding sessions, on the other hand, received a rating of about 4.3 from student participants. These results are in Figure 10.

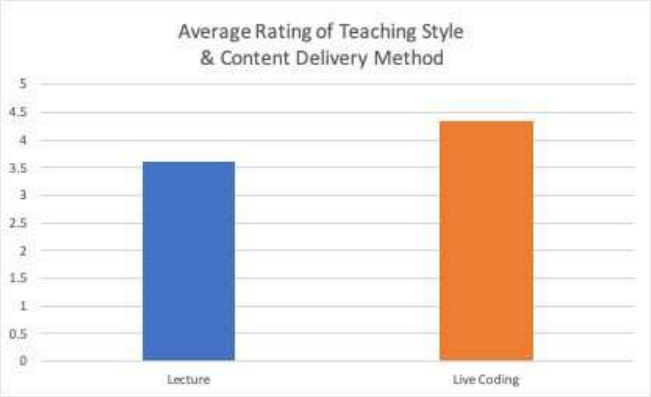


Fig. 10. Split-Class Study Survey: Average Rating of Teaching Styles

As a final assessment of the split-class study, students participated in a short, 10-question Kahoot quiz based on the topic of operator overloading. This was mainly used to assess how well the learners learned the subject. A sample of the quiz questions, which includes a mix of true-or-false and multiple choice, questions is shown in Figure 11.

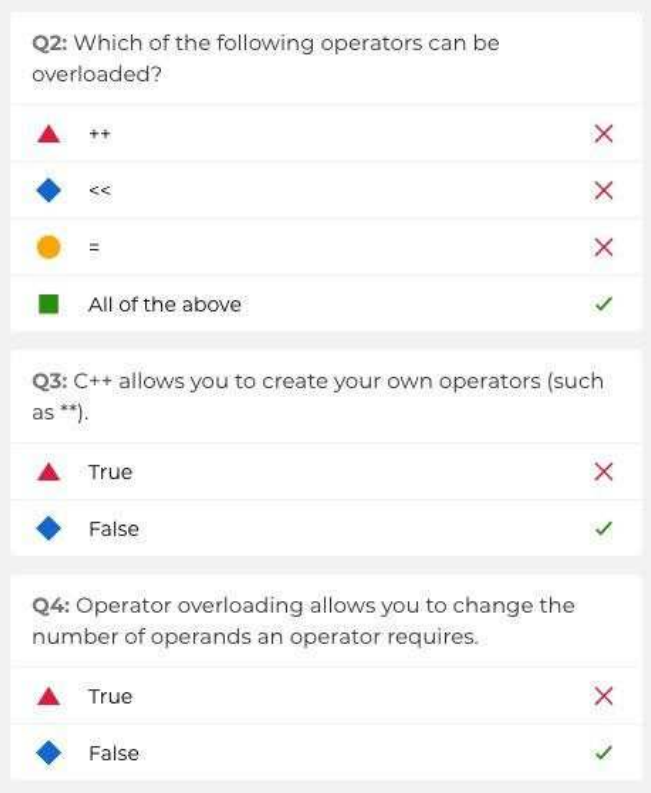


Fig. 11. Sample of Operator Overloading Questions from Kahoot Quiz

Overall, students performed well on the listed Kahoot quiz, with 78 percent of the responses being correct answers (as exhibited in Figure 12). One noteworthy thing to consider when reviewing these results is that Kahoot brings a somewhat different dynamic to the typical quiz, as students gain additional points for the speed/timeliness of their answers. Because of this time limit, some students admitted that they chose the wrong answer simply because they failed to read the answer options fully before making their selection. A perfect example of this is Question 2, which is shown inn Figure 11. About 64 percent of students answered the question correctly, with some of the incorrect responses likely being due to the time constraint. With that kept in mind, the overall student performances may have been higher in a different quiz environment other than Kahoot.

