Rooms of Their Own: Structured Small-Group Learning in a Realtime Browser-Based IDE

Jacob Roberts-Baca
Stanford University
Stanford, CA, USA
jtrb@stanford.edu

Joshua Delgadillo
Stanford University
Stanford, CA, USA
anon2@university.edu

Chris Piech
Stanford University
Stanford, CA, USA
piech@cs.stanford.edu

Abstract

In this paper, we present *bcode*, a lightweight, browser-based collaborative code editor designed to support small-group learning in both in-person and virtual CS1 courses. Whereas most collaborative IDEs in the literature are built for enterprise contexts, *bcode* draws on research in collaborative and peer-based learning. The platform facilitates real-time, multi-user editing by providing instructors "rooms" in which "groups" of students coauthor code, allowing structured collaboration. The contents of each group are visible to the instructors at all times, enabling live oversight and just-in-time teaching. The platform can also provide LLM-generated hints to help students get unstuck. *bcode*'s design centers on accessibility for students and instructors: only a short alphanumeric code is required to join a room, and the platform runs Python, C++, and graphics code entirely within the browser using WebAssembly, eliminating the cost of server infrastructure.

We deployed *bcode* in two large-scale educational contexts: an in-person CS1/CS2 teaching assistant program at our R1 institution and two global offerings of a CS1 MOOC whose combined enrollment exceeded 30,000. Instructors voluntarily adopted the tool, and we observed high engagement across both deployments, with a regression analysis suggesting *bcode* correlates with increased student participation at a marginal significance. Our evidence also suggests that the flexibility of the room and group system made *bcode* adaptable to a range of teaching formats, including in-person and remote. We discuss design lessons from observed usage patterns and outline future directions, including additional features supporting collaborative pedagogy and follow-up studies to assess causal impacts on participation.

CCS Concepts

• **Social and professional topics** → **CS1**; • **Software and its engineering** → **Pair programming**; *Integrated and visual development environments*; • **Applied computing** → *Computer-managed instruction.*

Keywords

Pair programming, CS1/CS2, IDE, Web browser, WebAssembly

SIGCSE '26, St. Louis, MO

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/10.1145/3770762.3772664

ACM Reference Format:
Jacob Roberts-Baca, Joshua Delgadillo, and Chris Piech. 2026. Rooms of Their Own: Structured Small-Group Learning in a Realtime Browser-Based IDE. In *Proceedings of Technical Symposium on Computer Science Education (SIGCSE '26).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772664

1 Introduction

Collaborative learning is a tried-and-true teaching strategy [6, 32, 37]. For CS1 courses in particular, notorious for their high attrition rates, small-group programming can improve outcomes from final exam scores [28] to retention, graduation rates, and diversity [16]. Small-group programming can also take a think-pair-share form as a tool for active learning, which can increase observed engagement in lecture [29]. Despite these benefits, instructors may not employ this strategy because of the increased time commitment. Some report spending up to 4x longer preparing for small-group activities compared to traditional lectures [6].

1.1 Main Contributions

This paper introduces *bcode*, a realtime, in-browser collaborative coding IDE for students to write, run, and learn to code together. *bcode*, accessible at 106a.vercel.app, is both an IDE for students and a platform for instructors. Instructors organize their students into small groups that collaborate to solve a shared coding problem in realtime. Designed for beginners in a CS1 course, it pairs an unobtrusive and simple user interface with a language-agnostic mechanism for running code locally within the browser. In this paper, we'll present

(1) *bcode*'s novel group-and-room system, which enables small-group collaboration while giving instructors the ability to oversee all code being authored at a glance
(2) A low-cost, language-agnostic mechanism for local code execution powered by WebAssembly/web workers that supports interpreting Python code, compiling and running C++, and displaying animated graphics
(3) The results of instructors using our tool across multiple in-person and remote deployments of a CS1 course with an enrollment exceeding 30,000 with significant engagement demonstrated in both contexts.

1.2 Related Work

A collaborative IDE that assists instructors in implementing small-group pedagogy must allow concurrent code editing by any number of students in a group, enable oversight of each group, and be flexible enough to handle a wide variety of pedagogical approaches.

