PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises

APRIL YI WANG*, University of Michigan, USA
YAN CHEN*, University of Michigan, USA
JOHN JOON YOUNG CHUNG, University of Michigan, USA
CHRISTOPHER BROOKS, University of Michigan, USA
STEVE ONEY, University of Michigan, USA

Peer assessment, as a form of collaborative learning, can engage students in active learning and improve their learning gains. However, current teaching platforms and programming environments provide little support to integrate peer assessment for in-class programming exercises. We identified challenges in conducting such exercises and adopting peer assessment through formative interviews with instructors of introductory programming courses. To address these challenges, we introduce PuzzleMe, a tool to help Computer Science instructors to conduct engaging in-class programming exercises. PuzzleMe leverages peer assessment to support a collaboration model where students provide timely feedback on their peers' work. We propose two assessment techniques tailored to in-class programming exercises: live peer testing and live peer code review. Live peer testing can improve students' code robustness by allowing them to create and share lightweight tests with peers. Live peer code review can improve code understanding by intelligently grouping students to maximize meaningful code reviews. A two-week deployment study revealed that PuzzleMe encourages students to write useful test cases, identify code problems, correct misunderstandings, and learn a diverse set of problem-solving approaches from peers.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: peer assessment, live programming, synchronous code sharing

ACM Reference Format:
April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 415 (October 2021), 24 pages. https://doi.org/10.1145/3479559

1 INTRODUCTION

Collaborative learning actively engages students to work together to learn new concepts, solve problems, and provide feedback [58]. Programming instructors often use various collaborative

*Both authors contributed equally to this research.

Authors' addresses: April Yi Wang, University of Michigan, Ann Arbor, Michigan, USA, aprilww@umich.edu; Yan Chen, yanchenm@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; John Joon Young Chung, jjyc@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Christopher Brooks, University of Michigan, 105 S State St., Ann Arbor, MI, 48103, USA, brooksch@umich.edu; Steve Oney, University of Michigan, 105 S State St., Ann Arbor, MI, 48103, USA, soney@umich.edu.

2573-0142/2021/10-ART415 $15.00
https://doi.org/10.1145/3479559

learning activities in teaching, such as group discussion, project-based work [41], pair programming [48], code debugging [26], and peer assessment [59]. In particular, peer assessment through reviewing and testing each other's solutions can improve students' motivation, engagement, and learning gains [38, 39, 53, 59], while reducing the effort required for instructors to provide scalable personalized feedback [39].

Despite the benefits of peer assessment, current programming and teaching environments provide little support to conduct peer assessment for **in-class programming exercises**—small scale programming exercises for students to practice during lectures or labs. As a result, peer assessment is typically conducted asynchronously rather than in a live classroom setting [59]. Prior research has made it easier for instructors to share and monitor code with multiple students in real time [8, 28]. However, designing real-time systems to enable both student-student interactions and student-instructor interactions in a live setting is still a challenge [10]. Moreover, students often struggle to give each other high-quality feedback or even start a fruitful conversation without proper moderation and effective grouping [7, 43]. In a needs analysis, we also found that it is difficult for instructors to effectively break students up into groups with appropriate balances of expertise in physical classroom situations. Further, because of the overall lack of expertise, peers can find it difficult to assess whether a given piece of code would fail unknown edge cases even if it generates the desired output for the test cases given by instructors.

In this paper, we present PuzzleMe, a web-based in-class programming exercise tool to address the challenges of peer assessment. PuzzleMe consists of two mechanisms: **live peer testing** and **live peer code review**. Live peer testing helps learners assess their code through moderated collection of test cases from peers. Inspired by the notion of the "sweep" [46], live peer testing seeks essential examples only for illustrating common and interesting behaviors rather than writing comprehensive test suites. PuzzleMe automatically verifies valid test cases by referencing an instructor-provided solution and shares valid test cases with the whole class. Live peer code review aims to provide personalized feedback at scale. It does this by automatically placing students in groups where they can discuss and review each other's code. PuzzleMe introduces several features to encourage meaningful code review, including a matching mechanism to balance student groups based on the number of correct answers and the diversity of those answers. PuzzleMe also includes mechanisms that allow instructors to create improvised in-class programming exercises, monitor students' progress, and guide them through solutions. Our design is inspired by formative interviews where we investigated the obstacles instructors face when conducting in-class programming exercises and encouraging peer activities.

To validate PuzzleMe's effectiveness, we deployed it to an introductory programming course for two weeks and conducted several exploratory studies. Our results show that the peer testing feature can motivate students to write more high-quality tests, help identify potential errors in their code, and gain confidence in their solutions. Further, the peer code review feature can help students correct misunderstandings of the course materials, understand alternative solutions, and improve their coding style. We also report on the use of PuzzleMe in an online lecture[1] and demonstrate its potential to be used at scale in synchronous online education. We found that PuzzleMe is perceived to be useful in a wide range of programming classes, reducing the stress of providing near-immediate feedback, helping instructors to engage students, and providing opportunities to explore different types of pedagogy.

The key contribution of this work is the design lessons learned from a series of mixed methods studies, which add to the body of work on personalized feedback, peer assessment, and in-class exercises. We believe these lessons can guide future interface design exploration in similar contexts

[1]During our deployment, this course migrated to a fully online setting due to the outbreak of COVID-19.

(e.g., live workshops and programming education via live streaming [10]). PuzzleMe shows the potential for increasing learning outcomes via in-class peer support without increasing teaching costs. Specifically, our contribution includes:

(1) an articulation of the needs and challenges that instructors have when conducting in-class exercises for introductory programming courses based on formative interviews with five instructors,
(2) two techniques—live peer testing and live peer code review—that enable peer assessment during in-class exercises, and
(3) PuzzleMe, a web-based system for instructors to carry out in-class programming exercises with the support of live peer testing and live peer code review.

2 RELATED WORK

Our work builds on three threads of research: (1) peer assessment as collaborative learning, (2) real-time code sharing in educational settings, and (3) scaling feedback.

2.1 Peer Assessment as Collaborative Learning

Collaborative learning enhances the learning experience by involving multiple learners together in collective pedagogical activities [58]. Peer assessment is a form of collaborative learning [36, 60] where students critique and provide feedback to each other's work. Previous work has shown that peer assessment can reduce the time required for assessment activities while maintaining quality. For instance, Kulkarni et al. have shown that the majority of students can evaluate their peers' work fairly [38] and that rapid peer feedback can be helpful for mastering open-ended tasks [39]. However, students may find it difficult to construct high-quality feedback due to a lack of expertise [7, 38, 66]. They often need prompting, structured guidance, proper moderating, and effective grouping [7, 49, 66]. Prior work has also demonstrated the benefits of peer assessment in programming education. Sitthiworachart et al. [57] found that peer feedback in programming courses can be helpful for both the students providing help and those receiving it. Denny et al. [16] demonstrated that by reviewing others' code, students benefit from exposure to a wider diversity of solutions. Denner et al. [15] studied the mechanisms of pairing students to encourage positive influence. Hundhausen et al. [32] proposed a pedagogical code review that can promote positive attitude and train students in critical review skills. Moreover, Dave et al. [12] found that by providing "in-flow peer review"—peer review while a problem is in progress—reviewers and reviewees can gain greater motivation.

2.1.1 Peer tests. Creating and sharing test cases is a popular form of peer review in programming education. However, in prior work, the process of sharing test cases has had to be manually performed by instructors. Smith et al. [59] incorporated peer testing into a lower-division programming course. They found that peer testing improved students' engagement and their self-efficacy as software testers [59]. However, the timescale for these peer testing exercises is not appropriate for in-class exercises; tests and reviews were submitted through web forms and distributed by email, and the assignment schedule had to be modified (adding approximately five days) to include time for students to submit and review peer tests.

Code Defenders [53] encourages software testing through gamification; students write test suites that are assessed by running them against a series of variants of a working codebase produced by others (a form of mutation testing). An evaluation of Code Defenders found that through gamification, participants wrote better test suites. Like PuzzleMe, Code Defenders engages students in the testing process by giving them real-time feedback on the tests they write. Code Defenders uses a more formal evaluation framework than PuzzleMe. Further, Code Defenders relies on mutation

testing in pairs, meaning that students are (1) not testing their own code (but instead are testing variants of an existing codebase) and (2) the testing mechanism relies on splitting students into groups where one student writes code variants and the other tests them. By contrast, PuzzleMe does not rely on a pairing mechanism for peer testing and allows students to write tests for each other's code, which makes it appropriate for a much wider variety of in-class exercises.

