Collaborative Classroom

A tool for personalized note-taking and feedback in Live Programming Courses

Aishwarya Venkatesh
Department of CSE
PES University
Bangalore, India
aishwarya.2002@gmail.com

Yamini Agarwal
Department of CSE
PES University
Bangalore, India
yaminiagarwal09@gmail.com

Yash Patil
Department of CSE
PES University
Bangalore, India
yash.r.patil98@gmail.com

Vidhu Rojit
Department of CSE
PES University
Bangalore, India
vidhu.tjit@gmail.com

Viraj Kumar
Department of CSE
Dayananda Sagar University
Bangalore, India
viraj-cse@dsu.edu.in

1 CONTEXT

The COVID-19 pandemic has necessitated a rapid shift to online teaching. One of the many new challenges that teachers and students have faced is the paucity of internet bandwidth. Often, this degrades or entirely prohibits the quality of video streaming. For lectures in programming courses where instructors use "live programming" (defined as "the process of designing and implementing a [coding] project in front of class during lecture period" [1]), an alternative to video streaming is a shared programming environment that allows students to view a live, read-only copy of the instructor's code while listening to the instructor. This approach only requires audio-only (as opposed to video) streaming as well as negligible overhead to share textual updates to the programming environment. We propose *Collaborative Classroom*, a tool based on this approach, which also includes a novel feature: personalized note-taking for learners. We also conduct a preliminary investigation of our tool's feasibility as a low-bandwidth alternative to video streaming in programming-intensive courses and to assess the utility of personalized note-taking.

Existing Tools. Many tools support live programming in classrooms. One such tool is Visual Studio Code (often abbreviated as VSCode, https://code.visualstudio.com), a full-fledged Integrated Development Environment (IDE) with a "live share" mechanism that permits a programmer to share their coding environment so that others can simultaneously view their changes and collaboratively edit the code. However, the number of simultaneous users is capped at 30, which is smaller than typical class sizes in India [2]. A much larger number of collaborators can be supported with web-based code-sharing platforms such as Codeshare (codeshare.io), CodeTogether (codetogether.com), CryptPad (cryptpad.fr), Floobits (floobits.com), GitDuck (gitduck.com), and Snipper (snipper.io).

Many of these tools are witnessing a surge in usage due to the COVID-19 pandemic, and new features are being added in response to user requests. Nevertheless, to the best of our knowledge, none of the available tools provide learners with mechanisms for *personalized note-taking* and *personalized instructor feedback*. We describe these novel features in detail in Section 2, and we explain why they are potentially valuable for learners. We describe the technical details of our tool in Section 3, and then describe a preliminary study to evaluate our tool with this feature in a live programming context in Section 4. Finally, we present our conclusions in Section 5.

2 PERSONALIZED FEATURES

The first three authors have experienced live programming as students in a traditional classroom, and they share two key insights. First, even when instructors do their best to articulate their thought-processes out loud as they write code, students may have questions about specific lines or regions of code. Second, students often find it helpful to make notes on key points highlighted by the instructor, either during the initial articulation, or during subsequent questions-and-answers. In both these cases, it is helpful for students to personally annotate a copy of the code with their doubts and comments. In a traditional classroom where students only have notebooks, they must write down enough of the code to maintain context as well as their annotations. With online code-sharing tools, the burden on students can be potentially reduced since they have ready access to the code.

Unfortunately, existing code-sharing tools allow students to either have **full editing** rights or **read-only** access to the code, neither of which is suitable for *personalized note-taking*. With full editing rights, the notes taken by one student are automatically shared with the instructor and all other students, which can be confusing. With read-only access, students must first copy-paste the code elsewhere

before they can edit it. Not only is this cumbersome, it diverts student attention from the instructor's exposition at potentially critical junctures.

Our tool has two features that directly address this issue. First, the *Student Interface* of our tool (Figure 1) has certain elements directly under the instructor's control, including the set of files under discussion (our tool allows the program under discussion to be split across multiple files, unlike many of the code-sharing tools mentioned earlier) and the code view. Only the instructor can modify the code, and these modifications are immediately reflected to all students. Crucially, the *Student Interface* also has certain elements that are under the direct control of students. These include the note-taking features (triggered by the *Add Note* button on the top right of the screen), the ability to *Run* the given code at any time and observe the output or syntax errors that it generates, and the ability to switch to any other file among the instructor-specified list of files, independent of where the instructor is within the code base (the instructor's location is by "Follow cursor at <*filename*>: line <*number*>" in Figure 1.) We believe that it is important for students to move within the code base at their own pace, while also having a quick way to "sync" with the instructor when they desire. This is particularly important while taking notes since some students may wish to write detailed notes. These notes are associated to line numbers in specific files: the "1" within the blue circle in Figure 1 indicates that this note is associated with line 1 of **File.py**. The platform allows for these annotations to be added to the code files as comments and downloaded at the end of the session.