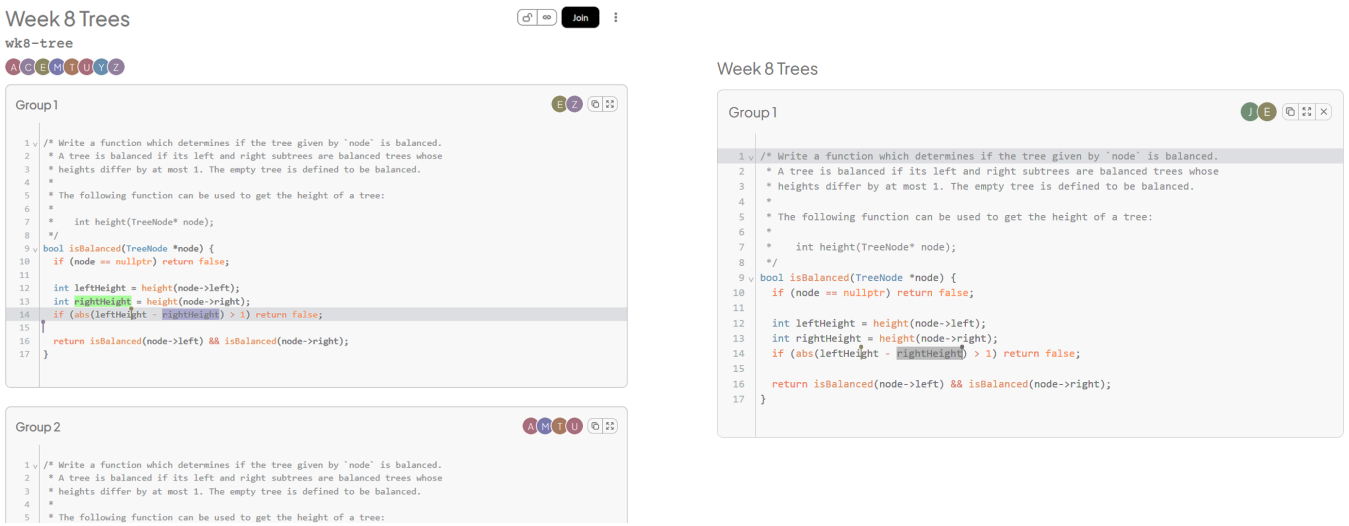

Figure 1: A screenshot of *bcode*. Left, the teacher view, where instructors observe all rooms at a glance. Right, the student view where peers author code within a small group.

Most collaborative editors are not education-specific. Collaborative editors have been in development for the last decade and a half, though most have focused on the enterprise use case. Collabode [14], a seminal early collaborative Java IDE was developed as a tool for "expert users" and focused on minimizing disruptions from errors caused by other editors. Similar tools of the era also aimed to enable collaboration within the common tools of professional SWEs by supporting Java UI frameworks (CoRED [20]) or building Eclipse extensions (ATCoPE [12], CoEclipse [11]). CPROOF allows teams of professionals using different Java IDEs to collaborate using a shared underlying layer behind the IDE-specific extension [21]. More modern tools like JetBrains Code with Me [18], VSCode LiveShare [34], and Repl.it [27] still target a general software engineering use case, but offer additional language support and advanced features like co-debugging, shared terminals, and an in-browser experience. Since these are the dominant collaborative editing tools, many instructors end up adapting them for their needs, despite the lack of pedagogically-grounded features (such as instructor oversight of multiple groups, a simple UI, starter code, or support for their course's libraries) [36].

Many education-focused collaborative IDEs disallow concurrent editing or prescribe specific group sizes and roles. CodeWave, an early pioneer, included several education-focused features such as messaging, feedback annotation, and code replay, but was meant for larger-scale projects and implemented a restrictive locking system that prevented two students from editing the same line at the same time [33]. Similarly, COLLECE [4] manages collaboration through requesting and granting access to specific actions, limiting its usability. PBPCLG assigns students writer, reader, commentator, and supervisor roles with only one writer and supervisor possible at a time [7]. IDEOL designates compiling, running, and debugging as exclusive operations, limiting opportunities for collaboration [25, 31]. A recent tool, PearProgram [3], is designed specifically for distributed pair programming, only supporting groups of two with explicit pilot and co-pilot roles where only the pilot can edit. CodePilot [35] extends the same goal to the end-to-end software life cycle with additional features like issue tracking and GitHub synchronization. While students in pairs were allowed to access features freely, leading to self-organized collaboration styles like traditional pair programming, developer/tester, and developer/assistant, the platform was still limited to only two students at a time. Additionally, only CodeWave offers the instructor live monitoring of student code.

Pedagogically-rooted collaborative IDEs make tradeoffs between feature set and lightweight setup. Several of the early collaborative IDEs required a downloadable desktop tool [11, 12, 21, 33], making the experience more cumbersome for students, particularly in CS1 where an IDE can be daunting [19]. While modern pedagogical tools offer a browser-based experience, many come with very limited language functionality (only HTML/CSS/JS [35], only Python [3]). [35] and [13] offer graphics capabilities, but only through JS. Each of the tools that offer a browser-based experience execute code through an external server, leading to higher costs and barriers to scale. While small demos and very limited tools to execute Python [17] and C++ code [9] have appeared in the literature, none support collaborative editing and execution.

2 Introducing *bcode*

In this section, we will discuss the pedagogical origins of *bcode*, its intended use by students and teachers during instruction, and a few of its important features.