2.1.2 Peer review of peer tests. The tests that students write can also be useful peer review artifacts. Politz et al. [47] found that in-flow peer review can improve the quality of test suites and engage students early and thoughtfully. However, writing and reviewing comprehensive test suites can be burdensome for students, particularly those in lower-level programming courses [20]. Politz et al. [46] further proposed the idea of "sweep"—to help students explore a programming problem through interesting and representative examples. They found that sweep is very useful for students in introductory Computer Science (CS) courses to write tests and engage in peer review. However, Politz studied in-flow peer review of tests and the sweep mechanism outside of the classroom (e.g., for take-home assignments). The effect of providing rapid peer feedback in class remains largely unknown. PuzzleMe builds upon previous work and enables scalable real-time feedback for in-class coding exercises by allowing students to easily exchange test cases and perform code reviews.

2.2 Real-Time Code Sharing in Educational Settings

One challenge in supporting peer assessment for in-class programming exercises is sharing code in real time between students and instructors, and among students. Thus, we reviewed prior work on real-time code sharing in educational settings.

2.2.1 Real-time code sharing between students and instructors. Real-time code sharing is used for teaching programming via live coding, where instructors broadcast their programming activities in front of the classrooms or through recorded and live-streamed videos. To facilitate teaching programming via live coding, Chen and Guo proposed Improv, an IDE extension that integrates the presentation system directly in programming environments [8] where code edits in the IDEs are synced with slide presentations in real time. Chen et al. studied the experience of learning programming through live streaming [10]. They found that existing platforms do not adequately support interactions between viewers and streamers, which leads to a low level of engagement during the streaming sessions. Real-time code sharing between students and instructors also benefits instructors to mentor students' code and provide scalable help. For example, Codeopticon enables instructors to monitor multiple students' coding progress in real time [28]. In addition, real-time code sharing can be potentially useful for providing remote assistance and personalized support [9, 44].

2.2.2 Real-time code sharing among students. Real-time code sharing among students can increase social translucence in online programming courses. For example, Cocode presents students with the real-time code editor activities of others to improve social awareness [6]. Real-time code sharing among students can help support collaborative work. The Codestrates system follows the structure of a common notebook interface to allow block-like code representation for reprogrammable applications [50]. It shares the code and the applications across multiple users and devices, making it possible to support collaborative programming practices [4]. Relevantly, Codechella facilitates help-seeking and peer tutoring through real-time code sharing and program visualization [29]. PuzzleMe builds on these works by designing a real-time interface to support the sharing of test cases and code reviews.

2.3 Scaling Live Feedback

Providing timely and personalized feedback is crucial to help students engage with and learn from programming exercises [2, 61]. However, it can be prohibitively difficult, particularly in large classes [56]. Prior work has explored different approaches for scaling feedback: clustering and monitoring similar solutions to enable instructors to write feedback, and generating feedback automatically by synthesizing students' solutions.

2.3.1 Scaling instructors' effort. In large university classes where many students interact with a small number of instructors, giving high-quality feedback is challenging. Previous work has explored different approaches to scale instructors' effort. OverCode shows high-level patterns to instructors by clustering students' solutions, which helps them provide feedback at scale [24]. Similarly, MistakeBrowser and FixPropagator [30] help instructors scale feedback by using code synthesis to propagate bug fixes from one student to the rest of the class. TeacherASSIST scales instructor hints by crowdsourcing them from instructors in the learning platform [45]. While these tools alleviate the instructor's workload, it is difficult to apply these approaches in real-time, in-class settings as they are still limited by the instructor's bandwidth, particularly in classes with many students. Intelligent Tutoring Systems (ITSs) have proven effective in improving students' performance in programming learning [3]. However, building an effective ITS for a programming curriculum can be complex—it requires expert knowledge [52] and the ratio of tutor construction time to student interaction time can be as high as 100:1 [21]. PuzzleMe leverages peer assessment to alleviate the instructor's feedback load.

2.3.2 Synthesizing student solutions for feedback generation. Another approach to scaling feedback is to leverage prior student activity. For instance, Rivers et al. [51, 52] use student activity data to model the solution space and learning progress for programmers. This model was then used to provide students with personalized feedback. Gulwani et al. [27] introduced a system that could automatically repair incorrect programs based on correct solutions from students. They found that students perceived the automatic repair function to be useful. However, these approaches are difficult to use in real-time, in-class settings where the problems being given to learners are novel and historical data has not yet accumulated.

A complementary approach to synthesizing feedback is through *learnersourcing* [35], which engages students to leverage their learning efforts to create materials for future learners. For instance, in the video-learning setting, Weir et al. [62] leveraged learners' work to generate sub-goal labels for videos, which reduced the cognitive load of future students. Researchers have also shown that learnersourcing can be leveraged to scale feedback in math [63] and computer architecture education [23]. By synthesizing the effort of previous students, learnersourcing systems could assist students even when the lecturer is not available. Edwards et al. [18] introduced a similar approach of allowing students to create questions and share those with others. These approaches may not only lower the instructors' workload but also allow students to practice the subject with different types of feedback. PuzzleMe builds on this work of learnersourcing in a novel and task-specific way by sourcing not just student explanations or solutions but also a collection of test cases that are shared among students as learners explore the bounds of the problem.

3 IN-CLASS PROGRAMMING EXERCISE CHALLENGES

To better understand the current practices and challenges for conducting in-class programming exercises, we conducted formative interviews with instructors of introductory programming courses. We chose to focus on introductory programming courses because (1) they typically have larger enrollments, (2) students in introductory courses often need more support, and (3) large knowledge

Table 1. Instructors' course demographics in formative interviews.

PID	Size	Name	Language(s)
S1	260	Intro. to Prog. I (G)	Python
S2	250	Intro. to Engr. (U)	MATLAB
S2,S3	300	Intro. to Prog. II (U)	Python
S4	260	Intro. to UI Prog. (G)	HTML,CSS,JS
S5	260	Intro. to Prog. I (U)	Python

gaps between students are more likely, making it difficult for instructors to accommodate all students' needs.

3.1 Method

We recruited five instructors from different introductory programming courses at the authors' university. Table 1 presents an overview of the courses that the five interviewees taught. They include three undergraduate- and two graduate-level courses across three departments, with the same session time (80 minutes, twice per week). Each interview lasted 30 minutes. We asked instructors to recall the most recent introductory programming courses they had taught and explain what types of in-class exercises they conducted and what processes and tools were involved to facilitate the exercises. In addition, instructors were encouraged to tell us about any challenges they had encountered with conducting in-class programming exercises. Two authors separately conducted iterative coding to identify reoccurring themes using inductive analysis. We then merged similar codes to infer important findings. During the process, the codes of common practices for conducting in-class exercises, such as the types of exercises and tools, were merged smoothly. However, those of challenges in conducting programming exercises were rather difficult because of the authors' different perspectives (e.g., process vs. roles involved). By dividing the in-class exercise activity into different stages and identifying the major roles of each party, the authors discussed and finalized the codes.

3.2 Findings

3.2.1 In-class exercises are common. On average, interviewees spent a third of the lecture time on programming exercises. Interviewees often conducted two types of in-class exercises: multiple-choice questions (where students respond using audience response systems such as i-clickers) and programming exercises (where students write code on their laptops). In some cases, students needed to download starter code from code collaboration applications [25, 65]. None of the interviewees mentioned using assessment tools to collect and validate students' solutions. Instead, they often provided an example of an acceptable solution (e.g., on a projector) and asked students to self-check their code. All interviewees mentioned asking students to discuss with their peers or providing personalized help during office hours.

3.2.2 Impromptu exercises are valuable. To conduct effective exercises, instructors get feedback from students on what they understand and then improvise exercises based on that feedback. Some instructors would live code in class and call on students to verbally describe what code they should write (S2, S5). They encountered cases of "*a lot of people making similar misconceptions that I did not expect*" (S1), so they would often choose to let students vocally explain them to the rest of the class (S1, S4). Although vocal feedback has a low overhead cost, interviewees were concerned that "*you always get the same people participating*" (S4). S1 wished to leave more lecture time

for students to "*share their thoughts*", or "*learn from the person sitting 20 feet away.*" Additionally, vocal communication is often not accessible (e.g., hard to hear) for students and is not archived for students to revisit.

3.2.3 Hard to scale support for exercises. When conducting the exercises, all interviewees reported that they would walk around the classroom to observe students' progress (S1–S5), provide help (S1–S5), "*get teaching feedback*" by asking students "*what's hard about this*" (S1), or "*hearing about low level problems that we might not have thought about*" (S4). Four interviewees emphasized the physical challenge of navigating the classroom and interacting with students sitting in hard-to-reach spots: "*there are 130 people, you're going to run into backpacks, walking between different chairs. It's not the best way of walking around*" (S4). Moreover, because exercises were often short, instructors did not have time to gauge students' mistakes and adapt their teaching later on: "*I want to get a sense for how everyone is reacting so I can decide what I'm going to talk about next*" (S1).

