Developing Novice Programmers' Self-Regulation Skills with Code Replays

Benjamin Xie
Institute for Human-Centered Artificial Intelligence, McCoy Family Center for Ethics in Society
Stanford University
Stanford, California, USA
benjixie@stanford.edu

Jared Ordona Lim*
The Information School
University of Washington, Seattle
Seattle, Washington, USA
jorlim7@uw.edu

Paul K.D. Pham*
The Information School
University of Washington, Seattle
Seattle, Washington, USA
pkdpham@uw.edu

Min Li
College of Education
University of Washington, Seattle
Seattle, Washington, USA
minli@uw.edu

Amy J. Ko
The Information School
University of Washington, Seattle
Seattle, Washington, USA
ajko@uw.edu

ABSTRACT

Learning programming benefits from self-regulation, but novices lack support for developing these skills of cognitive control. To support their development, we designed Code Replayer, an online tool that enables novice programmers to practice programming and then replay their coding process to reflect and identify process improvements. To evaluate the impact of replaying code on self-regulation, we conducted a formative qualitative evaluation with 21 novice programmers who used Code Replayer to practice writing code. We found that after watching code replays, participants more frequently interpreted problem prompts and planned their solutions, two crucial self-regulation behaviors that novices often overlook. We interpret our results by focusing on two focal points in the design of code replays as a programming self-regulation intervention: interpreting pauses in replays and ensuring replays of struggle are more informative and less detrimental.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in HCI**; • **Social and professional topics → Computing education**.

KEYWORDS

self-regulation, process data, keystroke logs, metacognition, qualitative methods, computing education,

ACM Reference Format:
Benjamin Xie, Jared Ordona Lim, Paul K.D. Pham, Min Li, and Amy J. Ko. 2023. Developing Novice Programmers' Self-Regulation Skills with Code Replays. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1 (ICER '23 V1), August 07–11, 2023, Chicago, IL, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3568813.3600127

*Both authors contributed equally to this research.

1 INTRODUCTION

Programming is cognitively demanding because it requires special notation, uses abstractions, and lacks direct manipulation, making programs difficult to inspect [4]. This burden is especially high for novice programmers, who are still learning the syntax of programming languages while also learning core concepts in computing [65].

One skill that helps alleviate the burden of programming is *self-regulation*. Effective self-regulation helps a programmer monitor their cognitive processes as they understand a problem, search for analogous problems, search for potential solutions, identify goals, implement a solution, and evaluate their solution [32]. However, novices are often unaware of the need for self-regulation, its relationship with programming, or find it too taxing to simultaneously learn programming-specific knowledge as well as self-regulation skills [11, 49]. One explanation for this lack of awareness is that novice programmers lack scaffolded opportunities to develop these skills [11, 33, 35, 36].

Existing approaches to developing self-regulation skills focus on explicit instruction from an expert instructor or an emphasis on specific self-regulation behaviors. Prior work has explored explicit instruction to develop self-regulation skills through personalized tutoring (e.g. [32]) and live-coding demonstrations [44, 52, 55, 58], but these approaches require interactions with instructors who have programming and self-regulation expertise. Other work has explored using explicit prompting, such as cuing novices to understand the problem prompt prior to writing code [36, 49]. These approaches assume a formal learning environment such as an introductory computing course. However, people often learn programming in informal and online settings [5, 54]. Therefore, existing approaches to teaching self-regulation in programming may not transfer to informal and online learning contexts.

One opportunity to develop self-regulation skills across learning contexts is to focus novice programmers on *self-observation*. Self-observation is a crucial component to self-regulation and consists of learners tracking their own performance, processes, and outcomes and reactively change their strategies and behaviors [69, 71]. For example, effective self-observation can help students recognize unproductive struggle in a programming task and reassess their strategies [32].

Self-observation is more effective with the support of recordings. Recordings can help learners self-observe by making them aware of things that could have gone unnoticed and reducing the cognitive demand that comes with recalling prior actions [1, 71]. Without recordings, learners often rely on recall from memory to self-observe. This can be cognitively taxing, unreliable, or not specifically relevant to cognitive processes [2, 71].

For a recording to support self-observation, it must be convenient to create, not rely on recall or self-report, and be closely connected to cognitive processes. Recordings can promote learning by creating more awareness of the importance of self-regulation and its stages (e.g. goal setting), reminding learners of different stages, stimulating reflection, and recognizing the interrelatedness of different stages at every phase of a learning process [40, 56]. Diaries or journals are common recording techniques (e.g. [35, 56]). However, these recording techniques still require recall and self-report, making them potentially difficult to create, unreliable, and at risk of overlooking or misrepresenting important parts of cognitive processes [2, 10, 71]. Existing code visualization tools to support programming education (e.g. [15, 66]) tend to visualize program behavior such as code execution and/or record intermittent snapshots of code execution. While important to learning about the programming domain (e.g. for debugging [25]), intermittent recordings of code changes do not necessarily align with the cognitive processes of writing code, which includes phases such as changing strategies or debugging [33]. Prior studies have identified that visual representations of code writing processes could help students [9] and instructors [59] consider problem solving processes, but these studies did not investigate the impact of visualizations on self-regulation skills.

In this paper, we propose using replays of code edits (referred to as *code replays*) as recordings for novice programmers to self-observe with. We developed a *Code Replayer* tool to automatically create recordings of novice programmers' code edits. This makes recordings effortless to create, more reliable than recall or self-report alone, and closely tied to their cognitive processes because they show entire code editing processes.

We then conducted a formative study to understand how replays of coding processes could support self-regulation skills, investigating two research questions:

(1) How does reflecting on code replays affect novice programmers' self-regulation behaviors?
(2) How do novice programmers use code replays differently?

This study contributes a formative, empirical evaluation of how reflecting on replays of code writing processes can develop novice programmers' self-regulation skills. These findings can inform the design of pedagogy and tools that embed opportunities to develop self-regulation skills into online programming practice environments.

2 PRIOR WORK

2.1 Theoretical Foundation to Connect Self-Regulation, Metacognition, & Self-Recording

Self-regulation and metacognition are related constructs that are part of learning and skill development processes. Typically, *metacognition* refers to learners' knowledge, awareness, and regulation of their thinking and cognitive control (e.g. which strategies are most effective for learners) [71]. Related to that is *self-regulation*, which typically describes learners' process of cognitive control [68, 71].

To connect self-recording to self-regulation and metacognition, we applied Zimmerman's three phase model of self-regulated learning [71]. This model frames self-regulation as iterating between the three phases of *forethought*, *performance*, and *self-reflection*. Within this model, performance includes metacognition to monitor cognitive processes, and self-recordings are a tool to facilitate self-evaluation.

In the forethought phase, learners decompose a problem into elements and develop a plan based on prior knowledge of these elements. This phase involves task analysis: breaking a programming problem down into sequential steps and planning out actions. This goal setting produces an explicit feedback loop that then requires self-evaluation. The forethought phase also includes considerations of sources of self-motivation, but motivational factors were not a focal point of this study and we attempted to control for this with similar incentives for participation.

In the performance phase, learners must exercise self-control and self-observation. Self-control methods include both task-specific and general strategies for addressing specific components of a task. Examples of general self-control strategies include self-instruction (e.g. self-questioning when programming), imagery (e.g. converting textual information into mental diagrams, flow charts, or other images), help-seeking methods (e.g. asking an instructor for assistance), and interest incentives (e.g. gamifying a task to enhance motivation).

Learners must adapt strategies in the performance phase according to intended outcomes, making *self-observation* crucial. Self-observation consists of metacognitive monitoring and self-recording. Metacognitive monitoring refers to mental tracking of one's performance, processes, and outcomes. Within this paper, we will refer to metacognitive monitoring as *metacognition*. Releated to self-observation is self-recording, creating formal records of learning processes or outcomes, such as notes or reflection journals [35]. These self-recordings can support metacognition by increasing reliability, specificity, and time span of self-observations while decreasing students' reliance on recall [70].

The final phase is self-reflection, where learners evaluate and react to their own performance. Evaluations can include comparing performance to existing standards (e.g. prior performance, mastery of skills, comparison with peers) or making causal attributions to explain performance. Reactions can include cognitive and affective reactions to evaluations as well as adaptive decisions in which students consider modifying their strategies for future cycles of learning.

The three phases of forethought, performance, and self-reaction affect each other cyclically. Within this study, we considered how using code replays as a self-recording tool could support metacognitive monitoring. Zimmerman's self-regulation theory suggests that this improved monitoring could provide richer evaluations in the self-reflection phase, which could then lead to improvements to task analysis in the forethought phase and applications of strategies in the performance phase [71].

2.2 Prior Uses of Process Recordings

The most relevant prior work is Ditton et al. [9], which explored the impact of advanced CS students watching playbacks of their code writing process. They found that students preferred the playback over reviewing static code, but found mixed evidence about its usefulness. Multiple students mentioned that playbacks helped them see or visualize their process, but the study did not investigate thought process in detail or connect it to self-regulation. We build upon this prior work by evaluating the effect of code writing replays/playbacks on different stages of metacognitive processes for novice programmers.

Within programming education, live-coding is a common technique to teach students programming process; for example, Computer Science (CS) instructors might design and implement a program during lecture [44], where students observe instructors program or code alongside instructors [52]. While prior work has found evidence that live-coding helps teach programming process [52, 55, 58], it primarily focuses on having students observe or model instructors' process [58]. Therefore, a limitation to live-coding to develop self-regulation skills is that there is a lack of support for the transition from students observing instructors' process to regulating their own. Recent trends of live-streaming code for programming education have minimal evidence of effectiveness and also similarly focus on learners observing someone else's programming process rather than reflecting on their own [13, 16].