2.1 Contextualizing small-group learning

bcode was originally designed to be used by the in-person teaching assistants (TAs) in the undergraduate CS1/CS2 classes at our institution. In these courses, students attend weekly, hour-long discussion sections with 6-10 of their peers led by an undergraduate TA to review the material presented that week during lectures. A typical

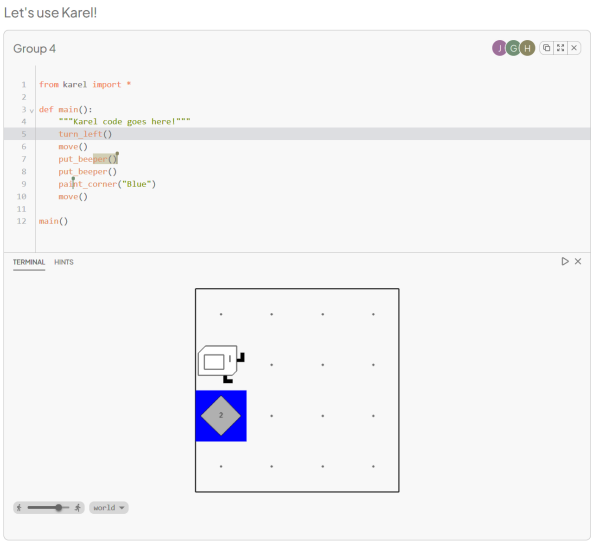

Figure 2: A screenshot of *bcode* showing a student interacting with the Karel CS1 library. Code is run locally on a separate browser thread and independently from other students in the room.

weekly section involves a TA (1) introducing a problem to their students, (2) building verbal consensus on a solution via discussion and pseudocode and (3) implementing that solution in code. *bcode* aims to build on this successful [1, 32, 37] idea by inserting itself into (3). Rather than have students write code individually on their devices or collectively as a class, instructors using *bcode* divide their students into small groups of 2-3 that each independently work to solve the problem at hand.

Small-group instruction of this form offers a number of benefits. For one, students feel more comfortable raising questions, concerns and ambiguities to their peer group than they might among the larger section, reducing the friction to inquiry. Secondly, it fosters a shared responsibility among the group. They mutually depend on one another to make progress, and are less likely to opt out or turn towards external resources when confronting roadblocks along the way. Thirdly, it frees up the instructor to check in with each of the groups individually and tailor their approach to the group's needs rather than aiming at a common denominator. The instructor takes on a supervisory role in which they can direct their attention tactically to those students who need it most in order to ensure a structured learning environment. Finally, group activities present a light-hearted interruption to the usual learning flow that, at times, can feel stiff and monotonous, energizing both students and instructor for the task at hand.

2.2 Designing the IDE

If the previous section presents the *theory* of small-group activities in CS education, then *bcode* is the *practice*. In this section, we will discuss the web interface of the platform, as shown in Figure 1, and how students and teachers might engage with it.

2.2.1 The Teacher View. Before class, instructors log into *bcode* via a Google or GitHub account and create a *room*, which represents a shared space where the class will author code. A room roughly corresponds to a "section" or "classroom" and typically (although not always) will contain the code for a single problem. As part of the configuration of a room, teachers provide a short alphanumeric URL suffix through which students will connect to the room, as well as the number of *groups*, each of which represents a "breakout room" where a small group of students will author code. A language can be chosen to control syntax highlighting (Python and C++ are currently supported), and the number of groups, language, and other configuration options to be discussed, can be changed dynamically during teaching as well.

Once a room has been created, teachers are shown a bird's-eye view of all the groups in the room. As students join and author code, the instructor can see their students' changes as well as who is making the change. Scrolling through this page provides valuable information at a glance of what the state of the class is and what kinds of approaches are being tried, and might prompt additional clarifications or announcements if needed. Additionally, bad actors or the rare unruly student can be removed from the room by IP address.

2.2.2 The Student View. Students join a room by visiting the URL or QR code provided by the instructor and are asked to enter their name and a group to join. *No sign in is required, and besides their name, no identifying information is collected.* This is an important design choice – not only does this greatly lower the friction required for instructors to adopt the tool, but also affords students some peace of mind that their changes will not be tracked or identified to them in any meaningful way after the session has concluded.

Students elect to join a group of their choosing. The names of the students that have already joined a group are shown before joining, and students often choose a group whose members are physically nearby. This design simplifies the join flow over manual assignment by the instructor (although this is still possible on an ad-hoc basis), and allows the platform to scale to larger audiences. At the time of writing this paper, *bcode* has been in regular use during the lectures of a course of ~50 students at our institution. Self-assignment parallelizes peer-finding such that *bcode* can scale to any number, even hundreds, of connected students.