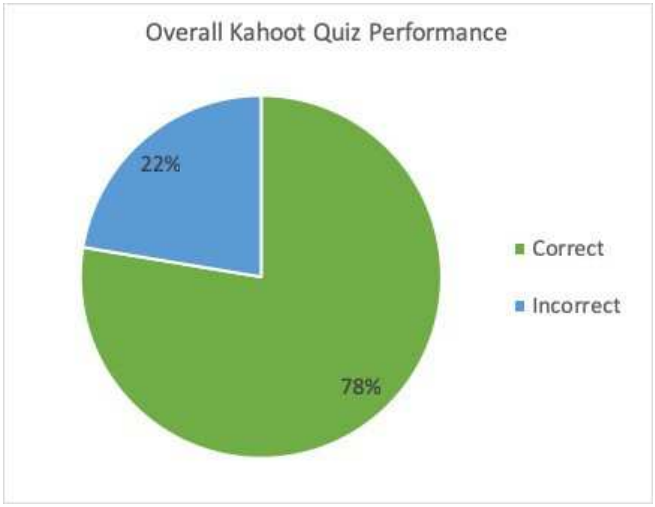


Fig. 12. Overall Performance of Kahoot Quiz

In addition to the Kahoot questions, we also asked the student participants two survey questions in regard to our hybrid teaching approach. Results of these inquiries can be seen in Figures 13 and 14. The first survey-style question asked students if the hybrid approach was effective for learning operator overloading, to which students could respond *Yes*, *Maybe*, or *No*. Most students felt the pedagogy was effective, with 69 percent answering *Yes*, 23 percent claiming *Maybe*, and only 8 percent answering *No*. The second survey question simply asked whether the individual prefers a combination of lectures and live coding when learning programming. Results did not include any *No* (negative) responses, with 85 percent of students answering *Yes* (the remaining 15-percent answering *Maybe*).

Looking beyond the results, the Kahoot quiz appeared to naturally bring out the excitement and engagement of the students. From the instructor point-of-view, it seemed definitively clear that students were having fun, and this is confirmed by the overwhelmingly positive responses to the added feedback questions we asked after the Kahoot quizzes. The first additional survey question asked participants to rate how fun the sessions were on a scale of 1-5. Responses came back as a perfect 5. The other survey

question asked students if they would recommend the sessions to others, to which 100 percent answered *Yes*.

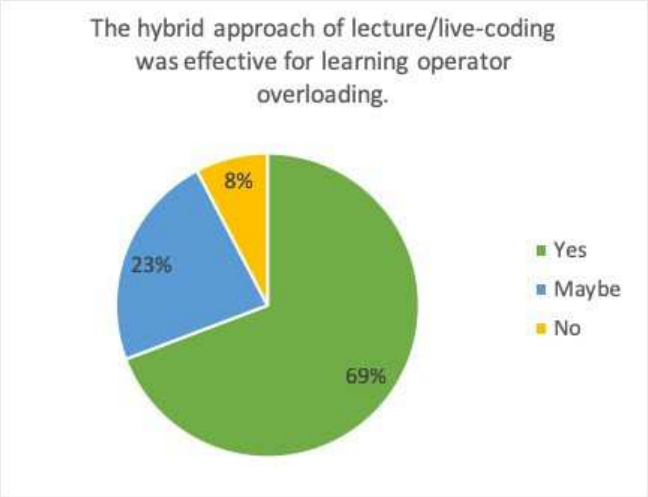


Fig. 13. Example 1 Survey Question with Overall Responses

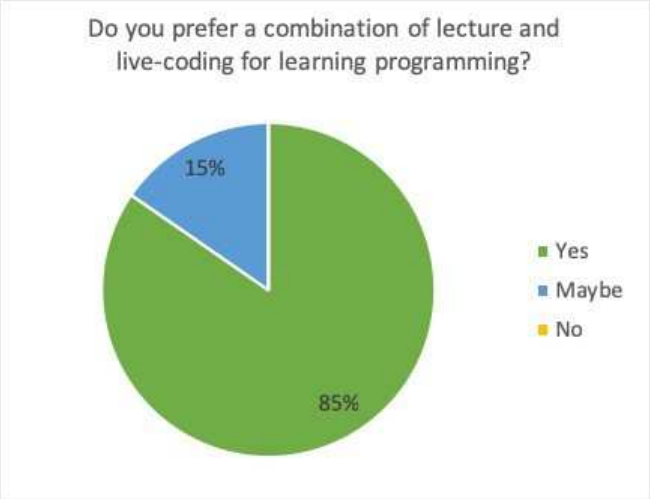


Fig. 14. Example 2 Survey Question with Overall Responses

### *B. Qualitative Data*

Student participation and engagement was overall quite positive, as indicated in the results obtained from our study. Class attendance was consistent for different types of study sessions. Students showed strong levels of attention and active involvement, especially during live coding and live hacking sessions, which we had anticipated.