Second, our tool allows students to highlight particular regions of the code that they are struggling to understand and then click on *Send Feedback* (Figure 2). Our tool collates this student-specific feedback into a combined instructor view (Figure 3) which shows regions of the code where students are struggling (the *x*-axis represents line numbers and the *y*-axis represents the number of students who have highlighted that line number).

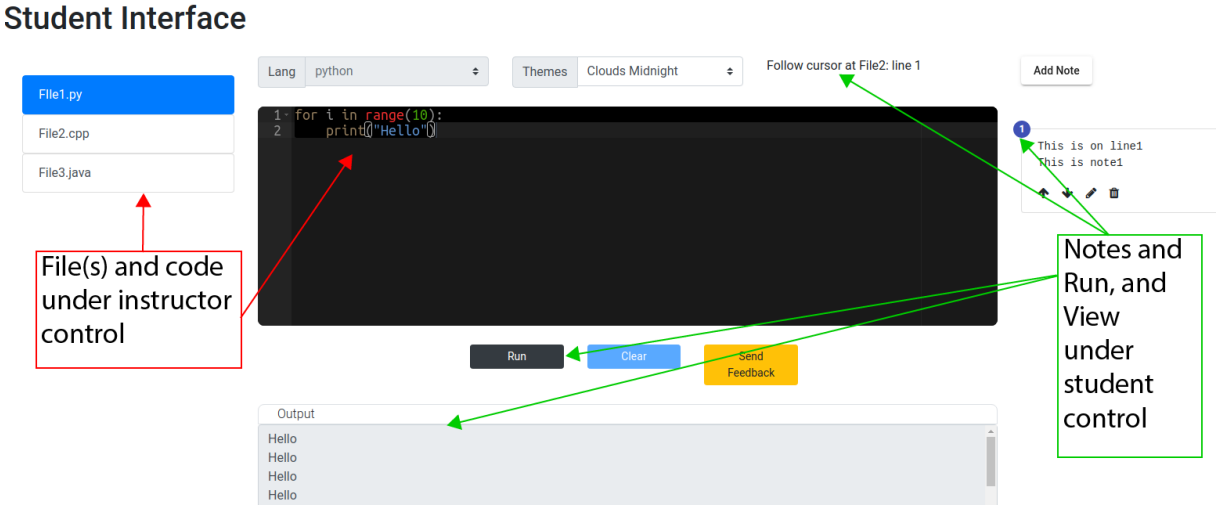

Figure 1. A screenshot of our tool's Student Interface showing elements under Instructor and Student control. Note that the Student is completing a note associated with line 1 in File1.py and reviewing the Output, whereas the Instructor's cursor has advanced to line 1 in File2.cpp

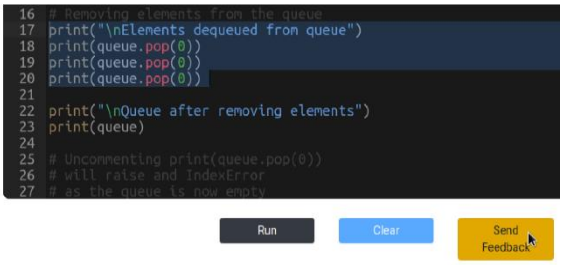

Figure 2. A student has highlighted lines 16 to 19 and is about to click on *Send Feedback* to request further explanation on this region of the code.

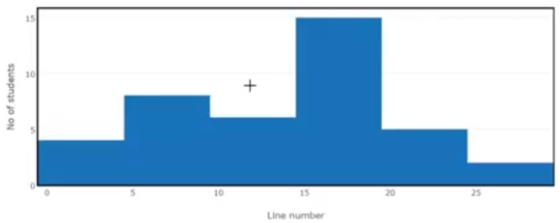

Figure 3. Instructor's summary of student Feedback indicating (in this instance) that many students are unclear about lines 15 to 19.

3 IMPLEMENTATION

Our tool provides instructors and students separate logins via a web interface hosted on a server. The instructor logs in and uploads the list of students along with their assigned user ids. This ensures that only authorized students can access the instructor's live programming session. Once the instructor starts a session, students can log in. The application is built on Nodejs with an angular frontend and a Mongo DB database. It uses Socket.io to simulate a pseudo publisher subscriber architecture. Ace, an editor of embeddable code based on JavaScript, is used as the editor in both views. JSZip, a JavaScript library designed to build, read, and edit .zip files, is used to download the source code, along with the notes the students have made. The instructor's view consists of a text editor that supports Java, C++, Nodejs, Ruby, Python and C#. The files created can be run using an online compiler, JDoodle [3]. As noted earlier, students and instructors can compile and run their code independently.

Our tool also has a Chat feature where students can post their questions as and when they have one and the instructor can answer whenever possible without being disturbed while teaching. These questions can be downloaded along with the answers as a text file. This feature saves time for students who have similar questions.