Multiple studies in non-programming domains have investigated the use of recordings, suggesting a need for being selective around their collection and use. Lippmann Kung and Linder [28] analyzed classroom dialog to study metacognition in physics classrooms. In psychotherapy, therapists found that reviewing session recordings was beneficial to their practice, but also identified a need for selectivity and sensitivity towards recordings [2]. A study in primary school math education found that students who chose to replay their past actions after struggling performed better in post-test [67]. In physical education, one study found that letting participants choose when they wanted videotaped replay benefited their overall performance when compared to regularly scheduled feedback [20]. In a study with music teachers, almost all felt video recordings of themselves improved their ability to conduct, but their perspectives varied in regards to frequency of feedback [37].

3 METHOD: 21 NOVICES USING PYTHON PRACTICE TOOL AND SEEING REPLAYS OF CODE EDITING

We conducted a study where 21 participants solved programming problems and watched replays on some problems in a 1-2 week, self-paced online course comprised of three modules in the Code Replayer web application. This study was approved by an Institutional Review Board (IRB) prior to recruitment.

3.1 Critical Self-Reflexivity and Positionality

This research required analysis of participants' verbal and written discourse, which are situated within cultural and societal norms. Therefore, we acknowledge our assumptions and values. By doing so, we follow critical approaches to quantitative methods which require researchers "to engage in critical self-reflexivity as a necessary first step for the long journey of deracializing statistics" [14].

We acknowledge the bias that comes from this research and research community being situated in Western, educated, industrialized, rich and democratic societies [18, 27]. In particular, our analysis was conducted in English with English-speaking participants. This is both exclusionary as well as potentially biasing in our analysis, as we have may misrepresented the data of participants with less English fluency.

We also acknowledge that our study focused on qualitative data, which is constructed and therefore not objective [17]. We engaged in a creative process that was heavily theory and technology laden. Therefore, we depict the complexity our data collection and analysis processes to make transparent potential errors alongside insights.

3.2 Study Participants: Novice Programmers from Two Courses

We recruited from two similar introductory programming (CS1) Python courses, Course A and Course B, partnering with their instructors. Course A was taught at the computer science department of a large public research university, while Course B was taught at a community college that was an order of magnitude smaller. Course A consisted of about 100 students, while Course B had about 25 students. Both institutions were located in the same urban region, which had a strong presence of multiple technology companies. Of our 21 participants, 19 were from Course A while the remaining two were from Course B. This discrepancy was largely because of size of student population and timing of study relative to the academic terms. Nevertheless, we felt the inclusion of students from a two-year institution was important, as their experiences are often excluded from computing education research [22]. Participation was voluntary, so there was self-selection bias.

We reported demographics for our 21 participants below. For free-response questions (gender, ethnicity, language), we used terminology provided by participants but bucketed their responses to prevent re-identifiability.

- 10 as woman or female, 9 as man or male, 1 genderqueer, and 1 declined to provide gender.
- 14 as Asian (including Korean, South Asian, Chinese, Cambodian), 2 as white or Caucasian, 1 as Hispanic/Latino, 1 as

"Mixed," and 3 declined to provide racial or ethnic information.

- 11 primarily spoke a language at home that differed from the language of instruction (English). Familial languages included Chinese (Mandarin, Cantonese), Hindi, Spanish, Thai, and Vietnamese. Nine others primarily spoke English at home, with 1 not disclosing.
- 14 took 0-1 prior programming courses, while 7 reported taking 2 or more prior programming courses.
- 15 were working towards a Bachelor's degree, 2 a graduate degree (Master's or Ph.D.), 1 an Associate's degree, 1 a GED, and 2 not working towards a degree.
- 3 were first-generation students (parents did not attend college), 18 others were not, and 1 was unsure.

Participation in this study was to serve as a "refresher" to students who had previously taken CS1 course or formative assessment to those currently taking their first.

We compensated participants $75 for completion of the online portion of the study, which participants reported taking 2-12 hours. Interviewed participants received an additional $25. These rates were commensurate with minimum wage at the study location. Participants in Course A also received participation credit. To avoid coercion, students in Course A who did *not* participate in this study had other opportunities to earn the same credit.

3.3 Overview of Study Design: Self-Paced Online Python Practice w/ Code Replays

We followed the following principles when designing this study:

1. **Low stakes, formative experience**: Learners should perceive Code Replayer as a low-stakes, formative experience to practice programming skills and develop metacognitive skills through reflection.
2. **Developing self-regulation skills requires scaffolding**: To develop novice programmers' self-regulation skills, we must design explicit scaffolded opportunities. This follows prior research suggests that novice programmers rarely develop self-regulation skills without explicit scaffolding or instruction [33].
3. **Opportunities for timely, scaffolded reflection**: Reflecting on recordings is a crucial part of the self-regulation process [71], but it also must be done so in a timely and proximal manner to ensure beneficial recall and reflection [10].
4. **Proximal, perceived value of participation**: Learners should perceive tangible and timely value for their participation.

Figure 1 provides an overview of the study flow. Participants were required to complete each phase before continuing on to the next one. In the remainder of this section, we describe each part of the study.

3.4 Pre-Survey Design: Prior knowledge and MSLQ measurement.

After signing up, participants completed an online pre-survey. We asked participants how many prior programming classes they had completed before and how many hours per week each participant spent doing work (defined as time spent on classwork, homework, and studying combined) in the class. We then asked participants, "When you are given a programming problem to solve, how do you usually start?" We then asked participants to measure their self-regulation using the nine-item metacognition sub-scale from the Motivated Strategies for Learning Questionnaire (MSLQ) [45, 47], further described in Section 4.2

3.5 Design of Online Practice Environment with Code Replays

Participants practiced writing Python across three practice sets: PS1 (baseline), PS2 (intervention), and PS3 (endline). Each practice set contained seven unique items that were the same for all participants (further described in Section 3.5.1). Only PS2 had code replay features.

In all three problem sets, participants responded to two reflection prompts after each item. The reflection prompts were designed to record the self-regulation that participants engaged in and encourage self-regulating behaviors. While submitted practice items could be incorrect, participants were required to complete reflection prompts in all items to complete the study. These prompts came from prior work on programming self-regulation [33] and were refined through piloting with a student from the target population. Figure 2e shows an example of two reflection prompts that followed every item in PS2. We qualitatively coded the following prompt that appeared in PS1 and PS3: *How did you approach this problem?*

PS1 provided a baseline for programming and self-regulation behavior before being primed about metacognition and using the Code Replayer. The code replay feature was then introduced to participants for use in PS2. After completing a practice item and answering the reflection prompts, participants then used the replay tool to see a replay of their programming process. They answered additional reflection prompts to encourage engagement with the replay tool and collect data about participants' new insights after watching their code replay. The replay feature was then removed for PS3, making the structures of PS1 and PS3 similar. We removed the replay to understand how changes in behavior and therefore programming self-regulation would persist after removing the code replay intervention. Participants could go back to review previously completed practice items, but could not attempt previously completed practice again.

3.5.1 *Practice items assessed basic Python constructs across three practice sets.*

We designed 21 introductory Python items to serve as practice across three practice sets, where each practice set was isomorphic in difficulty. The items covered concepts relating to basic Python constructs taught in class (variables, Boolean operators, math operators, conditionals, loops, lists, and function parameters). Each practice item required participants to read the prompt and fill in a function with valid Python code to fulfill all user tests. Figure 2a-c shows an example of a practice item.

We designed the items based on prior research findings with novice programmers (e.g. [65]), openly available online tools (e.g. CodingBat [42]), course materials shared from instructors from both courses, and course materials from the intermediate Python course that followed the courses that participants were enrolled in. In total, three authors designed 24 potential items.

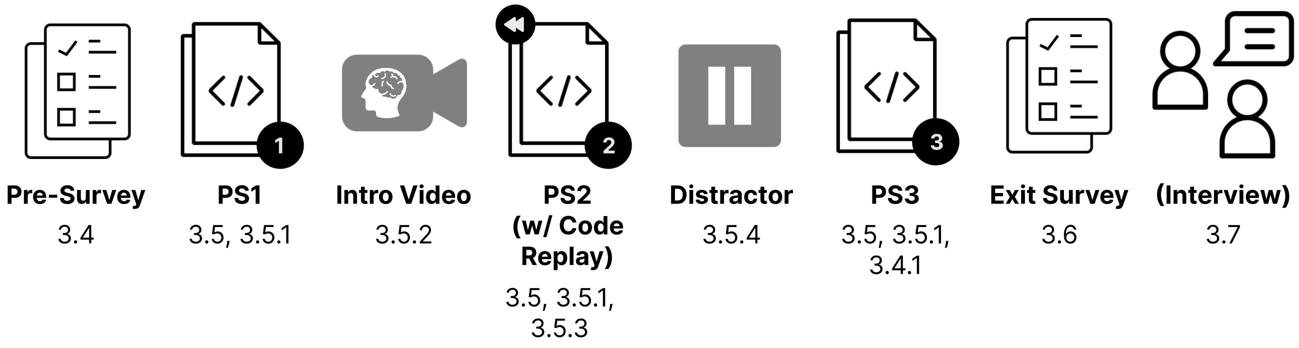


Figure 1: Overview of study flow with corresponding section numbers. Practice items were separated across three isomorphic practice sets (PS1-PS3). Only PS2 had code replays, with PS1 serving as a baseline and PS3 as a post-intervention comparison.