Having joined a group, students are shown a minimal IDE where they can view and author code and see changes by their peers in realtime. The goal of this interface is to be as straightforward and unobtrusive as possible. Built with accessibility in mind, *bcode* is supported on all major browsers and mobile phones. Students in CS1 courses lacking prior experience or familiarity with other IDEs should find *bcode* to be a pleasant experience. *The relative lack of prescriptive education-focused tooling, as found in other collaborative tools [3, 4, 7, 14, 25, 31, 33], is an important design choice.* Rather than overfit to any single approach, we intentionally keep *bcode* lightweight so that instructors may use it to support their own teaching styles. For example, one common strategy seen was to create a dedicated group after the small-group activity had concluded where the instructor would write their own "official" solution to the problem, essentially screen-sharing their work with the students.

2.3 Additional Features

2.3.1 Starter Code. Instructors have the ability to add starter code to a room which is visible to all groups by default. For example, all students might begin with the same function signature for the problem being solved. Changing the starter code will reset all groups, allowing instructors to quickly reuse the same room across problems if desired.

2.3.2 Code Execution. Enabling code execution in the room settings allows students to run their Python/C++ code. Output is displayed in a terminal interface built into the editor, and text input and animated graphical output are supported. Code is run locally for each student, so multiple students may run their code independently and see the output without interfering with one another.

Instructors can configure packages to include when students run their code. For example, Figure 2 shows an animated graphical interface for running Karel [2], a popular library for teaching CS1, which we had students use during our deployment of the tool. Our modular approach, built on top of a novel open-source package manager (github.com/cs106l/runtime), allows instructors at other institutions to create and distribute their own course-specific libraries for students to run, and will eventually support third-party package registries like Python's `pip`.

Behind the scenes, we use WebAssembly [15] and web workers [24] to handle execution entirely locally within a dedicated browser thread. This approach is practically zero-cost, requires no additional hardware other than the student's own device, and runs performantly on mobile devices such as phones and tablets. Furthermore, our approach is language-agnostic and runs the language toolchain (e.g. interpreter, compiler/linker) locally on the student's own machine. System calls (e.g. `open`, `read`, `write`) are rewritten to use a virtual in-memory filesystem, and shared memory buffers are used extensively to communicate data efficiently between the browser's main thread and the code thread in order to enable performant execution of animated graphical programs. This represents a departure from previous work [17], which aimed to run all student code on the browser main thread. We believe this will scale better to new languages, as already shown by its native ability to compile and run C++ code, and ultimately yield superior performance by utilizing multi-threaded execution.

2.3.3 Templates. An instructor may copy a *template link* to share the configuration settings of a room with their colleagues. This is useful to enable quick adoption of *bcode* among course staff. For example, during the 2025 offering of the MOOC we partnered with, course administrators used room templates to share pre-configured packages and starter code with ~100 teaching assistants.

2.3.4 LLM Hints. Especially in larger sections, it may be difficult for one instructor to support all groups alone. When a student is blocked on access to the instructor, they lose time that could have been better spent engaging with the problem. To address this, we have added an opt-in, LLM-driven hints feature where students may request a hint to get "unstuck". We generate hints by feeding the student's code, starter code, and prior hints as context to an OpenAI o3 model. Students can only request up to a maximum of three hints with a two minute cooldown (the specific numbers are configurable by the instructor). In an era where the appropriate role of LLMs in education is debated [8, 22, 26, 30], we see this feature as a happy medium between receiving no help at all and allowing unrestricted access to an LLM.

2.3.5 Locking Rooms. At some point, the small-group activity must end and the class must move on. To enable a quick transition, instructors may lock the room to prevent edits from students. While the room is locked, the instructor can still make changes–for example to share their own approach with students in realtime. All code is persisted to a database, and students may continue to visit the room afterwards for asynchronous review without concern that the code will have changed.

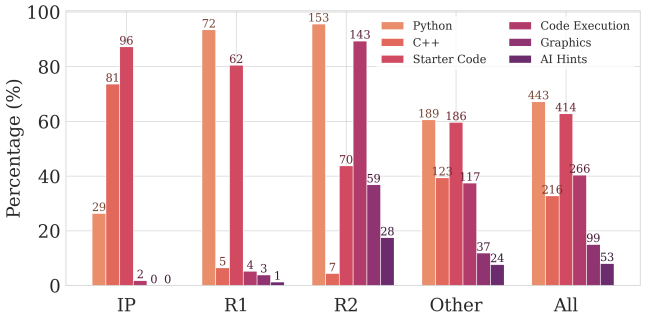

Figure 3: A breakdown of the proportion of rooms that utilized various *bcode* features across cohorts during our deployment of the tool. Raw room counts are included above each bar.

3 Scaling to Thousands of Learners

We deployed *bcode* across two broad learning contexts:

(1) *CS198*, a cohort of ~100 in-person undergraduate teaching assistants supporting ~1000 CS1/CS2 students at our institution
(2) The 2024 and 2025 offerings of *Code in Place*, a MOOC that teaches our institution's CS1 class at scale (two cohorts totaling ~1800 TAs and ~30000 students)

Instructors in both contexts were presented the tool and offered to voluntarily use it in their weekly, 50-minute discussion sections. We show a breakdown of feature usage across each cohort in Figure 3, and the table below summarizes the context, number of TAs who used *bcode*, and total number of rooms created per cohort.