3.2.4 Difficulty in connecting students with each other. All interviewees encouraged students to talk to each other while performing exercises: "*I don't care if they get the answer right. As long as you have the discussion you learn from it*" (S4). Four interviewees applied peer instruction [14] to multiple-choice questions in their lectures (S1, S2, S3, S5) and found it encourages students to monitor themselves in learning and build connections with each other. However, instructors reported a lack of intrinsic motivation for students to interact with their peers (S1–3). Instructors reported ineffective grouping as an important reason. Due to the physical distance between students in classrooms, instructors often pair students with their neighbors. Pairs are matched without regard to their diverse backgrounds, solutions, and levels of knowledge, which does not ensure that students in a pair would have meaningful conversations with each other (S3, S5). In addition, instructors mentioned the social hurdle for nudging peer interactions. Students would feel hesitant to engage with others socially without proper prompting. This corresponds to prior work on structuring roles and activities for peer learners to increase student engagement and process effectiveness [37, 55]. Instructors also reported that students would feel less comfortable asking for help from peers than instructors (S3, S4).

3.3 Summary of Design Goals

Driven by the findings, we formed three design goals for tools that scale support for in-class programming exercises:

- **Improvising in-class exercises and synchronous code sharing (3.2.1, 3.2.2)**: Instructors need a synchronous code-sharing platform for improvising in-class exercises. Tools should support various teaching needs, including creating and sharing exercises, verifying students' solutions, monitoring students' progress and activities, and walking through answers.
- **Scale student support (3.2.3):** Students need to get timely and on-demand support during in-class programming exercises.
- **Encourage live peer interaction (3.2.4):** Tools should encourage students to interact with their peers in real time to engage them and motivate active learning.

4 PUZZLEME DESIGN

We designed PuzzleMe as a platform to improve in-class programming exercises for instructors and students. PuzzleMe allows instructors to easily create and share exercises, monitor students' progress, improvise exercises as needed, and demonstrate correct solutions through live coding.

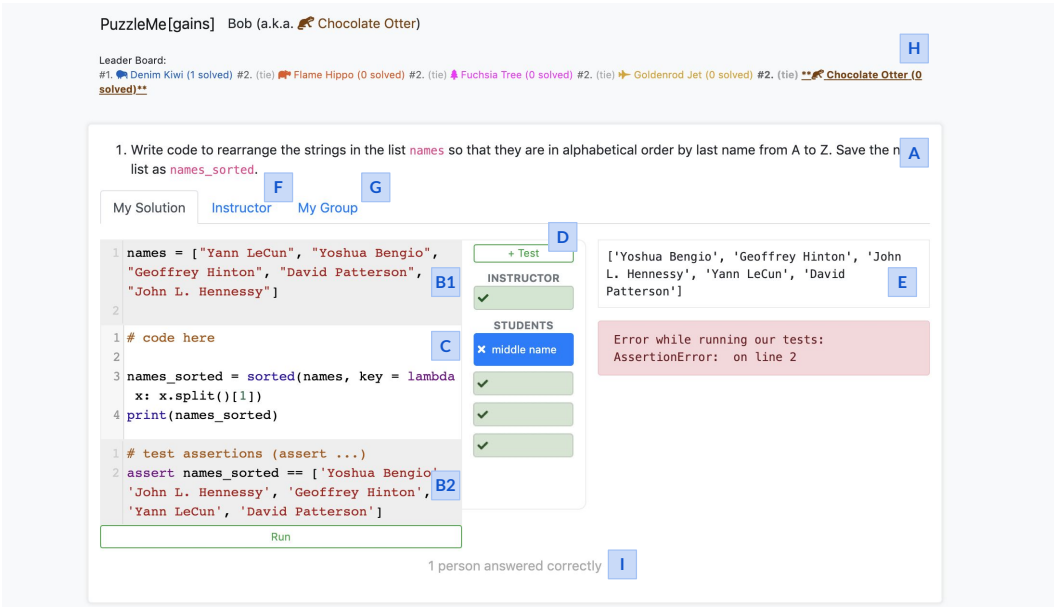

Fig. 1. PuzzleMe is a peer-driven live programming exercise tool. The student view shows: (A) a problem description; (B) an informal test that consists of the given conditions (see B1) and the assertion statement (see B2); (C) a code editor where students can work on their solutions; (D) a test library that contains valid tests shared by instructors and other students; (E) the output message of the current solution; (F) access to the instructor's live coding window; (G) access to peer code review; (H) a leaderboard of the number of problems that students have finished; and (I) the number of students who have finished the current problem.

On the student side, PuzzleMe supports live peer testing and live peer code review to scale support and encourage peer interaction.

4.1 In-Class Programming Exercises

PuzzleMe supports the types of exercises that instructors described in our interviews: programming exercises[2], multiple-choice questions, and free-response questions. For a programming exercise, an instructor can provide a problem description (Fig.1.A) and starter code (Fig.1.C) for students to build on. The editor includes a read-only code area with instructor-specified input variables (Fig.1.B1) and assertions to evaluate program output (Fig.1.B2). Fig.1.E shows the code output and error messages.

PuzzleMe also supports creating impromptu exercises in response to students' feedback and performance. Instructors can import exercises they prepared in advance or improvise and modify exercises during class. As soon as the instructor modifies existing exercises or creates a new exercise, their modifications are propagated to students. PuzzleMe propagates character-by-character edits, as we found in pilot tests that these more frequent updates kept students engaged if they had to wait as instructors wrote the exercise.

4.1.1 Live coding for answer walkthrough. Prior work has found that students prefer when instructors write out solutions in front of the class—known as "live coding" [54]. Live coding also allows instructors to teach through experimentation (for example, by demonstrating potential pitfalls as

[2]PuzzleMe currently supports Python but could easily be extended to other programming languages.

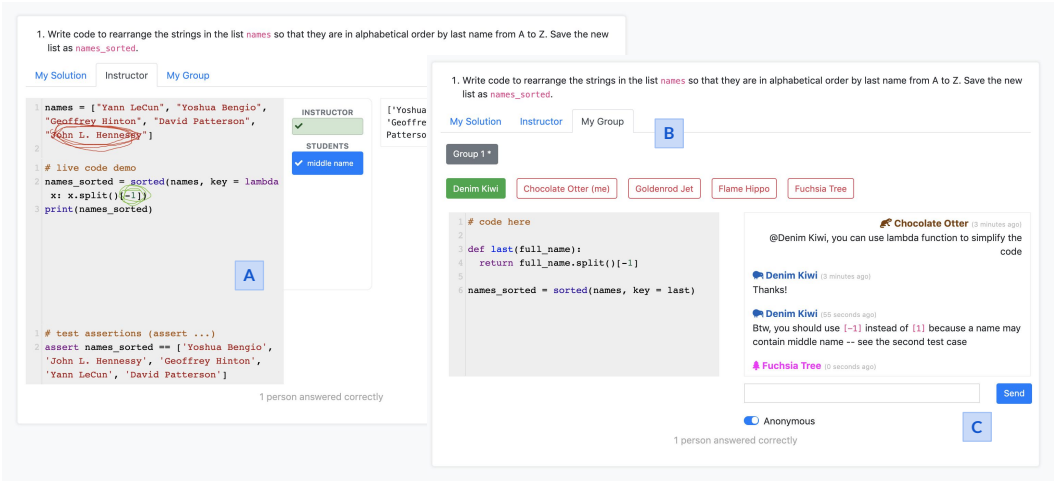

Fig. 2. The live code view and the peer code review view. (A) Instructor can enable live coding mode to demonstrate coding in real-time, and use the built-in sketching feature to assist their demonstration; (B) Live peer code review allows students to check other group members' solutions and provide reviews in a chat widget (as shown in C).

they go along), narrate their thoughts, and engage students by asking questions [54]. PuzzleMe enables live coding and propagates instructor's code changes to students (Fig.2), which allows students to easily copy and experiment with the instructor's code. PuzzleMe also enables free-form sketches and annotations to allow instructors to draw explanatory diagrams to augment their code.

4.1.2 Monitoring class progress. To monitor students' progress, similar to Codeopticon [28], PuzzleMe allows instructors to examine an individual student's response in real time. In addition, PuzzleMe presents an anonymous leaderboard (Fig.1.H) based on exercise completion time. PuzzleMe also displays how many students have written working solutions, to give a sense of progress relative to their peers (Fig.1.I).

4.2 Live Peer Testing

To give students timely feedback during in-class programming exercises, we designed **live peer testing**—a practice of writing and sharing lightweight tests in real time—in PuzzleMe.