In the post-session survey for the split-class experiments, a few qualitative questions were included. One such question asked participants to explain what they liked about the sessions. A student from the live coding group mentioned that they enjoyed "coding along instead of just listening," and went on to add that it provided a "faster understanding" and that it was helpful to "see how to actually use a tool or idea". Other participants of the live coding session stated that they liked the "hands-on element" and the provided explanation of the material. Students from the traditional lecture group enjoyed "the details" and how well the topic was explained. One student from the session added an interesting note that they "liked the smaller group," as it "seemed easier to get answers to class questions asked."

The students were asked as to what they disliked about the sessions, and the live coding participants did not have anything to share, other than one student comment about having more "debugging in class." On the other hand, students in the traditional lecture sessions noted that they "like hands on stuff and would have appreciated some." One student was even clear in stating that they "learn better when (they) get hands on and do it in person." All these obtained user data are encouraging, and the potential promise of our hybrid teaching model is evident from them.

### *C. Limitations*

While our study can provide beneficial insights to the effectiveness of our hybrid pedagogical approach, it does also point to some of its limitations. Most notably, the study and its data are collected over only a couple of semesters and taken from students of one single upper-level course offering, with survey participation among those students being around 70 percent. These reported results can certainly look different with a larger sample size of students.

Additionally, the opinions of students (as manifested through the survey responses), regardless of their educational backgrounds, or knowledge levels, can be impacted by their relationship to the instructor. To gain a greater understanding of the impact of our hybrid teaching model and its effectiveness, we need to examine students across multiple courses, which adopt our scaffolded live coding-based hybrid pedagogical approach but are taught separately by different instructors.

## IV. Conclusion

This study aims to utilize the many known positive outcomes of live coding to enhance teaching within an undergrad CS2 level computer programming class by blending live coding techniques with engaging and intriguing secure coding live demos in an in-person classroom. Although there is more research work to be done with this hybrid pedagogical model, the results obtained through this preliminary experimental study indicates that our scaffolded live coding approach can be extremely useful in teaching coding lessons, as well as making a difference in student learning plus engagement. We found that our approach can be adopted for reinforced learning of concepts initially taught via a traditional-style lecture.

Additionally, adding layers of secure coding and live hacking demos (code vulnerability exploitation examples), through our hybrid approach, enables instructional delivery of a well-rounded, multidimensional learning experience for students, which appears to be helpful in terms of classroom engagement and enhanced learning. The added emphasis on secure coding concepts (software security topic components) also provides benefits to students, who aim to become versatile and successful computer programmers.

We observe that the use of the peer-mentor strategy to implement the hybrid teaching model allowed students to operate in a more relaxed environment that facilitated active learning through engaged discussion, higher participation, and improved performance.

However, we do realize that there our research results and collected data from these initial experiments are preliminary in nature. Moreover, the scope of our current research work is limited to an extent. For instance, in the context of the split-class study portion within our experiments, we have not accounted for some of the following research agenda items, which could form potential directions for future studies:

- Are there any possible threats to the validity of our findings, including experimental results, in terms of the specific concept (like operator overloading) being taught?

- Comparative analysis in terms of difficulty (or challenge) level of mastery between different computer programming concepts, based upon several trials using our approach

- Comparative analysis in terms of performances and conceptual learning levels between different split groups and how the grouping strategy affects (or impacts) the learning?

Thus, there are multiple avenues of extending our research study in future, as well as building upon our present work. We hope to carry out more rigorous experiments for examining more closely the use of our hybrid teaching model, including our scaffolded live coding techniques, in an effort uncover additional evidence that reaffirms the merits of our approach. However, the preliminary evidence obtained through our initial investigation forms a good starting foundation and a positive reference for adoption of our approach resulting in expanded future studies. Our overall observations indicate that students can be offered a level of both enhanced learning and engagement through the usage of our multi-layered blend of traditional lectures and scaffolded live coding techniques.

## REFERENCES


[1] Collins, Nick, et al. "Live coding in laptop performance." *Organised sound* vol. 8, no. 3, pp. 321-330, 2003. [Online]. Available: https://www.cambridge.org/core/journals/organised-sound/article/live-coding-in-laptop-performance/08F42B84BBCA427C345030481A3DDA0D

[2] Brown, Andrew R., and Andrew Sorensen. "Interacting with generative music through live coding." *Contemporary Music Review* vol. 28, no. 1, pp. 17-29, 2009. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/07494460802663991

[3] McLean, Alex. "Making programming languages to dance to: live coding with tidal." *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design*. ACM, 2014. [Online]. Available: https://dl.acm.org/citation.cfm?id=2633647

[4] A. Hindle, "Hacking NIMEs." 2016. [Online]. Available: https://pdfs.semanticscholar.org/8b45/ee328633899f86e60221065982a9dd945700.pdf

[5] S. Oualline, *C++ Hacker's Guide*. 2008.