Name	Context	# TAs	# Rooms
CIP4	Virtual	53	77
CIP5	Virtual	86	160
CS198	In-person	30	110
Other[1]	N/A	113	312

Each cohort was given a short presentation on how to use *bcode*. A random subset of TAs was invited to use the tool for *CIP4*; for *CS198* and *CIP5*, all TAs were invited. TAs participating in *CIP5*

[1] *Other* contains data from users who didn't correspond to either *CS198* or *Code in Place*. We suspect that many of these users were curious students of the instructors!

received more targeted exposure on how to use the tool compared to other cohorts, including weekly help sessions and a user guide document.

3.1 Response from instructors

Throughout the deployment, we have consistently heard praise from instructors, course staff, and students who have used *bcode*. In a volunteer survey offered to *CIP4* and *CIP5* TAs ($N = 24$), TAs noted that the tool reduced pressure on students to share screens, enabled real-time collaboration across breakout rooms, and allowed instructors to monitor group progress more efficiently and equitably. When asked "What did you like about the tool?", TAs responded that it "allowed everyone to code live, which was fun and encouraged participation from every member" and that "students finally started talking with each other and discussing ideas in breakout rooms." When asked to rate how likely they were to recommend *bcode* to another TA/instructor on a scale from 1-10, respondents averaged 9.12/10.

3.2 How groups were used

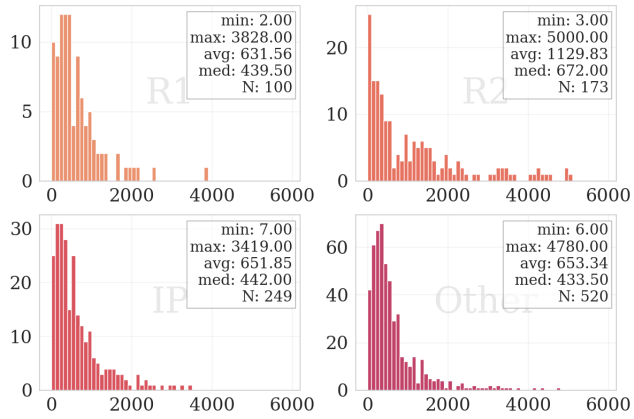

Figure 4: A histogram of the count of textual modifications per used group. The vertical axis represents group count, the horizontal axis the number of modifications in that group.

We were interested to see how instructors provisioned groups in their rooms, and how those groups were utilized. As students write code, their local clients persist their changes to the database every 5 seconds. We define a *group* to be *unused* if it received fewer than two updates overall, and a *room* to be unused if every one of its groups was unused. We find that across cohorts, a large percentage of rooms went unused (30.1%, 30.8%, 27.1%, 29.3%)[2], which we believe corresponds with rooms used for testing but not for instruction. After filtering out such rooms, we found that the count of groups per room varies somewhat across cohort (medians 2, 1, 3, 2) with slightly right-skewed distributions (means 2.0, 1.9, 3.2, 2.8)[2]. *Code in Place* TAs typically had usage in only one or two groups whereas *CS198* TAs had usage in 3.

[2]The numbers in parentheses correspond to the *CIP4*, *CIP5*, *CS198*, and *Other* cohorts, respectively.

After filtering out unused rooms from the dataset, we find that a majority of rooms had every one of their groups utilized (60.8%, 81.8%, 64.1%, 56.4% of rooms per cohort)[2]. In other words, a majority of rooms with at least some usage in one group showed usage across all of their groups. This indicates that instructors tended to choose an appropriate group count to meet student demand, despite different patterns in group use across contexts.

3.3 Student engagement in groups

As stated previously, students' updates are persisted to a database every 5 seconds, and we can track the evolution of each group's code over time, allowing us to compute the total count of textual modifications (insertions and deletions of characters) per group. Figure 4 shows a histogram of such counts across *used* groups (i.e. groups with at least two updates). It is important to note that this is not the number of characters written, but rather the sum of all additions and deletions over the course of a group's lifetime. Among all cohorts, *CIP5* exhibited the most engagement, with a median of 672 modifications per group compared to the other cohorts, which all hovered around 440. We trace this trend to the additional instruction *CIP5* TAs received on how to use *bcode* effectively in their sections.