The 24 items were then reviewed by four computing/content experts and three psychometric experts, all of whom were not authors. Two authors then used this feedback to remove three items and revise the remaining 21 items for clarity and to adjust difficulty. They then organized items to distribute difficulty evenly across each problem set.

Each item also had accompanying unit tests, where correctness consisted of passing all unit tests in a single submission. In Code Replayer, participants could only continue on after they passed all unit tests or attempted five submissions.

3.5.2 Video to introduce metacognition. We created a four-minute video lesson on metacognition to serve as an introduction to the concept before participants began watching code replays in PS2. This explicit instruction provided novices with an abstract understanding of problem solving knowledge that can help support their development self-regulation skills when paired with reflective practice [33, 34]. Therefore, "priming" students about metacognition is crucial to support effectively practicing and developing self-regulation skills.

The video defined metacognition as "thinking about your own thinking," and explained its relation to programming, drawing from findings previously observed by Loksa et al. 2016 [32]. It also included examples of metacognitive programming strategies to aid in the user's understanding of how self-regulation is applied to writing code, such as interpreting a prompt or writing out a plan in pseudo-code and comments. The video ended by describing the importance of self-reflection and metacognition in an individual's ability to improve their programming ability.

3.5.3 Design of keystroke logs and code replays. To collect data to support code replays, Code Replayer passively collected keystroke logs as students answered practice items. Keystroke logs included events such as code edits (including deletions) and code submissions, with accompanying timestamps for each event. For PS2, we also tracked the amount of time participants watched replays. These keystroke logs followed a ProgSnap2 compliant format for future comparison with other process data [50]. We used these keystroke logs to create replays of participants' coding process.

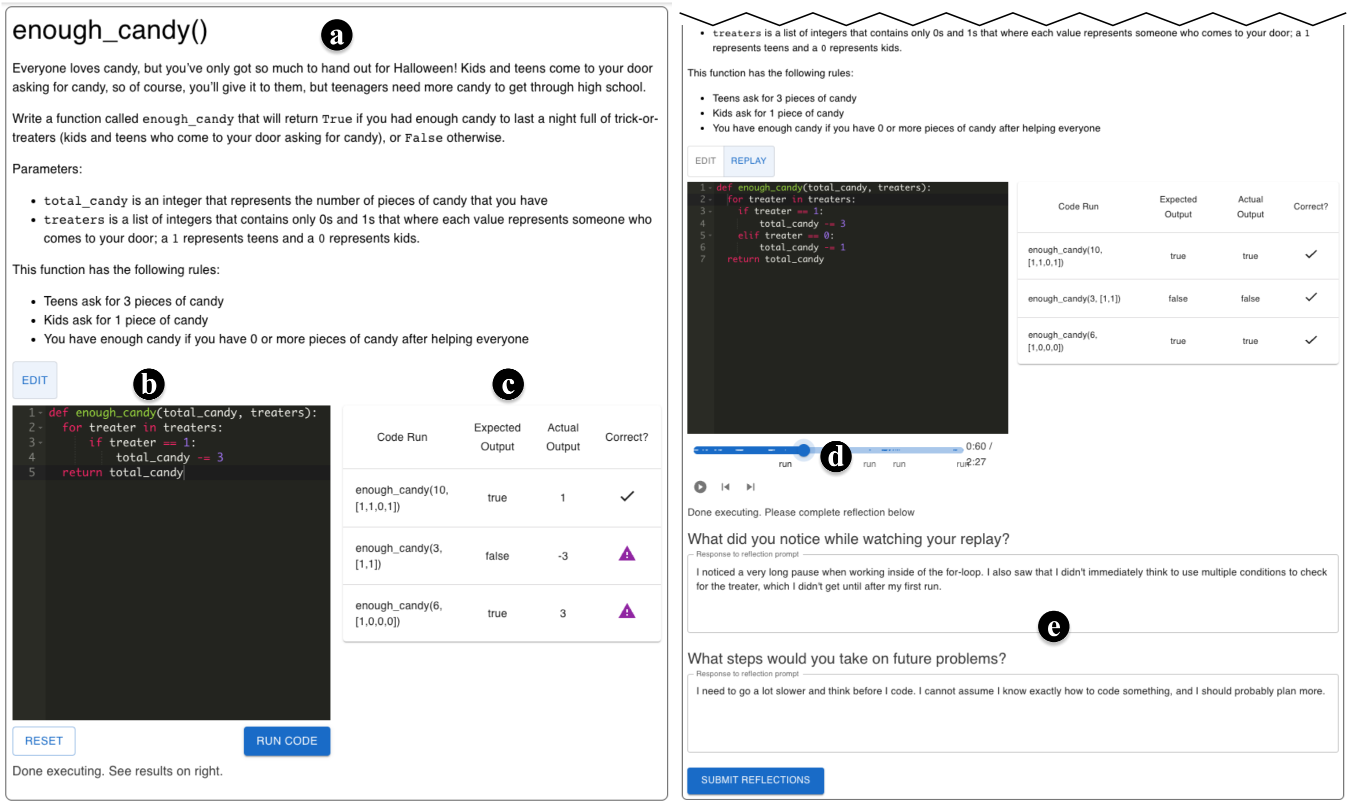
The intro video, shown in Figure 1 3.4.2, informed participants about the features of code replay and encouraged them to engage them as a way to engage in self-reflection on programming ability. Participants' replay of their coding process enabled them to view all code edits and answer submissions they made in real time. After finishing practicing an item in PS2, they could switch to replay mode and see their replay. The interface is shown in Figure 2d. Similar to a video player, participants could play or pause the replay or click on the progress bar or drag the slider to go to a specific position. They could also use next and previous buttons to jump to the next or previous event respectively. This enabled them to skip pauses where no events occurred.

While we encouraged students to engage in replays, watching the replays was not required of participants in PS2. The reflection questions appeared when the participant completed a problem, regardless if they watched their code replay or not. Furthermore, while the reflection questions in PS2 explicitly ask participants about their replay, it was not required of participants to engage in the replay before answering the reflection response.

3.5.4 Distractor task: unrelated online survey. After using code replays in PS2, we gave the students a brief distractor task before continuing to regular practice without code replays in PS3. The objective of the distractor task was to mitigate short-term, temporary learning gains related to viewing and reflecting on code replays. A distractor task occupies participants' working memory with content unrelated to self-regulation in programming so participants would rely more so on long-term memory when working on PS3 [29, 61]. The distractor was a survey that asked participants various non-computing questions such as how long each participant takes to shower, a musical instrument they would like to learn, and a logic question on tracking days in a seven-day week. Because these questions were not related to the research questions, none of the data gathered was analyzed.

3.6 Exit Survey: Demographics, MSLQ, Interview Recruitment

After completing the practice sets, participants completed an exit survey. The exit survey began by asking participants for consent to use their keystroke log data for our analysis. After this question, we asked if participants would be interested in a 30-minute recorded online interview intended to follow up on their metacognition while



enough_candy()

Everyone loves candy, but you've only got so much to hand out for Halloween! Kids and teens come to your door asking for candy, so of course, you'll give it to them, but teenagers need more candy to get through high school.

Write a function called `enough_candy` that will return `True` if you had enough candy to last a night full of trick-or-treaters (kids and teens who come to your door asking for candy), or `False` otherwise.

Parameters:

- `total_candy` is an integer that represents the number of pieces of candy that you have
- `treaters` is a list of integers that contains only 0s and 1s that where each value represents someone who comes to your door; a `1` represents teens and a `0` represents kids.

This function has the following rules:

- Teens ask for 3 pieces of candy
- Kids ask for 1 piece of candy
- You have enough candy if you have 0 or more pieces of candy after helping everyone

EDIT

```
def enough_candy(total_candy, treaters):
  for treater in treaters:
    if treater == 1:
      total_candy -= 3
  return total_candy
```

Code Run	Expected Output	Actual Output	Correct?
enough_candy(10, [1,1,0,1])	true	1	✓
enough_candy(3, [1,1])	false	-3	⚠
enough_candy(6, [1,0,0,0])	true	3	⚠

RESET RUN CODE

Done executing. See results on right.

- `treaters` is a list of integers that contains only 0s and 1s that where each value represents someone who comes to your door; a `1` represents teens and a `0` represents kids.

This function has the following rules:

- Teens ask for 3 pieces of candy
- Kids ask for 1 piece of candy
- You have enough candy if you have 0 or more pieces of candy after helping everyone

EDIT REPLAY

```
def enough_candy(total_candy, treaters):
  for treater in treaters:
    if treater == 1:
      total_candy -= 3
    elif treater == 0:
      total_candy -= 1
  return total_candy
```

Code Run	Expected Output	Actual Output	Correct?
enough_candy(10, [1,1,0,1])	true	true	✓
enough_candy(3, [1,1])	false	false	✓
enough_candy(6, [1,0,0,0])	true	true	✓

run run run run 0:60 / 2:27

Done executing. Please complete reflection below

What did you notice while watching your replay?

Response to reflection prompt

I noticed a very long pause when working inside of the for-loop. I also saw that I didn't immediately think to use multiple conditions to check for the treater, which I didn't get until after my first run.

What steps would you take on future problems?

Response to reflection prompt

I need to go a lot slower and think before I code. I cannot assume I know exactly how to code something, and I should probably plan more.