4.2.1 Conceptual model of testing. One of the challenges of enabling live peer testing in introductory programming courses is that many testing frameworks require advanced knowledge [20]. For example, Python's `unittest` module requires an understanding of classes, inheritance, user-defined functions and more. By contrast, students in introductory courses often do not learn how to define functions or intermediate control flow mechanics until several weeks into the course. Students in introductory courses benefit more from identifying interesting and representative examples rather than constructing comprehensive test suites [46]. We thus designed a model for live peer testing that would be flexible enough to test any number of configurations while still being conceptually simple enough for students in introductory classes. In our model, the code editor is split into three parts: 1) setup code that specifies pre-conditions, 2) the student's code, and 3) assertion code that tests whether the student's code produces the correct output given the pre-conditions. The setup (Fig.1.B1) and assertion code (Fig.1.B2) are "paired", independent of the student's code (Fig.1.C).

This can be visualized as a "flipbook" that flips through pairs of pre-conditions and expected post-conditions.

4.2.2 Creating test cases. Students can create a new test case by clicking the "+ Test" button (Fig.1.D) and editing the setup code and assertion code. Instructors can specify whether tests are disabled, enabled, or enabled and mandatory. In addition, instructors can choose to require that students write a valid test case before they can start writing their solution, which can be useful for adapting different types of pedagogy like test-driven learning [33].

4.2.3 Verifying and sharing test cases. To help guide students to create effective test cases (and avoid sharing invalid test cases), PuzzleMe includes mechanisms for verifying test cases. For this to work, instructors write a reference solution that is hidden from students. To be considered acceptable, a student test must pass the reference solution (to prove that it is valid) and fail an empty solution (to prove that it is non-trivial, such as empty tests).

PuzzleMe notifies students of their test case status and students can always update and resubmit their unverified test cases. PuzzleMe shares verified test cases with all students as a test library (Fig.1.D). When students execute their code (pressing the 'Run' button), PuzzleMe uses all the cases in the library to examine their answers.

4.3 Live Peer Code Review

Live peer testing provides feedback on students' test cases and creates a test library to help them write more robust code. However, passing test cases does not ensure high-quality code. Valid solutions may contain unnecessary steps or bad coding practices. This might not affect the output of the program but may harm students' long-term coding ability [22, 42]. Thus, it is important for students to get feedback on both the correctness and the quality of their solutions. Prior work suggests that providing comparisons can help novice learners better construct feedback [7, 49]. PuzzleMe supports **live peer code review**, a feature that allows students to see and discuss each other's code. As Fig.2 shows, students can check other group members' solutions and provide reviews in a chat widget.

We originally designed the discussion widget (Fig. 2) as a "peer help" tool where students could press a "help" button to request assistance from another student. However, in pilot testing, we found that struggling students were hesitant to ask for help, even when they could do so anonymously. By framing this discussion as peer code review rather than peer help, however, PuzzleMe removes students' stigma around asking for help while still enabling valuable discussions between students.

4.3.1 Matching learners. By forming students into groups after working individually on the problem, our goal is to encourage meaningful conversations around everyone's solutions—for students who have incorrect solutions to clarify misconceptions and for students who have correct solutions to learn from others who use a different approach to solve the problem. Therefore, matching learners effectively is crucial for encouraging meaningful discussions among group members. Based on instructors' needs in the formative study to leverage peer help and match peers with diverse solutions so that they could learn from each other, we propose two heuristics for matching learners. First, students who have incorrect solutions should be paired with at least one student who has a correct solution. Second, if a group has multiple students who have correct solutions, they should have different approaches to or implementations of the problem. We determine group sizes by the proportion of students with incorrect solutions so that students who have incorrect solutions are paired with at least one student who has a correct solution. Next, we calculate the Cyclomatic Complexity Number (CCN) of the correct solutions as a proxy measure for approach. We then allocate the correct solutions by as many different approaches as possible. In addition, students may

feel hesitant to start conversations because of social barriers or feel stressed when their solutions are put together with others [19]. To address this issue, PuzzleMe keeps students anonymous [43] when sharing their solutions and reviews.

4.4 Implementation

We implemented PuzzleMe as a web application. We used ShareDB [17], a library that uses Operational Transformations (OTs) to keep instructors' and students' data updated in real time. To ensure that it can scale to large numbers of students, PuzzleMe executes all code client-side, using Skulpt [1], a library that transpiles Python code to JavaScript. We have published our source code[3] for researchers to evaluate and build on.

5 EVALUATION

In total, we conducted three studies to evaluate the effectiveness of PuzzleMe for scaling support (Studies 1 and 2) and its general applicability (Study 3). To evaluate its effectiveness, we deployed PuzzleMe in an introductory programming class for two weeks. For the first week, we compared students' performance with and without the support of PuzzleMe in four lab sessions. For the second week, we collected and analyzed the PuzzleMe usage data for an online lecture. We then conducted an interview study with teaching staff from other programming classes to understand the broader applicability of PuzzleMe.

5.1 Course Background (Studies 1 and 2)

The introductory programming course consisted of a weekly lecture and two weekly lab sessions where students were assigned to one of nine smaller lab sections for hands-on practice of coding. There were 186 undergraduate students enrolled in the class, one full-time instructor (Professor) and five Graduate Student Instructors (GSIs). Most students did not have prior experience in programming. Each GSI individually led one or two lab sections with 10–30 students. The class format changed to online with optional participation in lectures and lab sessions during the second week of the deployment.

5.2 Study 1: Using PuzzleMe in Face-to-Face Lab Sessions

To gain a holistic understanding of how PuzzleMe can be used to improve students' learning experience in in-class programming exercises, we conducted a user evaluation in four lab sections during the first week of the deployment. In particular, we aimed to answer the following questions:

- Q1 Compared to conventional methods, would *live peer testing* encourage students to write higher quality test cases? Are test cases shared by peers helpful for students to assess their code?
- Q2 Compared to conventional methods, how would *live peer code review* affect students' willingness to seek feedback from others? Would live peer code review yield more meaningful and constructive feedback?

We chose self-assessment and face-to-face discussion as a representation of conventional methods because they were widely applied by instructors in our formative studies.

5.2.1 Study setup. The GSIs gave students a set of problems to work on based on the material they were learning at the time. We used one of the programming exercises to evaluate live peer testing (noted as E1 for answering Q1) and another programming exercise to evaluate live peer code review (noted as E2 to answer Q2). For E1, the GSIs gave students 8–10 minutes to work

[3]https://github.com/soney/puzzlemi

Table 2. The four lab sections were randomly assigned into the treatment condition or the control condition.

Session ID	Condition	GSI	Total Students
T1	Treatment	I1	16
T2	Treatment	I2	16
C1	Control	I2	12
C2	Control	I1	19

individually and encouraged them to create additional test cases. For E2, the GSIs gave students around 5 minutes to work on the solution individually before placing them into groups. They then gave another 5 minutes to discuss with peers and continue working on their solutions. We used a between-subjects design where the four lab sections were randomly assigned to use PuzzleMe with or without the live features.

(Treatment) Using PuzzleMe with the Live Features: For E1, students were encouraged to write, verify, and share test cases using PuzzleMe, and use others' test cases to assess their code. For E2, PuzzleMe assigned students into groups to perform live peer code review after working individually on the problem.

(Control) Using PuzzleMe without the Live Features: Both live features in PuzzleMe were disabled in this case. Instead, students were encouraged to write test cases in their standard code editor for E1. For E2, students were asked to show their computer screens to people sitting next to them and discuss each other's solutions after working individually.

5.2.2 Data collection. We gathered the usage log of PuzzleMe (which tracks and timestamps every student submission attempt), the test cases students created in live peer testing, and the feedback students sent one another through live peer code review. In addition, we engaged in a follow-up interview with two GSIs and six students, where we asked about their learning or teaching experience of the lab sections and how they perceived the usefulness of PuzzleMe. Finally, we made observational notes during the lab sessions, where two of the authors sat at the back of each lab session and collected data on students' participation in the class, overall performance on the programming exercises, and usability issues with PuzzleMe.

5.2.3 PuzzleMe encourages students to create and share test cases. Table 3 shows that with the live features, students wrote 0.72 test cases on average (σ = 0.73). Otherwise, students wrote 0.26 test cases on average (σ = 0.44). In both conditions, students wrote less than one test case on average. We believe this reflects realistic use of the tool with novice programmers in an in-class setting where students were given less time and were less motivated to engage in the exercises as compared to writing test cases as an assignment. To evaluate test quality, two authors manually coded student-written test cases into three levels (Figure 3), assigning a 0 if the test case was invalid or duplicated the default case, a 1 if the test case only performed additional checks on the output without changing the given conditions, or a 2 if the test case checked assertions under new conditions. We found that the average quality of the test cases in the treatment condition was 0.96 (σ = 0.71). The average quality of the test cases in the control condition was 0.63 (σ = 0.74). We did not find any incorrect tests that slipped through the validation procedure. We also found that 37.5% of students improved the code quality after the group discussion in the treatment condition, while only 12.9% improved the code in the control condition.