3.4 *bcode* may increase engagement

After verifying that *bcode* was used constructively, we analyzed its effects on engagement. The randomized invitation emails in *CIP4* form a quasi-experimental setup suitable for this analysis. We defined the following four groups of TAs:

- *Used* (53) — used *bcode* in their section, measured by having an account and having created at least one room
- *Invited* (369) — received the email invitation to use *bcode*
- *Non-invited* (191) — did not receive the email to use *bcode*
- *All* (560) — Every *CIP4* TA

For each group, we examined:

- *Attendance drop* — the average percentage point decrease in attendance between the Week 5 and Week 6 sections during which *bcode* was used (lower is better)
- *Learner talk time* — the average percentage of section dialogue time belonging to students (as compared to the TA), measured by Zoom meeting transcripts
- *Questions asked* — the average number of questions asked by students during the section, measured by AI transcript analysis as done in [10]

There is no panacea for measuring engagement [23], but these three proxies are suitable given the data *Code in Place* provided. *Code in Place* uses *Section attendance* internally to gauge community health, while *Talk time* and *Questions asked* reflect student participation and correlate with high-leverage instructional strategies such as uptake [10].

Metric	*All*	*Non-invited*	*Invited*	*Used*
Attendance Drop (%)	2.55	3.03	2.30	1.28
Talk Time (min)	15	14	16	18
Questions Asked	7.0	6.8	7.1	7.6

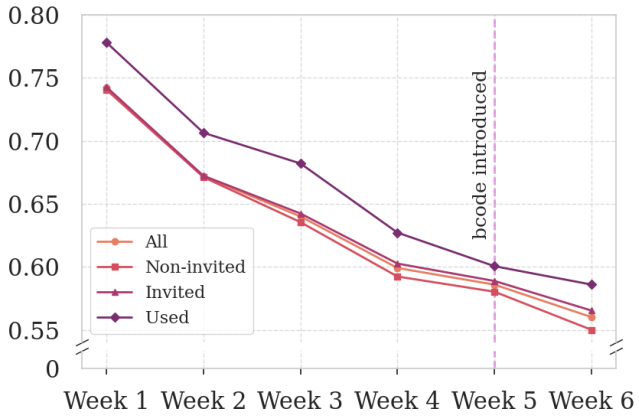

Figure 5: Average section attendance rates in *CIP4* by experimental group. The *CIP4 Used* group maintained higher attendance rates across all weeks.

The table above shows the metrics collected. Across each statistic, TAs in the *Invited* and *Used* groups performed best. While the margins are highest between the *Used* and *Non-invited* groups, centering the analysis here is misleading because *bcode* usage was voluntary. TAs who opted to take up *bcode* after receiving an invitation could be more experienced and have superior section statistics anyway. Figure 5 suggests this, as the *Used* group's attendance exceeded that of the other groups before *bcode*'s introduction.

Since non-use of *bcode* among invitees was not random, we conducted the following Intent-to-Treat regression [5] to measure the effect of being invited to use *bcode* on student engagement, while controlling for existing differences in TA skill level:

$$\textit{Talk time} \sim \textit{Invited} + \textit{Covariates}$$

Talk time was chosen as the most stable metric to regress because attendance can fluctuate due to external factors (e.g., a student's schedule or illness) and counting questions is vulnerable to AI analysis errors. Additionally, talk time is the metric *bcode* was most directly intended to increase – grouping students should get them to engage more.

We regress this against *Invited*, an indicator for whether the TA received an email advertising *bcode* and *Covariates*: whether or not the TA is based in the US, whether or not they are a minor, the average student talk time over the previous sections, whether the TA was a former student or TA, when their section takes place, and what proportion of the sections the TA went to. We chose covariates that indicate how experienced a TA is to better isolate the effect that *bcode* has over innate teaching ability.

We observed a coefficient of 1.7525 on *Invited* which points to an expected 1.7525% increase in talk time when an TA is introduced to *bcode*. With $p = 0.098 < 0.1$, these results are not conclusive, but certainly promising.

4 Discussion

Across the three cohorts in which it was deployed, *bcode* received positive feedback, facilitated a variety of small-group activity types, and showed potential for increasing engagement. Section 3 and Figure 3 demonstrate the flexibility of *bcode*, with some instructors choosing to make one group for each student, others small groups, and others, especially those in *CIP5*, a single group where all students participate together in a shared editor.

CIP5 saw higher engagement levels (as measured by textual modifications per group), smaller group counts, and higher utilization of rooms than *CIP4* and *CS198*. While we cannot say with certainty, we hypothesize that more targeted TA instruction on the tool during *CIP5* led to higher engagement among that cohort's students. During the pre-section workshops, *CIP5* TAs were encouraged to use *bcode* with one group before advancing into multiple groups, which may account for the observed median of one group per room. We also believe that Zoom preparation sessions, a user-guide with helpful information, and access to forum support where TAs could raise concerns, may have led these TAs in aggregate to use *bcode* in a way that was more conducive to student engagement. This hints at a tradeoff: though *bcode*'s interventions provide flexibility across a range of learning contexts, its effectiveness may depend on structured guidance and support for TAs.

bcode's compromise is to provide first-class support and sensible defaults for the small-group use case, while allowing customization to broaden the base of instructors for whom it is useful. Rather than pursuing an all-in-one design, we intentionally kept *bcode* lightweight–an approach that has enabled a smooth transition from its original in-person use in *CS198* to remote deployment in *CIP4* and *CIP5*. Notably, *bcode* omits several commonly requested features, such as built-in student–teacher communication, video conferencing, and automated group assignment. By delegating these responsibilities to the instructor, *bcode* allows the instructor to integrate it into their existing workflow.