SUBMIT REFLECTIONS

Figure 2: Code Replayer interface. Participants read a formatted problem prompt (a), then write code in a code editor (b), run their code and see results of the unit test (c). After getting all unit tests correct or five submissions, participants can access the "replay" mode where they can view their replay (d). Annotations in the progress bar replay dot are events and submissions have "run" shown underneath them. Participants must then respond to reflection prompts (e) before continuing on.

completing the online portion of the study. After this, we asked participants to look at a sample coding problem. Participants were then asked to consider how they would approach the problem and describe their problem solving process. Participants self-reported their typical problem solving process before participating in the study and afterwards were asked to recall if there were any changes. Similar to the pre-survey, the participants were then again asked to measure their self-regulation using the nine-item metacognition sub-scale from the MSLQ instrument [45, 47].

After measuring their self-regulation, participants were asked on feedback for the online portion of the study, particularly focused on what they found helpful and what they did not find helpful about the code replay intervention. We asked participants if they used any outside resources to help them answer programming problems, and if so, what tool they used.

We collected demographic data at the end to avoid stereotype threat [62, 64]. While research on educational data mining often omits demographic data [41], trying to generalize results or interventions beyond groups for whom an intervention was designed for could mean that learners get interventions that are not suited for them [3]. Therefore, we collected and reported demographic data.

The demographics questions were optional and followed inclusive practices [39, 60, 63]. We asked questions about what degree participants were working towards, whether their parents/guardians completed a college/university degree, gender, ethnicity, age, languages spoken, familial language, and disabilities. Demographics of participants was previously reported in Section 3.2.

3.7 Follow-Up Interviews

We conducted semi-structured interviews with 12 self-selected participants within 10 days of them completing the online portions of the study. Interviews were conducted online and recorded video (including screen sharing) and audio with consent. Two authors and creators of Code Replayer conducted the interviews, which could contribute to response bias [8].

We asked participants how much time they spent on the practice sets, to share a general recollection of their experience, and whether they used any external tools to complete the work (e.g. Python Tutor). We asked each participant about their individual use of the code replay feature–particularly, how much or how little they engaged with it with the questions "How did watching your replays make you feel?", "What sort of thoughts came to mind?", and "Why (or why didn't) you engage with the replays?" (RQ1). We also asked participants to recall if they had previously ever reflected on their learning in a class they took, if Code Replayer had any particularly challenging practice items, and if Code Replayer reminded them of previously used tools.

Participants were then asked to watch one of their code replays with the interviewer; while sharing their screen, interviewees selected what they recalled was the most difficult practice item for them. We then asked them what they remembered about the problem and why they thought it was difficult. As they watched the code replay, we asked participants to think aloud and share what they noticed about how they solved the problem. Once the code replay had ended, we asked participants what they were thinking as they looked at their final solution to see how they reflected with code replays (RQ2).

To conclude the interview, participants were asked why they thought watching code replays could be helpful or unhelpful, as well as any other previous experiences that using the tool reminded them of.

4 DATA ANALYSIS OF SURVEYS, REFLECTION RESPONSES, INTERVIEWS

We analyzed data from reflection prompts, surveys, and interviews. Analysis of qualitative data requires judgment of data that we constructed in our study [17]. In this section, we describe the complexity of our qualitative coding process to allow readers to analyze this data on their own terms and employ similar analyses [7, 57].

While we intended to analyze log data (described in Section 3.5.3), we found noise in the data that came with the use of this tool in a discretionary setting. Analysis of log data often focuses on timestamps to calculate pause duration (e.g. [23, 24]), but this had broad variation. For example, the amount of time spent between loading a problem and beginning code edits (time "reading the prompt") ranged from 4.8 seconds to 23.1 hours. While some participants used this time primarily to read the prompt and plan their solution, others left the site open as they engaged in other tasks (e.g. work a shift on their job). We discuss future work related to log data analysis in Section 6.

4.1 Qualitative coding of reflection prompts to understand impact of replays on self-regulation

We qualitatively coded reflection prompts from practice before and after using the Code Replayer to understand how replay affected metacognition. We analyzed responses to the prompt "How did you approach this problem?" in PS1 and PS3 to code for evidence of programming behaviors as well as self-regulation behaviors. We analyzed 294 responses in total, coming from 21 participants responding to a combined total of 14 items from PS1 and PS3. In this section, we follow recommendations from Hoyt 2010 [19] on reporting our process for qualitative coding data using an established rating scale.

We applied an existing codeset for programming self-regulation behaviors [32, 33, 35], as shown in Table 1. The researcher with the most familiarity with programming self-regulation defined an initial code set based on prior work [33, 35]. They then had two other researchers and themselves code randomly selected responses. All three researchers then discussed discrepancies, came to consensus, and then iteratively refined the code set and code definitions for clarity. Because the orignal code set applied for journal reflection for coding projects, we needed to make adaptions to clarify the codebook for reflections on shorter code writing problems. Across five rounds, the three researchers came to consensus on 19% of the data (56 responses). The remaining responses were then equally split up such that two researchers coded each response. After each researcher independently coded the assigned responses, we created pairs among the 3 researchers to reach 100% agreement between 2-3 researchers for all responses. A common discrepancy was determining whether a reflection referred to implemented code (PIM in Table 1), a decided upon plan (SPL), or a potential solution (PSS).

4.2 Analyzing Change in MSLQ Rankings to identify disparate impacts of code replays

We used the metacognitive self-regulation subscale from the Motivated Strategies for Learning Questionnaire (MSLQ) to measure differences in metacognitive skills before and after the study [46]. Prior research developed validity evidence for use of this instrument [45] as well as for its use in computing education (e.g. [24]). The subscale consisted of nine Likert-type items where each item asked participants to rate how true statements were from "1: not at all true of me" to "7: very true of me" [46]. Three items were reversed (where a higher score suggested less self-regulation), so their scores were reflected prior to analysis. Following instructions on the use of the MSLQ [45], we took the mean of all items in the subscale to measure participants' self-regulation skills before and after the study.

An inconsistency in survey design resulted in the pre-survey and exit survey MSLQ having different ranges of potential scores. The pre-survey MSLQ ranged from 0-7 (instead of the intended 1-7 as the exit survey did). Following the principle of conservative interpretation of data, we chose not to compare MSLQ scores between pre-survey and exit surveys. We instead compared *rankings* of participants in pre-survey and exit survey. With this, we could identify how participants' metacognitive skills changed relative to other participants before and after the study.

We used rankings by average MSLQ score and changes in these rankings to identify participants whose self-regulation skills changed in disparate ways. This method is similar the *contrasting groups* method of psychometrics to make judgements on borderline scores [30, 31].

4.3 Thematic analysis of interviews to understand how code replay affected SRL

To better understand how students used Code Replay and what factors impacted its effects, we conducted a thematic analysis on the transcripts of the interviews we conducted with participants.

First, three researchers identified sensitizing concepts ([6, 43]) related to item design (e.g. difficulty), human factors (e.g. prior programming and self-regulation knowledge, familiarity with UX), and replay attributes (length of replay) affecting usage of code replays. Two researchers then reviewed interview notes, transcripts, and recordings and added notes to a virtual collaboration platform.

Three researchers then participated in collaborative affinity diagramming these notes to inductively generate themes.

5 RESULTS

We used data collected from our study to answer our three research questions related to how code replays affected self-regulation behavior, how participants used code replay differently, and what factors affected the use of code replays.

We refer to participants using an anonymous ID where the leading letter refers to the course they were enrolled in (A or B) followed by a sequential number. IDs ranged from A-01 to A-19 and B-01 to B-02. We refer to participants using they/them pronouns for anonymity.

5.1 Most participants watched most replays

The goal of our study was to understand how code replays could supplement novice programmers' typical practice to develop their self-regulation behaviors.

While we did not intend to manipulate the amount of code replays participants watched, variations in engagement with code replays could confound study findings. Figure 3 shows the distribution of participants by number of code replays started, as measured by analysis of log data. We found that about half of participants (10/21) started watching replays for all 7 items in PS2. Four in five (17/21) watched most of their replays (4 or more).

5.2 RQ1: Effect of Code Replays on Self-Regulation

To understand the effect of code replays on metacognition, we qualitatively coded the participants' responses to reflection prompts before and after they watched their replays. Table 1 shows our analysis of 294 reflection prompts for 21 participants completing 7 items in PS1 before watching their code replays and then 7 more items in PS3 after watching their code replays.

We found differences in the frequency of reported self-regulation behaviors before (PS1) and after (PS3) watching code replays. There was a decrease in the number of reflection prompts mentioning implementation of solutions (PIM, -17%) which was statistically significant ($\chi^2 = 10.135$, $p < 0.05$, Pearson's χ^2 test with Yates' continuity correction and Bonferroni correction) with a small effect size ($\phi = 0.19$). We also found that reflections after watching code replays more frequently mentioned interpreting the prompt (PIN, +6%) and planning (SPL, +5%). While these were not statistically significant, analysis of interview data showed how replays could have supported these behaviors.

5.2.1 Replays improved self-regulation through reflection and self-explanation. Analysis of interviews and think-aloud data identified that a majority of the 12 interviewed participants felt that code replays helped them see and reflect on their process and then identify potential improvements. A-10 and A-16 felt that replays enabled them to get a "third person" perspective of themselves coding. A-10 felt that replays helped them demonstrate their understanding to themselves by positioning them as a teacher self-explaining their previous behaviors:

> A-10: *"I was like standing behind myself doing this code... Whenever I'm talking about my ideas of what I'm learning, I feel like that helps me. Because I am kind of teaching it to other people what I'm doing. And I feel like if you're teaching or explaining what you're doing to other people, to myself, then I guess it shows I'm understanding..."*

Using replay as a reflection opportunity enabled participants to recognize how much time and effort they dedicated to interpreting the prompt. A-16 used the initial pause in the replay to consider how long they read the prompt (PIN) and planned their solution (SPL).