Table 3. The number and quality of test cases students wrote in E1 (mean: $\overline{x}$, standard deviation: σ). The number of students who improved code (calculated by completion status) after group discussions in E2 (total: N). Our comparison suggests that the number of test cases is significantly different in the two conditions (p = 0.002, Mann-Whitney U test with power = 0.98); the number of students who improved code is also significantly different in the two conditions (p = 0.02, proportions z-test given the binary data type).

E1 Writing Test	**Condition**	$\overline{x}$	σ
Number of Test Cases*	Treatment	0.72	0.73
	Control	0.26	0.44
Test Quality	Treatment	0.96	0.71
	Control	0.63	0.74
E2 Code Discussion	**Condition**	**N**	**Total**
Number of Students Who Improved Code*	Treatment	12	32
	Control	4	31

Our comparison suggests that the number of test cases (p = 0.002, Mann-Whitney U test with power = 0.98) is significantly different in the two conditions. We do not find significant differences between the test quality in the two conditions (p = 0.13, Mann-Whitney U test with power = 0.353).

In the follow-up interviews, the students and instructors explained why they felt the live peer testing feature was useful. Live peer testing gives students feedback on their test cases, ensuring the quality of the shared test pool because PuzzleMe verifies the test cases against a known correct solution before sharing across the student body. In contrast, students in the control condition were hesitant to write new tests because they "*don't know if my code is being tested correctly*" (C2). Second, the participants felt that writing tests improved their understanding of the learning materials overall. Students commented that the "*writing test was helpful to practice the new coding skill learned from class readings*" (T2). Similarly, I2 mentioned that "*I think [writing tests] might be helpful for students to think about what they should expect from their program*". Lastly, both students and instructors

```
names = ['Alice', 'Bob', 'Charlie']                    Score: 0
# code here

assert names_sorted == ['Charlie', 'Alice', 'Bob']
```

```
names = ['Alice', 'Bob', 'Charlie']                    Score: 1
# code here

assert len(names_sorted[0]) > len(names_sorted[1])
```

```
names = ['Alice', 'Bob', 'Mark']                       Score: 2
# code here

assert names_sorted == ['Alice', 'Mark', 'Bob']
```

Fig. 3. An example of three different levels of test cases for one exercise (Problem description: alphabetically sort the given array, `names`, and assign the output to a variable named, `names_sorted`). Test cases were manually coded into three levels: 0 if the test case was wrong, meaningless, or duplicated the default case; 1 if the test case did not create new examples of `names` but added additional checks on the output `names_sorted`; 2 if the test case contained new examples of `names` and `names_sorted`.

reported that the live peer testing feature helped the former gain confidence in their solutions, and both instructors indicated that PuzzleMe might help identify problems in students' solutions. Moreover, the instructors reported that the live peer testing feature reduced their teaching stress as students gained confidence:

> (PuzzleMe) takes off some stress from me to check students' code. A lot of times students don't have a lot of questions, but they want me to check their code and make sure their code is correct. Students always have more faith in other students than themselves. If they pass their own assertion, they will still be unsure. If they pass others' assertions, they will be definitely more sure. (I2)

5.2.4 PuzzleMe scaffolds group discussions. We observed that the face-to-face group discussions were largely affected by the layout of the classroom in the control condition—"*(Face-to-face discussion) depends on the physical setting and how many people are sitting. One of my lab session[s] is smaller while the other session is more spread out*" (I2). In addition, some students found it difficult to have meaningful conversations with their neighbors in the classroom setting. One student mentioned in the follow-up interview, "*Sometimes, neither of us know the solution. It's a little awkward*" (a student from C2). Instructors reported that the matching mechanism overcomes the physical limitation of face-to-face discussion:

> In my class, students who sit in the front always finish their code, so talking to neighbors didn't really work. I like the matching in PuzzleMe because it can really pair students based on their solutions. (I2)

5.2.5 PuzzleMe encourages students to explore alternative solutions. PuzzleMe encourages students to explore and discuss alternative solutions, which may help them better apply the concepts they learned. For example, one session covered sorting and advanced functions. Students could solve a problem by either passing a `lambda` expression or a traditional named function to the `sorted` function. Students found it useful to see others' solutions and understand both ways to solve the problem—"*PuzzleMe is very helpful especially my classmates ask questions that I didn't think to ask*" (a student from T1); "*Live peer code review is good because we get to see the other ways people do their code.*" (a student from T2).

In addition, the results suggest that the live peer code review feature helps students improve their completion status. As shown in Table 3, the number of students who improved code ($p = 0.02$, proportions z-test given the binary data type) after group discussions in E2 significantly improved with the live peer code review feature. However, we suspect that multiple factors may lead to an improvement in completion status. For example, students may directly copy and paste peers' solutions without thinking, which is not our intention when designing live peer code review. To understand how the live peer code review feature helps students complete the code exercise, we continued with Study 2 to collect the PuzzleMe usage logs from an online lecture.

5.2.6 Limitation. Although we designed the experiment to balance the multiple confounds (e.g., instructors, room size, as shown in Table 2), it is difficult for us to conduct a rigorous comparison between the two conditions given that we deployed PuzzleMe in an authentic usage scenario. Factors like total students who showed up to each session were hard to control and may reduce the external validity of the results. Thus, we focused on empirical evidence of how PuzzleMe can be used to conduct in-class programming exercises. A well-controlled exhaustive study—including use of the application during a complete course and evaluation of the students' grades or the instructors' perceived workload—will be needed to explore the pedagogical benefits of PuzzleMe.

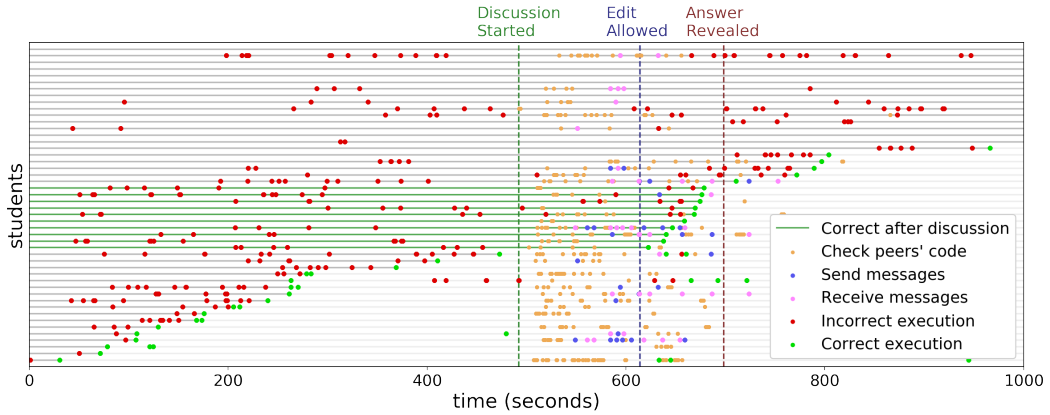

Fig. 4. Usage logs from the online lecture. With the help of peers, 10 students who had incorrect solutions were able to pass the problem (as indicated in green horizontal lines). Live peer code review helps students identify cavities in their code and inspires them to explore alternative approaches.

5.3 Study 2: Integrating PuzzleMe in an Online Lecture

Study 1 indicates that the live peer code review feature is related to the improvement of code completion. To further explore whether students were using the tool as we expected, we collected and analyzed the PuzzleMe usage data for a lecture. By the time of the deployment, the class was changed to an online format where students attended lectures synchronously using video conferencing tools. This allows us to additionally test the benefits of using PuzzleMe in online classrooms.

5.3.1 Study setup. During the live-streamed lecture, the instructor gave students an in-class programming exercise through PuzzleMe to practice the concepts they learned about that week (higher order functions—`map`, `filter`, and list comprehensions). There were multiple ways to solve the problem and students were encouraged to explore and find the most concise solution. After initial exploration for about five minutes, the instructor turned on live peer code review mode and asked students to discuss with their peers. We collected and analyzed the events log from the usage data. In total, we collected data from N=48 students who participated in the programming exercise.

5.3.2 Results overview . Figure 4 shows the event logs where each row represents a student and the x-axis represents the timeline. We sorted students based on the first time they passed the default test. On average, each student had 13.75 attempts with incorrect solutions (meaning they ran their code and failed at least one test case) and 2.63 attempts with correct solutions.

PuzzleMe connected students who were struggling with the problem with their peers for help. Before the live peer code review started, 17 students passed the problem. After the peer code review activity, an additional 10 students were able to pass the problem. Finally, 6 students passed the problem after the instructor revealed the correct solutions, and the remaining 15 students did not finish the problem within the given time (1000 seconds). We observed an additional 8 students passing the problem after the given time.