[6] Rubin, Marc J. "The effectiveness of live-coding to teach introductory programming." *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013. [Online]. Available: https://dl.acm.org/citation.cfm?id=2445388

[7] Raj, Adalbert Gerald Soosai, et al. "Role of Live-coding in Learning Introductory Programming." *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. ACM, 2018. [Online]. Available: https://dl.acm.org/citation.cfm?id=3279725

[8] Raj, Adalbert Gerald Soosai, Jignesh M. Patel, and Richard Halverson. "Is More Active Always Better for Teaching Introductory Programming." *Learning and Teaching in Computing and Engineering (LaTICE)* 2018. [Online]. Available: http://pages.cs.wisc.edu/~gerald/papers/IsMoreActiveAlwaysBetter.pdf

[9] Matthíasdóttir, Ásrún. "How to teach programming languages to novice students? Lecturing or not." *International Conference on Computer Systems and Technologies-CompSysTech*. vol. 6. 2006. [Online]. Available: https://www.researchgate.net/profile/Asrun_Matthiasdottir/publication/251812193_How_to_teach_programming_languages_to_novice_students_Lecturing_or_not/links/548990140cf268d28f0afce4.pdf

[10] Paxton, John. "Live programming as a lecture technique." *Journal of Computing Sciences in Colleges* vol. 18, no. 2, pp. 51-56, 2002. [Online]. Available: https://dl.acm.org/citation.cfm?id=771332

[11] Brown, Neil CC, and Greg Wilson. "Ten quick tips for teaching programming." *PLoS computational biology* vol. 14, no. 4, 2018. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006023

[12] Govender, Irene, and Diane Grayson. "Learning to program and learning to teach programming: A closer look." *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education (AACE), 2006. [Online]. Available: https://www.learntechlib.org/p/23232/

[13] Alam, Ekky Novriza, Soni Fajar Surya Gumilang, and Muhammad Azani Hasibuan. "Design And Development of Programming Learning Platform Based On Heuristic Approach In Live Code Module With Iterative And Incremental Method (case Study: Information System Telkom University)." *eProceedings of Engineering* vol. 2, no. 2, 2015. [Online]. Available: https://libraryeproceeding.telkomuniversity.ac.id/index.php/engineering/article/view/2205

[14] "Stack Overflow Developer Survey 2018," Stack Overflow. [Online]. Available:https://insights.stackoverflow.com/survey/2018

[15] "2018 Developer Skills Report" HackerRank. [Online]. Available: https://research.hackerrank.com/developer-skills/2018/

[16] Noone, Mark, and Aidan Mooney. "Visual and textual programming languages: a systematic review of the literature." *Journal of Computers in Education* vol. 5, no. 2, pp.149-174.

[17] Chattopadhyay, Ankur. "Beware of input buffer misbehavior and make your code behave: a nifty hands-on assignment on secure coding at the CS0 and CS1 levels: nifty assignment." *Journal of Computing Sciences in Colleges* vol. 30, no. 4, pp. 118-118, 2015. [Online]. Available: http://cis1.towson.edu/~cssecinj/modules/other-modules/build-the-lab/build-a-lab-beware-of-input-buffer-misbehavior-make-your-code-behave-cs1/

[18] Vessey, Iris, and Ajay Paul Sravanapudi. "CASE tools as collaborative support technologies." *Communications of the ACM* vol. 38, no. 1, pp. 83-95, 1995. [Online]. Available: https://dl.acm.org/citation.cfm?id=204882

[19] Herbsleb, James D., et al. "Distance, dependencies, and delay in a global collaboration." *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000. [Online]. Available: https://dl.acm.org/citation.cfm?id=359003

[20] Williams, Laurie A., and Robert R. Kessler. "Experiments with industry's "pair-programming" model in the computer science classroom." *Computer Science Education* vol. 11, no. 1, pp. 7-20, 2001. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1076/csed.11.1.7.3846

[21] Ghorashi, Soroush, and Carlos Jensen. "Integrating Collaborative and Live Coding for Distance Education." *Computer vol.* 50, no. 5, pp. 27-35, 2017.