> A-16: *"I can see how long the time when I was not writing anything until I started to write. There's a time, you know, there, so I guess that's the time I spend on like thinking about the prompt and how to code."*

Another participant used replays to notice that they did not dedicate enough time to interpret a problem prompt (PIN) prior to starting to code:

> A-12: *"When I watch back what I was doing at the time, I can see what myself thinking and see myself arguing with myself. Should I do that? Or should I do this?... Sometimes I noticed I haven't read the question actually, and I just go right into coding. I go back and then realize what I did was not what the question was asking."*

While we designed code replays to improve self-regulation skills, two participants felt replays also helped improve their programming skills by finding new applications of learned concepts and identifying gaps in their understanding. One participant used replays to identify opportunities to apply `elif` (else-if), a concept they learned in lecture (A-07, described further in Section 5.3.3). Another participant used replays to identify gaps in their programming knowledge. They used patterns in code edits to identify programming concepts they were less familiar with:

> A-01: *"Replays show me how I arranged my code and which parts I changed... which shows areas where I'm not familiar with these kind of code."*

Another participant found different benefits to watching a replay based on whether they got a problem correct or not:

> A-01: *"If the code is correct, I will watch the replays to see what I change[d] during the programming, and I can find which part of that I'm weak. If the code's wrong, I mainly use the you replay to reorganize my logic and thinking."*

But one highly motivated participant (A-08) did not find a benefit in watching code replays. They had middle 1/3 MSLQ rank that remained relatively unchanged (slight increase). They were a graduate student taking their first programming course for help with data analysis. Despite their motivation to learn from replays, they did not feel they got value from them:

> A-08: *"I think I was kind of like overthinking it... I was thinking like 'Oh, they probably gave us this tool, and... metacognition is like going to come into play at some point. So like maybe I'm supposed to see something*

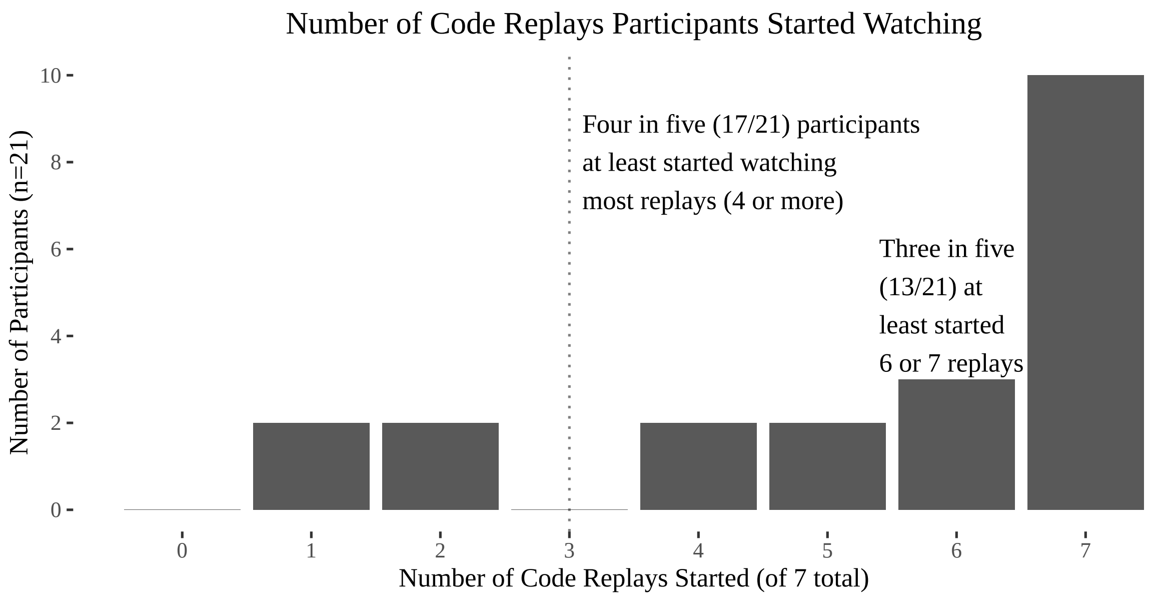


Figure 3: Histogram of code replays participants started.

that's going to be like that. I didn't know before, or like something was going to like expand in my mind from watching myself over again.' And I just like wasn't getting that."

5.2.2 *Perceptions of pauses/gaps varied.* Pauses or gaps in the code replays occurred whenever participants were not actively editing their code. They often appeared in the beginning of replays when participants were reading the prompt, within code edits when participants stopped to think or take a break, or trying to execute code when participants reviewed feedback.

Participants often used pauses/gaps in code replays to recall behaviors. One participant felt that short pauses/gaps in the replay before starting to write code suggested they did not spend enough time reading the prompt, so they tried to change that behavior:

A-16: *"I thought [code replays were] helpful because I can get a clue to why the question was wrong and I can also notice the gaps between thinking and typing. If the gap was short then, I was reading the prompt not so carefully, so I spent more time reading the prompt."*

Another participant focused on pauses in their replay that occurred after their code failed to execute. They interpreted those pauses as evaluating code relative to test-cases (PEI) or interpreting the prompt (PIN):

A-18: *"The main thing I noticed while I was watching them was like the pauses [...] where I had to re-look at the test cases, or reread the question again, because there was a bug in my code."*

But long pauses could be cumbersome because there were no actual changes to watch. As an extreme example, A-18 speculated that for larger project-length assignments, replays would not be helpful because it would be inefficient to replay everything:

A-18: *"[replays] were helpful for shorter problems, but I was thinking of how we would use it on a bigger assignment. I don't think it'd be super helpful since I'm not actively coding during 100% of my time. I am mostly looking at my work. I feel like for smaller coding chunks, it is more effective since it is a shorter time. If it's a really long code, I don't know if I want to watch back everything."*

5.2.3 *Participants perspectives on replays of struggle were polarized.* Participants tended to struggle more on more difficult problems. For easier problems, A-07 felt replays would not be helpful because they would be too quick. However, two participants (A-03, A-08) reported appreciating the positive reinforcement of seeing replays of getting a problem correct:

A-03: *"If I got the problem right on the first try, it made me feel good to watch it work."*

For more difficult problems where participants struggled more, four interviewed participants found that watching replays of errors ranged from not helpful to detrimental experiences.

Two participants found watching errors they just made "not helpful" (A-05, A-12). A-12 started ranked in the bottom 1/3 by MSLQ score and increased to the middle 1/3. They felt that watching the same error again could reinforce poor programming habits:

A-12: *"I guess sometimes you might make the same mistakes watching [the replay].... When I normally code, if I got really stuck, I tend to open a blank page so that I don't have any influence from my first [try]... Sometimes if you watch it again, you may tend to do the code the same way. If the code works, but it's not a good way to code it, you might end up doing the same thing."*

Table 1: Descriptions and examples of programming and self-regulation behaviors we qualitatively coded for in 294 reflection prompts, split evenly between PS1 and PS3. Percentages reflect the proportion of reflection prompts (out of 147) that were coded for a given behavior for the problem set before (PS1) and after (PS3) participants watched their code replays.

Behaviors	Description	Example Response	PS1	PS3	Δ
PIN: Interpret Prompt	Interpreting the prompt, reconsidering actions in reference to the prompt, or decomposing the problem.	*I read the prompt and knew what it was asking of me, to subtract expenses from revenue...* A-09	52%	59%	+6%
PSA: Search for analogous problems	Demonstrating intent to reuse knowledge or code from related problems.	*I found that this problem is kind of similar to the previous reverse number question, so I go back and check some of my code there...* A-16	2.0%	3.4%	+1%
PSS: Search for solutions	Adapting solutions to related problems or by finding solutions in textbooks, online, or from classmates or teachers.	*I thought about how [my instructor] said in class that you can multiply integer n with strings...* A-18	10%	7%	-3%
PEP: Evaluation a potential solution	Demonstrating testing or evaluating outcomes intent to test a potential solution.	*I knew this should be a for loop that iterates between 1 and n.* A-11	4%	2%	-2%
PIM: Implement a solution	Translating a solution into code.	*I use str() to convert int to string, and then compare the result to target.* A-07	82%	65%	-17%
PEI: Evaluate implemented solution	Evaluating correctness and quality of an implementation.	*Once I tested my first run and realized I didn't get what I wanted, I looked at the actual output and decided where in my code I would need to make small adjustments...* A-08	20%	16%	-4%
SPL: Planning	Intended work goals or intended order of work.	*First I need to tell what's the original price of this car, then I tell its current price depending on my credit score. Finally I tell if I could buy this car...* A-15	33%	38%	+5%
SPM: Process Monitoring	Work being started, work currently in progress, when a task is complete, or that identify actions as part of their process.	*I first split the value of n into 3 numbers, and then calculated whether the sum of the 3 numbers is equal to the value of target..* B-01	46%	47%	+1%
SCM: Comprehension Monitoring	Identifying known or unknown programming concepts or understanding of the problem prompt. Reflection about the understanding of code or problem prompts.	*I know that you can multiply a string by an integer and it will give me X number times of the string...* A-12	18%	22%	+4%
SRE: Reflection on Cognition	Reflecting on prior thoughts, behaviors, or feelings.	*With incredible frustration... I was pretty nervous about running the code* A-05	14.3%	13.6%	-1%
SSE: Self-explanation/ rationale	Code explanation for increased understanding or to provide rationale to decisions or behaviors.	*I feel like a for loop is needed to get the different numbers to multiply and then also keep track because with factorials you build on the previous multiplication problem.* - A-10	44%	46%	+2%
(none)	no codes apply to this response	*Sorry, I chose to skip the question.* B-01	2.7%	7.4%	+5%

B-02 also felt that replays reinforced sub-optimal skills and strategies, which they referred to as "tunnel vision." We elaborate on this further in Section 5.3.2.