5.3.3 Code review is correlated to better completion status. For students who were not able to complete the problem before peer code review, we ran a proportions z-test to identify whether there is a correlation between using the code review feature (as indicated by blue and orange dots in

Solution A

```
common_words = ['of', 'a', 'the', 'an']

def acronymBuilder(sentence, to_ignore = common_words):
    return ''.join([i[0] for i in sentence.split() if i not in common_words])
```

Solution B

```
common_words = ['of', 'a', 'the', 'an']

def acronymBuilder(sentence, to_ignore = common_words):
    return ''.join([i[0] for i in sentence.split() if i not in to_ignore])
```

Solution C

```
common_words = ['of', 'a', 'the', 'an']

def acronymBuilder(sentence, to_ignore=common_words):
    words, acro = [], []
    for word in sentence.split():
        if word not in to_ignore:
            words.append(word)
    for word in words:
        acro.append(word[0])
    return ''.join(acro)
```

Fig. 5. Three example solutions that pass the default test. Solution A is a false positive because students did not use the function arguments correctly. Solution B is the most elegant way to solve the problem. Solution C is correct but does not demonstrate an understanding of advanced list operations.

Figure 4) and solving the problem. The result shows that there is a strong correlation between using the code review feature and eventually solving the problem ($p = 0.02$). We examined the editing histories of the code and did not observe students directly copying and pasting others' solutions. Our interpretation is that students are self-motivated to work out their own solutions rather than having a correct solution since the programming exercise is voluntary and not associated with grades. This corresponds to the observation that students who failed to improve the code were inspired by their peers' code while continuing to work on their original solutions.

5.3.4 Who initiates the talk? Conversations need nudging. Although students reported in Study 1 that they felt more comfortable talking to peers in PuzzleMe, we observed that there was no discussion going on in 8 of the 16 groups. Most members in the 8 groups "shied away" from talking to peers, though they still actively engaged with the tool to check their peers' code or make code execution requests. In half of the other groups, students who already had the correct solutions (in the helper role) initiated the conversation; in the other half, students who sought help initiated the conversation. This result corresponds with the general challenge of nudging peer-driven conversations in formative study. Next, we looked into the content of the conversations and found that the common topics included making comparisons between various solutions, providing suggestions on variable naming and formatting, seeking help on debugging, and clarifying the lecture content. In addition, we examined the group discussions and did not find any propagation of misconceptions through peer code review.

5.3.5 Code sharing improves engagement. Lastly, we observed that students who used the live peer code review feature tended to stay engaged with the exercise. For students who passed the default test, PuzzleMe inspired them to explore alternative approaches and think about issues in their code that they might have missed. We manually examined students' code and found some solutions contained a similar bug that the default test case did not catch (as shown in Figure 5, Solution A). This bug was caused by students' misunderstanding of a critical concept in previous lecture sessions—using default values for function arguments. Among the 17 students who initially passed the problem, 8 students made this mistake. Through talking to peers and checking their code, three students were able to identify the bug in their original code. In addition, several students passed the problem with correct yet naive solutions (as Figure 5, Solution C shows). PuzzleMe informed these

students of the existence of other solutions and encouraged them to explore them further. In total, we found 13 students who had additional attempts after they passed the problem the first time.

5.4 Study 3: The Practical Applicability of PuzzleMe

To explore the practical applicability of PuzzleMe, we conducted an exploratory study with four programming course instructors from the authors' university. Participants had varied experience teaching a wide range of programming topics in both in-class and online settings (as Table 4 shows). We first gave participants a walkthrough of PuzzleMe and asked them to interact with the features. Then we asked them to brainstorm the possible use cases of PuzzleMe in their courses. We also encouraged participants to envision other features they would like to have in a future design.

5.4.1 The use case of PuzzleMe in various programming topics . Table 5 summarizes the use cases for PuzzleMe. For live peer testing, most of the use cases (6/7) relate to specific topics (e.g., User Interface (UI) testing), and the rest (peer challenge) are class activities that can increase students' engagement (P3). The six specific topics are categorized as the functionality of a system (e.g., security testing, UI testing, data visualization), different modalities (e.g., in-circuit testing, design feedback), and more generic topics (e.g., algorithm design).

In particular, P1 listed a series of topics in computer security that could use the live peer testing feature, including fuzz testing, side channel attacks, and cross-site scripting attacks. Analogous to the test-driven learning approach [34], P1 envisioned that the live peer testing feature could help students learn programming concepts by writing test cases to "*break the instructors' program*" (with respect to data security), as prior work has proposed [59]. Instructors (P1, P2) also suggested peer design feedback in which students can benefit from diverse responses from their peers.

More than half of the use cases (3/5) for the live peer code review feature related to team collaboration (e.g., guided peer support, working in groups, group competition), and the other two related to team matching. P1 suggested that rather than showing each other's code immediately, which may cause students to "*lose the motivation to work on your own implementation*", a future design of PuzzleMe could allow one student to guide others to complete the problem first, and then unlock each other's code for further review. P3 mentioned that using a leaderboard among individuals or groups would motivate students to be more engaged with the exercise.

Participants also envisioned a set of use cases for other subjects. P2 mentioned that students in graphical design or creative writing courses could get peer support and feedback on their artifacts without solely relying on and waiting for their instructors' feedback. P3 suggested that PuzzleMe could be used for coding interviews (e.g., "*it's kind of similar to hackerrank (a coding interview site)*"), where one student plays the role of the interviewee to write the program and the other students are interviewers writing test cases to challenge their peer.

Table 4. Participants' background in Study 3.

PID	Course Name	Course Size
P1	Intro to Computer Security	300
P1,P2	Intro to UI Development	150
P2	Intelligent Interactive System	50
P3	Applied Data Science (Online)	145
P4	Natural Language Processing	100
P4	Data Mining	80

Table 5. Use cases of two PuzzleMe features—live peer testing and live peer code review—in various programming topics.

Live Peer Testing
Computer security-related testing: Test the security level of others' programs (e.g., login systems).
UI testing: Navigate each other's UI (e.g., the responsiveness effect).
Data visualization: Interact with each other's viz. systems (e.g., missing value, different input data).
Algorithm: Test edge cases for others' algorithms (e.g., regular expression to extract domain from a URL).
In-circuit testing: Test each other's circuits (e.g., virtual probe for breadboard testing).
Physical and digital artifact design feedback: Write feedback on each other's designs.
Peer challenge: One student writes a program, the other writes tests to challenge (break) it.
Live Peer Code Review
Working in a group: Students can work in groups to solve problems.
Roleplay: Students can choose to be a helper or a help seeker based on their interests.
Guided support: Students can provide hints to other students in their group.
Different matching mechanism: Match students by their process (similar/different), or engagement level.
Group competition: Students are divided into groups and compete with other groups.

5.4.2 PuzzleMe lowers the effort for setting up in-class exercises. Besides use cases, participants also pointed out the potential benefits of using PuzzleMe for lowering the effort involved in setting up in-class exercises. For example, in a UI development class, students can take a UI front-end-related exercise on PuzzleMe without the effort of configuring the environment (P3). Similarly, for exercises that require external resources (e.g., libraries, data sets), PuzzleMe could provide a resource hub to which the students can easily connect (P4). More generally, PuzzleMe could support instructors to create new exercises by modifying the previous ones, making it easier for students to practice on the same topic iteratively (P1, P4).

6 DISCUSSION

Our evaluation studies demonstrate that the design of PuzzleMe allows instructors to improvise in-class programming exercises, while effectively leveraging peer feedback through live peer testing and live peer code review. In particular, PuzzleMe motivates students to write more test cases, identify gaps in their code, and explore alternative solutions—all important pedagogical goals of an introductory programming course. We reflect on the design of PuzzleMe and discuss the implications for future Human-Computer Interaction (HCI) and Computer-Supported Cooperative Work (CSCW) research.

6.1 Design Lessons

6.1.1 The benefits of learnersourced test creation. Learnersourced test case creation benefits multiple stakeholders. Although individual students' test case coverage might not be as thorough as those written by instructors, they save time and effort and enable improvised programming exercises. More importantly, the process of creating test cases helps learners verify their understanding of the problem and of test-driven development while simultaneously contributing to a larger pool of verification instruments to be used by the whole class. Writing tests also helps learners practice the ability to predict the outcomes of a given input by manually walking through their code (either in their mind or by writing intermediate results on paper), a critical strategy for scaffolding novice programmers [64]. Finally, the test cases themselves provide additional diagnostic material that

might be used by instructors post-hoc, allowing them to reflect on the misconceptions students formed and aiding in the design of future learning activities.

6.1.2 The social and cultural value of building collaborative learning platforms. The live peer testing and live peer code review features in PuzzleMe scaffold peer interactions in programming classes. We observed that students used topics related to their interests when creating test cases (e.g., in the name-sorting exercise, students created test cases using names that are popular nationally, regionally, and culturally), which shed light on the unique social and cultural aspects of students' backgrounds [13]. We believe that the opportunity to create test cases related to their interests adds additional motivation for students to engage in the test creation process. In addition, our evaluation indicates that through sharing tests and discussing code with peers, students feel more engaged and connected with the class. Particularly in online classrooms, students would largely benefit from a stronger degree of social presence [40] by interacting with their peers in various learning activities. Designers of future collaborative learning platforms may learn from our design and leverage synchronous technologies to build social and structured activities [7, 10, 43] in platforms that connect students.