4 EVALUATION

To evaluate *Collaborative Classroom*, we conducted a small-scale study where we compared our tool against VSCode with "live share". Our study had 18 participants who are currently pursuing their 5th semester in Engineering. The educational context of the study was JavaScript fundamentals.

First, we conducted a baseline test to ascertain the participants' prior knowledge of JavaScript fundamentals. This test involved a total of 10 questions. Some of these questions involved **code tracing**, such as:

Given the following JavaScript code snippet, what is the output? (The expected answer is: 10)

```
var x = 10;
function test() {
    var x = 20;
}
test();
console.log(x);
```

Other question tested students' knowledge of JavaScript **syntax**, such as the following MCQ:

Which function among the following is used to register a function to be invoked once? (The correct choice is **setTimeout()**.)

The baseline test was followed by a 30-minute lecture on Object Construction in JavaScript using the VSCode IDE. Audio was provided via a separate online conferencing system. The instructor shared the link for a read-only live-share session and participants were allowed to choose between the Browser option and Application extension to join the session. Each participant needed to be admitted to the lecture individually. Participants could join as Guest Users as well. The instructor created files and developed JavaScript code while periodically executing it to display the output in the terminal. Participants communicated with the instructor using VSCode's inbuilt Chat feature.

After the first lecture, the students were tested on concepts from this lecture with a short test (**Test-1**) similar in format to the baseline test. A second 30-minute lecture was conducted on Prototypal inheritance using *Collaborative Classroom* for live programming and an online conference call for audio. Students made notes, navigated within the code, and executed it independently as the lecture progressed. At the end of the lecture, students are asked to download the code files along with the questions and answers asked during the session. This lecture was followed by another test (**Test-2**) of a similar format, focused on concepts from the second lecture.

Analyzed the difference in the percentage (Test-2 minus Test-1) on both code tracing and syntax type of questions. We plot these against the student scores in the baseline test on each of these categories of questions:

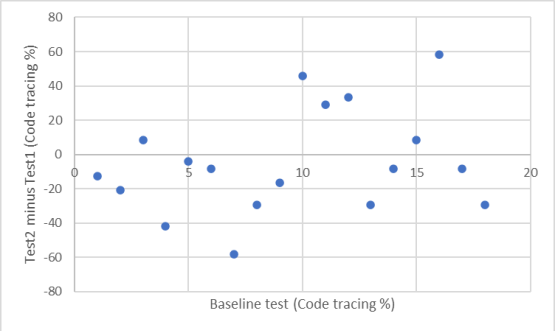

Figure 4. Each dot represents one student. The x-axis is the baseline score (%) on code tracing questions, the y-axis is the gain (Test 2 minus Test 1) in score (%) on code tracing questions.

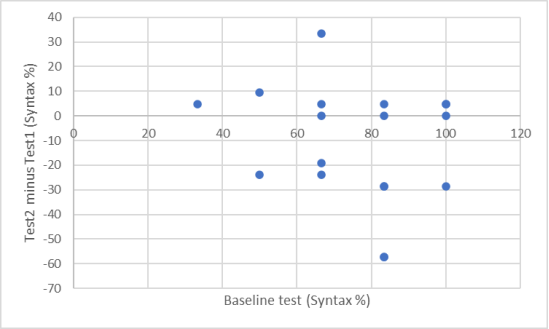

Figure 5. Each dot represents one student. The x-axis is the baseline score (%) on syntax questions, the y-axis is the gain (Test 2 minus Test 1) in score (%) on syntax questions.

There is no evident trend in Figure 4 and Figure 5. Thus, while our limited data does not suggest that *Collaborative Classroom* leads to better (or worse) outcomes on tests, it is consistent with the hypothesis that the tool is beneficial to both "weak students" (i.e., students with low scores on the baseline test) and "strong students" (i.e., students with high scores on the baseline test).

At the end, students were asked to fill out a survey to obtain their qualitative feedback regarding our tool. Overall, both VSCode and tools had positive feedback, but 77.8% said they would prefer to use *Collaborative Classroom* over VSCode for live programming lectures.

5 CONCLUSIONS

We have presented a tool, *Collaborative Classroom*, with novel features to assist students in live programming courses. Not only does our tool require very little bandwidth (when compared to video streaming), it judiciously provides autonomy to students in ways that can be beneficial. Although the evaluation of our tool has several limitations (most prominently, it is based on just 18 students), our findings do suggest that the tool can be useful to students in programming courses.

6 DEMONSTRATION

A demonstration of *Collaborative Classroom* is available at: https://youtu.be/qA2FKAIUp4c

REFERENCES

[1] John Paxton. 2002. Live programming as a lecture technique. J. Comput. Sci. Coll. 18, 2 (December 2002), 51–56.

[2] Visual Studio Magazine. 2018. Developers Find New Use Cases for Visual Studio Live Share. https://visualstudiomagazine.com/articles/2018/12/10/live-share-uses.aspx

[3] JDoodle. https://docs.jdoodle.com/compiler-api/compiler-api#what-are-the-input-parameters-for-execute-api-call