More alarmingly, two participants thought replays made them feel negatively. A-16, who ranked in the top 1/3 in MSLQ in the pre-survey but then decreased to middle 1/3, felt that the replay made them feel "stupid" (A-16):

> A-16: *"Sometimes I would think of myself as kind of stupid because I spend a lot of time [coding]."*

A-03 felt frustrated seeing a replay of problems they were unable to solve:

> A-03: *"If I got the problem right on the first try, it felt good. If there were a lot of mistakes, especially if I didn't pass the problem, it was frustrating to watch back because you know what's wrong but you can't fix it."*

But not all participants found struggle to be detrimental. A participant with a high ranking MSLQ score felt that difficult problems were interesting because they showed how their strategies changed:

> A-05: *"On the ones where they got harder, the most interesting for me was watching where I changed course, or where I made mistakes like 'oh wait, that's a set not a list' or something. But more often, just sort of like 'oh I started to go at it this way and then in the process of doing that, I realized a more way elegant way of doing*

it.' Or I realized what a problem was that I was about to hit and then I was like, 'back up and solve it.''"

5.3 RQ2: How novices used replays differently

To understand how participants used replays differently, we conducted an in-depth analysis of three participants. We selected these participants because of extreme changes in their MSLQ rankings between the pre-survey and exit survey as well as demographic diversity.

Figure 4 shows changes in rankings in average MSLQ score between pre-surveys and exit surveys. The three participants we interviewed are emphasized with purple and thicker lines. They are A-05, who started with an MSLQ rank in the top 1/3 and finished with the highest ranked MSLQ score; A-07, whose MSLQ score started in the top 1/3 but dropped to the bottom 1/3, and B-02, whose MSLQ score started in the bottom 1/3 but finished in the top 1/3.

The correlation between the rank of number of replays watched (Figure 3) and change in MSLQ was very weak (Spearman rank correlation coefficient $r_s = 0.05$) and there was no significant difference (Wilcoxon Signed Rank Sum Test, $p = 0.97$). This suggests that there is a very weak relationship between number of replays watched and change in MSLQ rank, likely because of minimal variation in number of replays watched. Furthermore, all three emphasized participants described in this section started watching six (A-07) or all seven (A-05, B-02) of their code replays.

5.3.1 A-05: Replays encourage making process explicit, but scaffolding required.

A-05 was an older (over 40 years old), gender queer, White, multilingual, learner who was not working towards a degree. They were enrolled in their first Python course, but had previous experience with other programming languages. They had teaching experience, and used this prior experience to frame code replays as a tool that can enable "thinking about thinking" but also requires sufficient feedback and guidance to support the development of self-regulation skills.

Initially, A-05 had a very internal process where they worked out problems in their head. But after watching some uninteresting replays, they started to consider making their process external:

> A-05: *"I watched the first [replay] and realized that, like oh, most of my thought process is me, just sitting there clearly thinking for five minutes, and then writing a ton of code or something. In order to make [replays] more useful, I changed my process knowing that I could go back and watch myself and extrapolate what I was thinking."*

They also found it interesting how replays enabled them to see how their strategies changed throughout the problem:

> A-05: *"the most interesting for me was watching where I changed course, or where I made mistakes. Like 'oh wait, that's a set not a list' or something. But more often, just sort of like 'oh I started to go at it this way and then in the process of doing that, I realized a more way elegant way of doing it.' Or I realized a problem that I was about to hit and then I like, back up and solve it."*

Connecting Code Replayer to their teaching experience, they saw replays as something that could support interactions between teachers and learners:

> A-05: *"As a teacher brain, I can be like 'sure that'd be an amazing tool' because I could sit there with a student, and I could be like 'see, look what you were doing here' or I could look at those long pauses and be like 'hey, instead of having these sort of long pauses, and then writing out ten lines of code, why don't you just like, do that on the page?''"*

5.3.2 B-02: Replays aligned plans with process, but led to fixation on existing process.

B-02 was a young (18-25 yrs old) Hispanic/Latino man who was pursuing a Bachelor's degree while also working full-time. He started with an MSLQ rank in the bottom 1/3 (rank 16.5/21) and then finished with in the top 1/3 (rank 3.5). Because he worked full time, he used Code Replayer intermittently, using replays to review his thought process from hours prior:

> B-02: *"... I was like doing it between work. So I had, like a twelve hour jump from the start to finish. It was really cool for me to see like twelve hours ago, what was my head thinking?"*

B-02 used code replays to bridge a gap between his intended plan for solving a problem and how he actually solved problems. This was a recurring theme that he came back to four times in the hour-long interview. His ideal process involved considering the output and then working backwards from the desired output. His initial process however involved "just writing code immediately." He felt that reflecting on code replays enabled him to align his intended plans with his process by the end of PS3:

> B-02: *"I definitely I went into like wanting to see the replays, so that I could see that difference in what I wanted to do versus what I did… that definitely helped me later on because eventually... by the last couple of problems, I was actually doing most of what I wanted to do like the thought process that I wanted to use."*

B-02 felt that replays led to a "tunnel vision" in which reflecting on the wrong approach could lead to fixating on that approach:

> B-02: *"I feel like watching replays [in] the second problem set of me using a list might have … influenced my thinking to really like reflect hard on lists, right? Which made me feel like I could use a list to solve anything, which obviously wasn't the case. So I feel like, even if I look at any problem today, I might just try to make a list right away. So maybe it's kind of, like, giving you that tunnel vision."*

To overcome this fixation on sub-optimal strategies, B-02 wished he could see replays of other types of processes. He likened this to watching replays of e-sports and online gaming:

> B-02: *"You'd review your match replays, and you look through them and you see 'What was I doing?' and 'What were other people doing?''"*

5.3.3 A-07: Used Additional Tools in Addition to Code Replayer.

A-07 was pursuing a Bachelor's degree and came from a household that spoke a different language than the language of instruction.

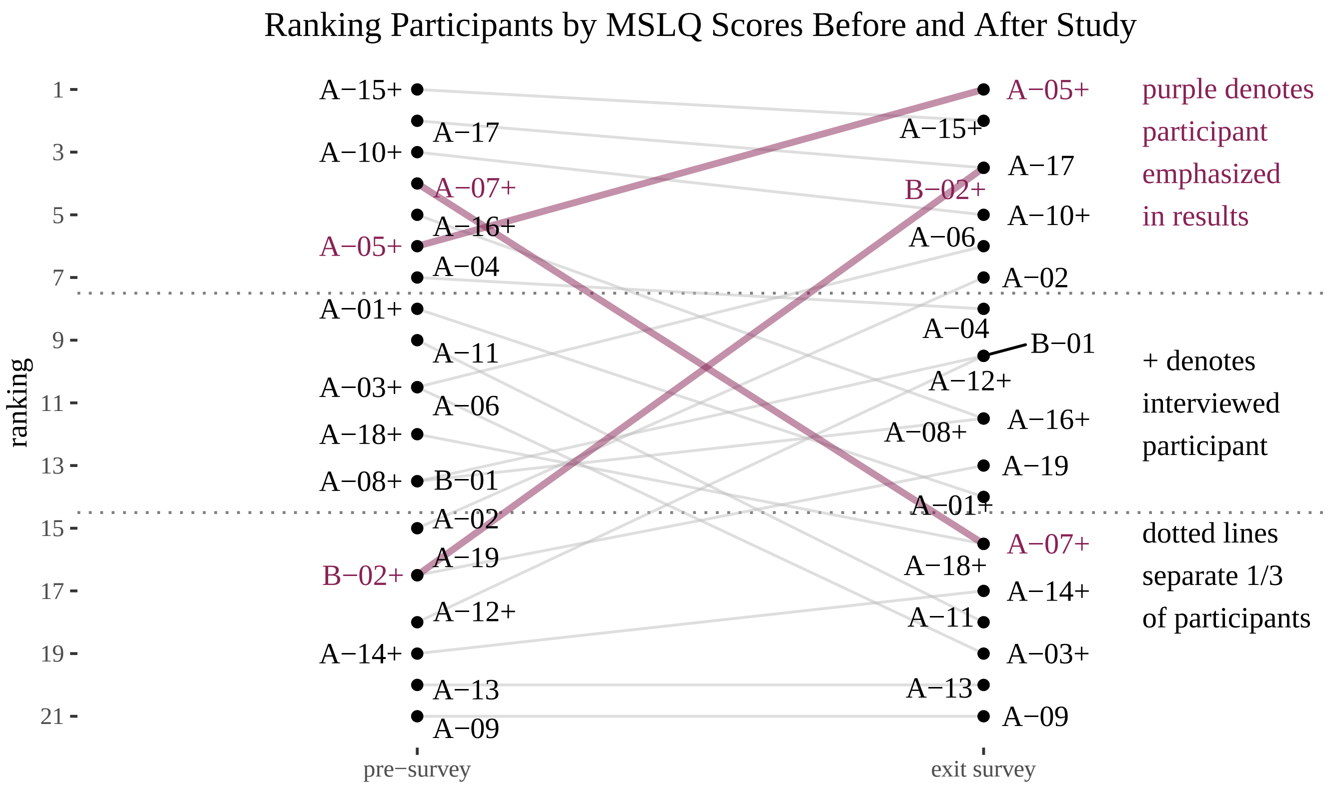


Figure 4: Rankings of participants' average MSLQ scores from pre-surveys and exit surveys. We emphasized participants with purple text and thicker purple lines in our results because of the extreme changes in their MSLQ rankings.