6.1.3 The challenges of connecting students. We designed PuzzleMe as a platform for easily creating and distributing programming exercises while engaging students through live peer testing and live peer code review. As we found when designing PuzzleMe, students might feel shy or hesitant to talk to each other, especially in situations that might expose their lack of understanding of the material to their peers, such as asking for help. We proposed various design decisions to create an encouraging collaborative space, such as providing real-time feedback on test cases so that students are comfortable sharing them with others, keeping students anonymous when sharing code and reviews, and allowing instructors to monitor and intervene in group conversations. However, our evaluation suggests that prompting conversations between students is still challenging. Compared to live peer code review, students might feel more comfortable with non-conversational interaction with peers (e.g., sharing and using test cases created by others). Future research could look into the reasons that students fail to connect with their peers and address the challenge from both social and technical perspectives. For example, it is worth exploring the effect of the pairing mechanism on students' self-efficacy and power dynamics in group discussions.

6.2 Future Work

6.2.1 Towards test-driven learning. As participants in Study 3 mentioned, PuzzleMe can be useful in supporting *test-driven learning*, a pedagogical approach that can improve comprehension of concepts but requires extra learning effort in creating test cases, particularly in early programming courses [34]. PuzzleMe can help address these challenges by reducing the barriers and learning costs for students to write test cases. PuzzleMe introduces a straightforward design that maps the given variables, solution code, and assertion statement in a linear order so that even students who have no experience in testing frameworks can easily pick up how to create tests. PuzzleMe ensures that students can get immediate feedback on the test by running it against the instructor's standard solution. This mechanism increases students' confidence in their test cases, and thus the problem formulation, before even starting to write a solution. Finally, peer-driven test creation provides the opportunity for HCI and CSCW researchers to further explore different implementations of

test-driven learning. For example, the instructor can provide a solution with intentional bugs and ask students to identify edge cases to catch these bugs or misconceptions.

6.2.2 Peer assessment beyond introductory programming. Adopting PuzzleMe's approach beyond introductory programming might require further design exploration regarding aspects such as assessment format or content presentation. Prior work has explored ways to improve peer assessment quality in open-ended tasks, such as providing comparisons [7], framing task goals carefully [31], and using expert rubrics [67]. Future work could explore the use case of live peer testing and live peer code review in open-ended assessment on programming-related topics (e.g., providing feedback for system architectural design, code review). Content-wise, P3 from Study 3 suggested a "top down camera view" for in-circuit live peer testing. Prior work also introduced a representation of Graphical User Interface (GUI) test cases that is more readable than a standard textual log [11]. Future work could explore the appropriate design for exercises that require more than a text exchange between peers [5].

6.2.3 Towards matching peers intelligently. PuzzleMe leverages the correctness of students' code for peer matching, but future work could use different criteria. For example, one could extract code fixes from students who have achieved the correct solution after multiple attempts and apply them to students who have incorrect solutions [30], capturing conceptual pathways through the problem space. One could also connect students who make similar mistakes, where one has resolved the problem and others have not, ensuring the help giver has experience in addressing their peers' issues. The matching criteria may also depend on the deployment context. For instance, in Massive Open Online Courses (MOOCs), creating culturally diverse groups of learners may provide additional learning opportunities—students may be able to not only learn a given programming concept but gain intercultural competencies while doing so.

6.3 Limitations

The design of PuzzleMe was tailored for introductory programming courses, and the design of the live peer testing component was focused on simple programs made up of a pair of given conditions and expected outputs. Advanced testing techniques like exceptions, callbacks, and dynamic tests are not implemented in the current system but may represent additional opportunities for learner collaborations. PuzzleMe's current design cannot be used directly for non-text-based programming and testing, like UI testing or Printed Circuit Board (PCB) testing. In addition, our evaluation was done on a small scale with fewer than 50 subjects. Future work should explore the effectiveness of peer assessment mechanisms in larger classrooms or MOOCs.

7 CONCLUSION

This paper presents PuzzleMe, an in-class programming exercise tool for providing high-quality peer feedback at scale. PuzzleMe achieves this by two peer assessment mechanisms: live peer testing, which allows students to identify and share test cases for assessing the robustness of programming work, and live peer code review, which groups students intelligently to improve code understanding. Our evaluation study demonstrates the usefulness of PuzzleMe in helping students identify cavities in their code and explore alternative solutions, and in reducing the teaching load for instructors. PuzzleMe opens up possibilities for HCI and CSCW researchers to further study learnersourced test creation and test-driven learning in introductory programming courses.

8 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos IIS 1755908 and DUE 1915515.

REFERENCES

[1] 2020. Skulpt. https://skulpt.org/ Accessed: April, 2020.
[2] Susan A Ambrose, Michael W Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. 2010. *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.
[3] John R. Anderson and Edward Skwarecki. 1986. The automated tutoring of introductory computer programming. *Commun. ACM* 29, 9 (1986), 842–849.
[4] Marcel Borowski, Johannes Zagermann, Clemens N Klokmose, Harald Reiterer, and Roman Rädle. 2020. Exploring the Benefits and Barriers of Using Computational Notebooks for Collaborative Programming Assignments. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 468–474.
[5] Brian Burg, Amy J. Ko, and Michael D. Ernst. 2015. Explaining Visual Changes in Web Interfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 259–268. https://doi.org/10.1145/2807442.2807473
[6] Jeongmin Byun, Jungkook Park, and Alice Oh. 2020. Cocode: Co-learner Screen Sharing for Social Translucence in Online Programming Courses. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–4.
[7] Julia Cambre, Scott Klemmer, and Chinmay Kulkarni. 2018. Juxtapeer: Comparative Peer Review Yields Higher Quality Feedback and Promotes Deeper Reflection. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article 294, 13 pages. https://doi.org/10.1145/3173574.3173868
[8] Charles H. Chen and Philip J. Guo. 2019. Improv: Teaching Programming at Scale via Live Coding. In *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale* (Chicago, IL, USA) *(L@S '19)*. Association for Computing Machinery, New York, NY, USA, Article 9, 10 pages. https://doi.org/10.1145/3330430.3333627
[9] Yan Chen, Jaylin Herskovitz, Gabriel Matute, April Wang, Sang Won Lee, Walter S Lasecki, and Steve Oney. 2020. EdCode: Towards Personalized Support at Scale for Remote Assistance in CS Education. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–5.
[10] Yan Chen, Walter S Lasecki, and Tao Dong. 2021. Towards Supporting Programming Education at Scale via Live Streaming. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW3 (2021), 1–19.
[11] Yan Chen, Maulishree Pandey, Jean Y Song, Walter S Lasecki, and Steve Oney. 2020. Improving Crowd-Supported GUI Testing with Structural Guidance. In *Proceedings of the SIGCHI conference on human factors in computing systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3313831.3376835
[12] Dave Clarke, Tony Clear, Kathi Fisler, Matthias Hauswirth, Shriram Krishnamurthi, Joe Gibbs Politz, Ville Tirronen, and Tobias Wrigstad. 2014. In-Flow Peer Review. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference* (Uppsala, Sweden) *(ITiCSE-WGR '14)*. Association for Computing Machinery, New York, NY, USA, 59–79. https://doi.org/10.1145/2713609.2713612
[13] Diana Cordova and Mark Lepper. 1996. Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization, and Choice. *Journal of Educational Psychology* 88 (12 1996), 715–730. https://doi.org/10.1037/0022-0663.88.4.715
[14] Catherine H Crouch and Eric Mazur. 2001. Peer instruction: Ten years of experience and results. *American journal of physics* 69, 9 (2001), 970–977.
[15] Jill Denner, Linda Werner, Shannon Campe, and Eloy Ortiz. 2014. Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education* 46, 3 (2014), 277–296.
[16] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite: Supporting Student-Driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 471–476. https://doi.org/10.1145/1953163.1953299
[17] Collaborative editing library. 2020. ShareDB. https://github.com/share/sharedb Accessed: April, 2020.
[18] Stephen H. Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 188–193. https://doi.org/10.1145/3059009.3059055
[19] Carolyn D. Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting Developers' Negative Feelings about Code Review. In *2020 IEEE/ACM 42nd International*

Conference on Software Engineering (ICSE '20). IEEE. https://doi.org/10.1145/3377811.3380414

[20] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug hunt: Making early software testing lessons engaging and affordable. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 688–697.

[21] Jeremiah T Folsom-Kovarik, Sae Schatz, and Denise Nicholson. [n.d.]. Plan ahead: Pricing ITS learner models.