4.1 Future Work

This study's finding of marginally significant results justifies a larger-scale experiment on *bcode*'s effects. A future experiment involving a larger user base and mandatory, rather than opt-in usage for the treatment group might enable more rigorous causal inference, helping to more conclusively determine whether *bcode* improves student engagement and learning outcomes.

While we desire for *bcode* to remain a general and lightweight tool, there are a number of functionality extensions we are interested in pursuing that may be integrated as modular plugins or room-specific "widgets". For example, we are interested in providing support for: course-specific test harnesses and libraries, steppable and/or breakpoint debugging, interactive graphics (e.g. mouse/keyboard input), networking, a student-visible filesystem, and dynamic run-time code analysis, such as visualizing the stack and memory layout of the running program. Our aim is that *bcode* can act as a general purpose platform for pre-flighting and researching novel educational tooling in a collaborative setting at scale.

Acknowledgments

To [anonymized] for initially pushing us to distribute *bcode* to a wider audience, and [anonymized] for seeing a potential in it that we could not have on our own!

References

[1] Anonymous Authors. 2025. Anonymous Undergraduate TA Program. *Anonymous Journal* 99, 1 (March 2025), 999. doi:XX.XXXX/XXXXXX.XXXXXX

[2] Byron Weber Becker. 2001. Teaching CS1 with karel the robot in Java. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, USA) *(SIGCSE '01)*. Association for Computing Machinery, New York, NY, USA, 50–54. doi:10.1145/364447.364536

[3] Maxwell Bigman, Ethan Roy, Jorge Garcia, Miroslav Suzara, Kaili Wang, and Chris Piech. 2021. PearProgram: A More Fruitful Approach to Pair Programming. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 900–906. doi:10.1145/3408877.3432517

[4] Crescencio Bravo, Rafael Duque, and Jesús Gallardo. 2013. A groupware system to support collaborative programming: Design and experiences. *Journal of Systems and Software* 86, 7 (2013), 1759–1771. doi:10.1016/j.jss.2012.08.039

[5] Tom Brody. 2016. Chapter 8 - Intent-to-Treat Analysis Versus Per Protocol Analysis. In *Clinical Trials (Second Edition)* (second edition ed.), Tom Brody (Ed.). Academic Press, Boston, 173–201. doi:10.1016/B978-0-12-804217-5.00008-4

[6] Ricardo Caceffo, Guilherme Gama, and Rodolfo Azevedo. 2018. Exploring Active Learning Approaches to Computer Science Classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 922–927. doi:10.1145/3159450.3159585

[7] Atef Chorfi, Djalal Hedjazi, Sofiane Aouag, and Djalleleddine Boubiche. 2020. Problem-based collaborative learning groupware to improve computer programming skills. *Behaviour & Information Technology* 41 (07 2020), 1–20. doi:10.1080/0144929X.2020.1795263

[8] Marian Daun and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, Turku Finland, 110–116. doi:10.1145/3587102.3588815

[9] Joshua Delgadillo and Chris Piech. 2025. A Fully Client-Side C++ IDE for CS1 Using WebAssembly. In *Stanford CS191W Winter '25*. ACM, Palo Alto, CA, 1–7. https://cs191.stanford.edu/projects/Delgadillo2025.pdf Accessed: 2025-06-28.

[10] Dorottya Demszky, Jing Liu, Heather C. Hill, Dan Jurafsky, and Chris Piech. 2024. Can Automated Feedback Improve Teachers' Uptake of Student Ideas? Evidence From a Randomized Controlled Trial in a Large-Scale Online Course. *Educational Evaluation and Policy Analysis* 46, 3 (2024), 483–505. doi:10.3102/01623737231169270 arXiv:https://doi.org/10.3102/01623737231169270

[11] Hongfei Fan and Chengzheng Sun. 2012. Achieving integrated consistency maintenance and awareness in real-time collaborative programming environments: The CoEclipse approach. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 94–101. doi:10.1109/CSCWD.2012.6221803

[12] Hongfei Fan, Chengzheng Sun, and Haifeng Shen. 2012. ATCoPE: any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development. In *Proceedings of the 2012 ACM International Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '12)*. Association for Computing Machinery, New York, NY, USA, 107–116. doi:10.1145/2389176.2389194

[13] Soroush Ghorashi and Carlos Jensen. 2016. Jimbo: a collaborative IDE with live preview. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering* (Austin, Texas) *(CHASE '16)*. Association for Computing Machinery, New York, NY, USA, 104–107. doi:10.1145/2897586.2897613

[14] Max Goldman, Greg Little, and Robert C. Miller. 2011. Collabode: collaborative coding in the browser. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering* (Waikiki, Honolulu, HI, USA) *(CHASE '11)*. Association for Computing Machinery, New York, NY, USA, 65–68. doi:10.1145/1984642.1984658

[15] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web up to speed with WebAssembly. *SIGPLAN Not.* 52, 6 (June 2017), 185–200. doi:10.1145/3140587.3062363

[16] Cecily Heiner. 2017. Coding, collaborating, and creating: a trio of pedagogical practices to improve instruction and retention in CS1. *J. Comput. Sci. Coll.* 33, 2 (Dec. 2017), 93–99.