They had the greatest decrease in MSLQ ranking, ranking in the top 1/3 in the pre-survey and then ending up at the bottom 1/3 in the exit survey. In addition to Code Replayer, A-07 used the external resources of Python Tutor [15] to test code and a computational notebook to take notes.

A-07 used Python Tutor to identify errors in their code:

> A-07: *"Firstly, I write the code on Code Replayer platform and then I click run to see if all the results show correct. But [for] almost all of them, the first time wasn't correct. So I just review them and after a few minutes, if I can't find the error, I go to Python Tutor to check it. But, if I can find out, I directly add it in the platform and then, when I'm done, I use Code Replayer to review my process."*

Python Tutor [15] had better support for finding errors, such as visualization of stepping through code. This suggested that by relying on it to debug, A-07 may engaged less with the debugging process, a crucial phase in self-regulation (PEI). It also meant that code replays no longer showed the incremental steps of code editing. Instead, A-07's code replays had long pauses (presumably while they worked in Python Tutor) and then large edits when they copied code back into Code Replayer. Nevertheless, they still felt that code replays helped them avoid repeating similar mistakes:

> A-07: *"[code replays] can remind me of something so in the future assignments I won't make similar mistakes [from before]"*

While we theorized that code replay could improve self-regulation skills, we did not expect them to support the development of programming skills. But by watching replays, A-07 recognized opportunities to apply a construct they learned in class, ELIF:

> A-07: *"When I was first coding everything I like to use 'if' and 'else' in all these things. But after [watching replays,] I noticed that I can actually use 'if' and 'elif' to limit the [conditions to] be more precise and useful."*

6 DISCUSSION: DESIGNING CODE REPLAYS FOR INTERPRETATION OF PAUSES

Our analysis identified how reflecting on code replays could support the development of novice programmers' self-regulation skills. Qualitative coding of reflection prompts identified that participants less frequently reported implementing a solution (PIM). Interviews identified that participants considered locations and durations of pauses in code edits as potentially relevant to planning despite replays showing no change during these pauses. Participants used code replays in different ways, such as using them to make their process more explicit (A-05) and to follow-through with problem solving plans (B-02). Using Code Replayer in tandem with other tools may not have benefited A-07's self-regulation skills, as their code replays no longer captured their entire code editing process. However, multiple participants found replays of struggles detrimental. Collectively, this paper contributes a formative, empirical evaluation on the feasibility of using code replays as a scalable intervention to develop novice programmers' self-regulation skills. In the remainder of this section, we consider ways to interpret these findings.

6.1 Limitations

One interpretation of our findings is that code replays did not impact self-regulation skills because that we found no statistically significant evidence of more frequent self-regulation behaviors before and after the study. However, this study was an initial design exploration and feasibility study in a context that was externally valid to self-directed practice. Analysis of rich qualitative data from reflection prompts reported immediately after completion of items as well as interviews with think-aloud provided us with insights to understand *how* participants used code replays in different ways towards disparate outcomes. Furthermore, we intended for this study context to be comparable to a formative practice environment. As such, we did not try to conduct this study in a controlled, lab-like setting. We made no attempts to control the amount of time spent using the tool. Participants were also learning new content as the study progressed. This dynamic environment made log data too confounded to provide a strong signal. Future work can explore the efficacy and effect size of code replay interventions on programming self-regulation skills in more controlled contexts, such as in time-constrained settings more consistent with classroom experiences. Future work could also investigate the development of programming self-regulation skills across a longer duration, as prior work in foreign language learning found that seven weeks of instruction with video replays improved undergraduates' metacognition [1].

Another interpretation of our findings is that variations in engagement with code replays confounded results. We found that 4 in 5 participants watched most of their code replays, there was a lack of correlation between number of replays watched and change in MSLQ rank, and all our emphasized participants watched all or almost all of their code replays. Therefore, we do not find evidence that variations in the number of replays had a detectable effect in our study. However, students engaged with replays different (as described in Section 5.3), so future work can explore how different "dosages" of code replays affect novice programmers' self-regulation skills.

6.2 Implications for research and practice

Another interpretation of our findings is that code replays can support the development of self-regulation skills in novice programmers. Zimmerman's framing of self-regulation includes learners reflecting on their own recordings [34, 71]. We found that participants were able to use replays to reflect on their process and identify improvements to their problem solving strategies. In particular, participants focused on spending more time interpreting the prompt and planning, two important self-regulation behaviors that novice programmers often overlook [32, 35].

However, replays may be less helpful for advanced learners. Multiple participants suggested that replays were unhelpful because they had sufficient mastery of some problems. This suggests that some participants may have had advanced enough programming or self-regulation skills such that watching code replays was an unnecessary burden. This aligns with Expertise Reversal Effect, which states that the effectiveness of scaffolding techniques depends on the levels of learner expertise [21]. This also aligns the theory of self-regulation we applied in this study, which states that learners begin to proceduralize self-regulation skills as they develop [71]. Future work can explore how to use code replays as more targeted interventions, perhaps focusing on reviewing replays of specific exercises or even "inflection points" within exercises (e.g. when a learner changed their problem solving strategy). This aligns with prior work on process recordings from beyond computing education which suggests a need for being selective about the collection and use of recordings, as previously described in Section 2.2.

A final interpretation is that we must design interactions and scaffolding around code replays to ensure they equitably develop self-regulation skills for novice programmers. One opportunity involves designing for gaps/pauses in the code replay. Our study identified that during pauses in code edits (e.g. when a participant is thinking), participants attempted to recall their thoughts or behaviors. Prior work found relationships between duration of pauses in code writing tasks and exam scores [24]. Future work could design techniques to provide more affordances towards thoughts and behaviors during pauses in code edits. Designing techniques to support richer code replays could involve unintrusively collecting additional data. For example, one participant wrote comments in their code to fill the pauses. Other designs could encourage and record think-aloud or prompts that complement rather than distract from learning experiences.

A critical tension that this study identified was balancing the benefit and the burden of having to relive past mistakes. Multiple participants identified how code replays were most beneficial for difficult problems, where they could reflect upon and improve their programming processes. However, multiple participants also indicated that reliving errors was not helpful, frustrating, or hindered their self-efficacy. Students found replays of challenging problems most helpful, but these replays may also be the most difficult to review because they can reflect struggles that students faced. Future research and use of code replays must ensure that replays do not erode learners' self-efficacy, perhaps through scaffolding, reviewing with an instructor or peer, and/or more targeted use of replays. For code replays to more equitably support learners, we must consider how learning from feedback is not only mediated by technology, but also by social and cultural norms.

Given these interpretations of our findings, there are many potential ways for instructors to consider incorporating code replays into introductory programming courses. One approach is using code replays in existing introductory courses. While prior work has explored having proficient programmers demonstrate their processes through live-coding or live-streaming [16, 44, 58], participants have suggested future designs of code replays that include them as an optional feature in an IDE, aid teachers in understanding students' processes. This could help scaffold students' transition from observing another programmers' process to regulating their own. Another approach is incorporating code replays into explicit curriculum that teaches metacognition (e.g. [33, 48, 49]). A crucial design question for this work will be how learners, their peers, and instructors can effectively interpret and use code replays. Alternative representations of replays, such as with charts [59] or heatmaps [12], could support shared interpretations. Shared interpretations could arise when students use code replays, perhaps with synchronized audio recordings, to reflect on pair programming activities [26, 38, 51, 53].

In conclusion, we return to the challenging yet common context of individuals learning programming online with limited access to instructors with computing and self-regulation expertise. This study contributed a formative evaluation of how reflecting on replays of code writing processes could develop the self-regulation skills of novice programmers practicing on their own. Our findings suggest that code replays can serve as a recording for novice programmers to reflect upon and develop self-regulation skills. However, tools require social support and cultural considerations for them to be equitably effective. Therefore, we must consider the interaction of factors including interface design, learners' prior experiences, and broader social, contextual, and cultural factors when designing pedagogy that uses code replays. By doing so, code replays may be able to equitably support the development of often neglected yet crucial self-regulations skills in tandem with programming skills.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1539179, 1703304, 1836813, 2031265, 2100296, 2122950, 2137834, 2137312, 12566082, funding from the McCoy Family Center for Ethics in Society, and unrestricted gifts from Microsoft, Adobe, and Google. Supplementary material available at https://github.com/codeandcognition/archive-2023icer-xie

REFERENCES

[1] Serhat Altıok, Zeynep Başer, and Erman Yükseltürk. 2019. Enhancing metacognitive awareness of undergraduates through using an e-educational video environment. *Comput. Educ.* 139 (Oct. 2019), 129–145. https://doi.org/10.1016/j.compedu.2019.05.010

[2] Mark Aveline. 1992. The use of audio and videotape recordings of therapy sessions in the supervision and practice of dynamic psychotherapy. *British journal of psychotherapy* 8, 4 (June 1992), 347–358. https://doi.org/10.1111/j.1752-0118.1992.tb01198.x

[3] Ryan Baker and Aaron Hawn. 2021. Algorithmic Bias In Education. (2021). https://doi.org/10.35542/osf.io/pbmvz

[4] Alan F Blackwell. 2002. What is Programming?. In *Psychology of Programming Interest Group (PPIG)*.