[22] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. 2015. Foobaz: Variable Name Feedback for Student Code at Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 609–617. https://doi.org/10.1145/2807442.2807495

[23] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. 2016. Learnersourcing Personalized Hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1626–1636. https://doi.org/10.1145/2818048.2820011

[24] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Trans. Comput.-Hum. Interact.* 22, 2, Article 7 (March 2015), 35 pages. https://doi.org/10.1145/2699751

[25] Max Goldman, Greg Little, and Robert C. Miller. 2011. Real-Time Collaborative Coding in a Web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 155–164. https://doi.org/10.1145/2047196.2047215

[26] Scott Grissom and Mark J Van Gorp. 2000. A practical approach to integrating active and collaborative learning into the introductory computer science curriculum. In *Proceedings of the seventh annual CCSC Midwestern conference on Small colleges*. 95–100.

[27] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. 2018. Automated Clustering and Program Repair for Introductory Programming Assignments. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 465–480. https://doi.org/10.1145/3192366.3192387

[28] Philip J. Guo. 2015. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 599–608. https://doi.org/10.1145/2807442.2807469

[29] Philip J Guo, Jeffery White, and Renan Zanelatto. 2015. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 79–87.

[30] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (Cambridge, Massachusetts, USA) *(L@S '17)*. Association for Computing Machinery, New York, NY, USA, 89–98. https://doi.org/10.1145/3051457.3051467

[31] Catherine M. Hicks, Vineet Pandey, C. Ailie Fraser, and Scott Klemmer. 2016. Framing Feedback: Choosing Review Environment Features That Support High Quality Peer Assessment. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 458–469. https://doi.org/10.1145/2858036.2858195

[32] Christopher D Hundhausen, Anukrati Agrawal, and Pawan Agarwal. 2013. Talking about code: Integrating pedagogical code reviews into early computing courses. *ACM Transactions on Computing Education (TOCE)* 13, 3 (2013), 1–28.

[33] David Janzen and Hossein Saiedian. 2008. Test-Driven Learning in Early Programming Courses. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '08)*. Association for Computing Machinery, New York, NY, USA, 532–536. https://doi.org/10.1145/1352135.1352315

[34] David S. Janzen and Hossein Saiedian. 2006. Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA) *(SIGCSE '06)*. Association for Computing Machinery, New York, NY, USA, 254–258. https://doi.org/10.1145/1121341.1121419

[35] Juho Kim et al. 2015. *Learnersourcing: improving learning with collective learner activity*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[36] Ingo Kollar and Frank Fischer. 2010. Peer assessment as collaborative learning: A cognitive perspective. *Learning and Instruction* 20, 4 (2010), 344–348.

[37] Yasmine Kotturi, Chinmay E Kulkarni, Michael S Bernstein, and Scott Klemmer. 2015. Structure and messaging techniques for online peer learning systems that increase stickiness. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. 31–38.

[38] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, and Scott R. Klemmer. 2013. Peer and Self Assessment in Massive Online Classes. *ACM Trans. Comput.-Hum. Interact.* 20, 6,

Article 33 (Dec. 2013), 31 pages. https://doi.org/10.1145/2505057

[39] Chinmay E. Kulkarni, Michael S. Bernstein, and Scott R. Klemmer. 2015. PeerStudio: Rapid Peer Feedback Emphasizes Revision and Improves Performance. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (Vancouver, BC, Canada) *(L@S '15)*. Association for Computing Machinery, New York, NY, USA, 75–84. https://doi.org/10.1145/2724660.2724670

[40] Patrick R Lowenthal. 2010. Social presence. In *Social computing: Concepts, methodologies, tools, and applications*. IGI global, 129–136.

[41] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. 367–371.

[42] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. 2015. AutoStyle: Toward Coding Style Feedback at Scale. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (Vancouver, BC, Canada) *(L@S '15)*. Association for Computing Machinery, New York, NY, USA, 261–266. https://doi.org/10.1145/2724660.2728672

[43] Thi Thao Duyen T. Nguyen, Thomas Garncarz, Felicia Ng, Laura A. Dabbish, and Steven P. Dow. 2017. Fruitful Feedback: Positive Affective Language and Source Anonymity Improve Critique Reception and Work Outcomes. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (Portland, Oregon, USA) *(CSCW '17)*. Association for Computing Machinery, New York, NY, USA, 1024–1034. https://doi.org/10.1145/2998181.2998319

[44] Steve Oney, Christopher Brooks, and Paul Resnick. 2018. Creating Guided Code Explanations with chat. codes. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 131.

[45] Thanaporn Patikorn and Neil T. Heffernan. 2020. Effectiveness of Crowd-Sourcing On-Demand Assistance from Teachers in Online Learning Platforms. In *Proceedings of the Seventh ACM Conference on Learning @ Scale* (Virtual Event, USA) *(L@S '20)*. Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/3386527.3405912

[46] Joe Gibbs Politz, Joseph M Collard, Arjun Guha, Kathi Fisler, and Shriram Krishnamurthi. 2016. The Sweep: Essential Examples for In-Flow Peer Review. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 243–248.

[47] Joe Gibbs Politz, Shriram Krishnamurthi, and Kathi Fisler. 2014. In-Flow Peer-Review of Tests in Test-First Programming. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14)*. Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/2632320.2632347

[48] David Preston. 2005. Pair programming as a model of collaborative learning: a review of the research. *Journal of Computing Sciences in colleges* 20, 4 (2005), 39–45.

[49] Thomas W. Price, Joseph Jay Williams, Jaemarie Solyst, and Samiha Marwan. 2020. Engaging Students with Instructor Solutions in Online Programming Homework. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3313831.3376857

[50] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R Eagan, and Clemens N Klokmose. 2017. Codestrates: Literate computing with webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 715–725.

[51] Kelly Rivers and Kenneth R. Koedinger. 2013. Automatic Generation of Programming Feedback; A Data-Driven Approach. In *AIED Workshops*.

[52] Kelly Rivers and Kenneth R Koedinger. 2015. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education* (2015), 1–28.

[53] José Miguel Rojas, Thomas D White, Benjamin S Clegg, and Gordon Fraser. 2017. Code defenders: crowdsourcing effective tests and subtle mutants with a mutation testing game. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 677–688.

[54] Marc J. Rubin. 2013. The Effectiveness of Live-Coding to Teach Introductory Programming. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 651–656. https://doi.org/10.1145/2445196.2445388

[55] Jeffrey Saltz and Robert Heckman. 2020. Using Structured Pair Activities in a Distributed Online Breakout Room. *Online Learning* 24, 1 (2020), 227–244.

[56] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated Feedback Generation for Introductory Programming Assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/2491956.2462195

[57] J. Sitthiworachart and M. Joy. 2003. Web-based peer assessment in learning computer programming. In *Proceedings 3rd IEEE International Conference on Advanced Technologies*. 180–184. https://doi.org/10.1109/ICALT.2003.1215052

[58] Barbara Leigh Smith and Jean T MacGregor. 1992. What is collaborative learning.

[59] Joanna Smith, Joe Tessler, Elliot Kramer, and Calvin Lin. 2012. Using Peer Review to Teach Software Testing. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 93–98. https://doi.org/10.1145/2361276.2361295

[60] Stephan Trahasch. 2004. From peer assessment towards collaborative learning. In *34th Annual Frontiers in Education, 2004. FIE 2004.* IEEE, F3F–16.

[61] Anne Venables and Liz Haywood. 2003. Programming Students NEED Instant Feedback!. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20* (Adelaide, Australia) *(ACE '03)*. Australian Computer Society, Inc., AUS, 267–272.

[62] Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. 2015. Learnersourcing Subgoal Labels for How-to Videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Vancouver, BC, Canada) *(CSCW '15)*. Association for Computing Machinery, New York, NY, USA, 405–416. https://doi.org/10.1145/2675133.2675219

[63] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. 2016. AXIS: Generating Explanations at Scale with Learnersourcing and Machine Learning. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale* (Edinburgh, Scotland, UK) *(L@S '16)*. Association for Computing Machinery, New York, NY, USA, 379–388. https://doi.org/10.1145/2876034.2876042

[64] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235

[65] Kimberly Michelle Ying and Kristy Elizabeth Boyer. 2020. Understanding Students' Needs for Better Collaborative Coding Tools. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems Extended Abstracts* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3383068

[66] Alvin Yuan, Kurt Luther, Markus Krause, Sophie Isabel Vennix, Steven P Dow, and Bjorn Hartmann. 2016. Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1005–1017. https://doi.org/10.1145/2818048.2819953

[67] Alvin Yuan, Kurt Luther, Markus Krause, Sophie Isabel Vennix, Steven P Dow, and Bjorn Hartmann. 2016. Almost an Expert: The Effects of Rubrics and Expertise on Perceived Value of Crowdsourced Design Critiques. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1005–1017. https://doi.org/10.1145/2818048.2819953

Received January 2021; revised April 2021; accepted July 2021