[17] Thomas Jefferson, Chris Gregg, and Chris Piech. 2024. PyodideU: Unlocking Python Entirely in a Browser for CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 583–589. doi:10.1145/3626252.3630913

[18] JetBrains Marketplace. 2023. Code With Me. https://plugins.jetbrains.com/plugin/14896-code-with-me Accessed: 2023-11-14.

[19] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83–137. doi:10.1145/1089733.1089734

[20] Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, Tommi Mikkonen, and Marc Englund. 2012. CoRED: browser-based Collaborative Real-time Editor for Java web applications. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work* (Seattle, Washington, USA) *(CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 1307–1316. doi:10.1145/2145204.2145399

[21] Yifan Ma, Zichao Yang, Brian Chiu, Yiteng Zhang, Jinfeng Jiang, Bowen Du, and Hongfei Fan. 2021. Supporting Cross-Platform Real-Time Collaborative Programming: Architecture, Techniques, and Prototype System. In *Collaborative Computing: Networking, Applications and Worksharing*, Honghao Gao and Xinheng Wang (Eds.). Springer International Publishing, Cham, 124–143.

[22] Stephen MacNeil, Juho Leinonen, Paul Denny, Natalie Kiesler, Arto Hellas, James Prather, Brett A. Becker, Michel Wermelinger, and Karen Reid. 2024. Discussing the Changing Landscape of Generative AI in Computing Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1916. doi:10.1145/3626253.3635369

[23] B Jean Mandernach. 2015. Assessment of student engagement in higher education: A synthesis of literature and assessment tools. *International Journal of Learning, Teaching and Educational Research* 12, 2 (2015), 1–14.

[24] MDN Web Docs. 2023. Web Workers API - Web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API. Accessed: 2025-07-02.

[25] Vu Nguyen, Hai H. Dang, Kha N. Do, and Thu D. Tran. 2014. Learning and practicing object-oriented programming using a collaborative web-based IDE. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. 1–9. doi:10.1109/FIE.2014.7044141

[26] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. 2023. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '23)*. Association for Computing Machinery, New York, NY, USA, 108–159. doi:10.1145/3623762.3633499

[27] Replit. 2025. Replit: Collaborative Browser-Based IDE. https://replit.com/ Accessed: 2025-06-27.

[28] Oscar Revelo Sánchez, Alexander Barón Salazar, and Manuel Ernesto Bolaños. 2025. Collaborative Programming as a Didactic Learning Strategy in CS1 Courses. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* 20 (2025), 32–38. doi:10.1109/RITA.2025.3541541

[29] Anshul Shah, Thomas Rexin, Fatimah Alhumrani, William G. Griswold, Leo Porter, and Gerald Soosai Raj. 2025. An Empirical Evaluation of Active Live Coding in CS1. *ACM Trans. Comput. Educ.* (June 2025). doi:10.1145/3743686 Just Accepted.

[30] Miriam Sullivan, Andrew Kelly, and Paul Mclaughlan. 2023. ChatGPT in higher education: Considerations for academic integrity and student learning. *Journal of Applied Learning & Teaching* (Jan. 2023). doi:10.37074/jalt.2023.6.1.17

[31] Hai T. Tran, Hai H. Dang, Kha N. Do, Thu D. Tran, and Vu Nguyen. 2013. An interactive Web-based IDE towards teaching and learning in programming courses. In *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*. 439–444. doi:10.1109/TALE.2013.6654478

[32] Jonathan G. Tullis and Robert L. Goldstone. 2020. Why does peer instruction benefit student learning? *Cognitive Research: Principles and Implications* 5, 1 (April 2020), 15. doi:10.1186/s41235-020-00218-5

[33] Jason Vandeventer and Benjamin Barbour. 2012. CodeWave: a real-time, collaborative IDE for enhanced learning in computer science. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 75–80. doi:10.1145/2157136.2157160

[34] Visual Studio Marketplace. 2023. Microsoft Visual Studio Live Share. https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare Accessed: 2023-11-14.

[35] Jeremy Warner and Philip J. Guo. 2017. CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1136–1141. doi:10.1145/3025453.3025876

[36] 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* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3334480.3383068

[37] Daniel Zingaro. 2014. Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 373–378. doi:10.1145/2538862.2538878