[5] Jonas Boustedt, Anna Eckerdal, Robert McCartney, Kate Sanders, Lynda Thomas, and Carol Zander. 2011. Students' perceptions of the differences between formal and informal learning. In *Proceedings of the seventh international workshop on Computing education research*. ACM. https://doi.org/10.1145/2016911.2016926

[6] Glenn A Bowen. 2006. Grounded Theory and Sensitizing Concepts. *International Journal of Qualitative Methods* 5, 3 (Sept. 2006), 12–23. https://doi.org/10.1177/160940690600500304

[7] Michelene T H Chi. 1997. Quantifying Qualitative Analyses of Verbal Data: A Practical Guide. *Journal of the Learning Sciences* 6, 3 (July 1997), 271–315. https://doi.org/10.1207/s15327809jls0603_1

[8] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. Yours is better!. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin Texas USA). ACM, New York, NY, USA. https://doi.org/10.1145/2207676.2208589

[9] Joseph Ditton, Hillary Swanson, and John Edwards. 2021. External Imagery in Computer 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, 1226–1231. https://doi.org/10.1145/3408877.3432426

[10] Karl Anders Ericsson and Herbert Alexander Simon. 1993. *Protocol Analysis: Verbal Reports as Data Revised Edition.* The MIT Press.

[11] Anneli Eteläpelto. 1993. Metacognition and the Expertise of Computer Program Comprehension. *Scandinavian Journal of Educational Research* 37, 3 (Jan. 1993), 243–254. https://doi.org/10.1080/0031383930370305

[12] Gordon Fjeldsted and John Edwards. 2022. Quantifying Student Struggles using Heatmaps and Keystroke Data. In *2022 Intermountain Engineering, Technology and Computing (IETC)*. 1–5. https://doi.org/10.1109/IETC54973.2022.9796894

[13] Enrico Gandolfi, Richard E Ferdig, and Robert Clements. 2022. Streaming code across audiences and performers: An analysis of computer science communities of inquiry on Twitch.tv. *British journal of educational technology: journal of the Council for Educational Technology* (Feb. 2022). https://doi.org/10.1111/bjet.13207

[14] Nichole M Garcia, Nancy López, and Verónica N Vélez. 2018. QuantCrit: rectifying quantitative methods through critical race theory. *Race Ethnicity and Education* 21, 2 (March 2018), 149–157. https://doi.org/10.1080/13613324.2017.1377675

[15] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. 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, 579–584. https://doi.org/10.1145/2445196.2445368

[16] Lassi Haaranen. 2017. Programming as a Performance: Live-streaming and Its Implications for Computer Science Education. 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, 353–358. https://doi.org/10.1145/3059009.3059035

[17] David Hammer and Leema K Berland. 2014. Confusing Claims for Data: A Critique of Common Practices for Presenting Qualitative Research on Learning. *Journal of the Learning Sciences* 23, 1 (Jan. 2014), 37–46. https://doi.org/10.1080/10508406.2013.802652

[18] Joseph Henrich, Steven J Heine, and Ara Norenzayan. 2010. Most people are not WEIRD. *Nature* 466, 7302 (July 2010), 29. https://doi.org/10.1038/466029a

[19] William T Hoyt. 2010. Interrater reliability and agreement. In *The Reviewer's Guide to Quantitative Methods in the Social Sciences*, Gregory R Hancock, Ralph O Mueller, and Laura M Stapleton (Eds.). Routledge, 141–154. https://doi.org/10.4324/9781315755649-10/interrater-reliability-agreement-william-hoyt

[20] C M Janelle, D A Barba, S G Frehlich, L K Tennant, and J H Cauraugh. 1997. Maximizing performance feedback effectiveness through videotape replay and a self-controlled learning environment. *Research quarterly for exercise and sport* 68, 4 (Dec. 1997), 269–279. https://doi.org/10.1080/02701367.1997.10608008

[21] Slava Kalyuga. 2009. The Expertise Reversal Effect. In *Managing Cognitive Load in Adaptive Multimedia Learning*. IGI Global, 58–80. https://doi.org/10.4018/978-1-60566-048-6.ch003

[22] Harrison Kwik, Benjamin Xie, and Amy J Ko. 2018. Experiences of Computer Science Transfer Students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. ACM Press, 115–123. https://doi.org/10.1145/3230977.3231004

[23] Juho Leinonen, Leo Leppänen, Petri Ihantola, and Arto Hellas. 2017. Comparison of Time Metrics in Programming. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 200–208. https://doi.org/10.1145/3105726.3106181

[24] Leo Leppänen, Juho Leinonen, and Arto Hellas. 2016. Pauses and spacing in learning to program. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '16)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/2999541.2999549

[25] Colleen M Lewis. 2012. The importance of students' attention to program state: a case study of debugging behavior. 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, 127–134. https://doi.org/10.1145/2361276.2361301

[26] Colleen M Lewis and Niral Shah. 2015. How Equity and Inequity Can Emerge in Pair Programming. In *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*. ACM Press, Omaha, Nebraska, USA, 41–50. https://doi.org/10.1145/2787622.2787716

[27] Sebastian Linxen, Christian Sturm, Florian Brühlmann, Vincent Cassau, Klaus Opwis, and Katharina Reinecke. 2021. How WEIRD is CHI?. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21, Article 143)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3411764.3445488

[28] Rebecca Lippmann Kung and Cedric Linder. 2007. Metacognitive activity in the physics student laboratory: is increased metacognition necessarily better? *Metacognition and Learning* 2, 1 (April 2007), 41–56. https://doi.org/10.1007/s11409-007-9006-9

[29] Ye Liu and Xiaolan Fu. 2007. How Does Distraction Task Influence the Interaction of Working Memory and Long-Term Memory? In *Engineering Psychology and*

Cognitive Ergonomics, Don Harris (Ed.). Vol. 4562. Springer Berlin Heidelberg, Berlin, Heidelberg, 366–374.

[30] Samuel A Livingston and Michael J Zieky. 1982. *Passing Scores: A Manual for Setting Standards of Performance on Educational and Occupational Tests.* Educational Testing Service.

[31] Samuel A Livingston and Michael J Zieky. 1989. A Comparative Study of Standard-Setting Methods. *Applied Measurement in Education* 2, 2 (April 1989), 121–141. https://doi.org/10.1207/s15324818ame0202_3

[32] Dastyni Loksa and Amy J Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16).* ACM, New York, NY, USA, 83–91. https://doi.org/10.1145/2960310.2960334

[33] Dastyni Loksa, Amy J Ko, Will Jernigan, Alannah Oleson, Christopher J Mendez, and Margaret M Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. ACM Press, 1449–1461. https://doi.org/10.1145/2858036.2858252

[34] Dastyni Loksa, Lauren Margulieux, Brett A Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2021. Metacognition and Self-Regulation in Programming Education: Theories and Exemplars of Use. *ACM Trans. Comput. Educ.* (Dec. 2021). https://doi.org/10.1145/3487050

[35] Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J Ko. 2020. Investigating Novices' In Situ Reflections on Their Programming Process. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE), Research Track.* ACM.

[36] Murali Mani and Quamrul Mazumder. 2013. Incorporating metacognition into learning. 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, 53–58. https://doi.org/10.1145/2445196.2445218

[37] Robert Lindsay McWilliams. 1996. *An Investigation Into the Use and Effectiveness of Videotape Self-evaluations of Conducting for Practicing Music Educators.* University of Minnesota.

[38] Laurie Murphy, Sue Fitzgerald, Brian Hanks, and Renée McCauley. 2010. Pair debugging: a transactive discourse analysis. In *Proceedings of the Sixth international workshop on Computing education research* (Aarhus, Denmark) *(ICER '10).* Association for Computing Machinery, New York, NY, USA, 51–58. https://doi.org/10.1145/1839594.1839604

[39] Alannah Oleson, Benjamin Xie, Jean Salac, Jayne Everson, F Megumi Kivuva, and Amy J Ko. 2022. A Decade of Demographics in Computing Education Research: A Critical Review of Trends in Collection, Reporting, and Use. In *Proceedings of the 2022 ACM Conference on International Computing Education Research* (Lugano and Virtual Event, Switzerland) *(ICER 2022).* ACM. https://doi.org/10.1145/3501385.3543967

[40] Ernesto Panadero, Julia Klug, and Sanna Järvelä. 2016. Third wave of measurement in the self-regulated learning field: when measurement and intervention come hand in hand. *Scandinavian Journal of Educational Research* 60, 6 (Nov. 2016), 723–735. https://doi.org/10.1080/00313831.2015.1066436

[41] Luc Paquette, Jaclyn Ocumpaugh, Ziyue Li, Alexandra Andres, and Ryan Baker. 2020. Who's Learning? Using Demographics in EDM Research. *Journal of Educational Data Mining* 12, 3 (2020), 1–30.

[42] Nick Parlante. 2017. Python Practice. https://codingbat.com/python. Accessed: 2022-4-20.

[43] Michael Quinn Patton. 2014. *Qualitative Research & Evaluation Methods: Integrating Theory and Practice.* SAGE Publications.

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

[45] Paul R Pintrich and And Others. 1991. *A Manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ).*

[46] Paul R Pintrich and Elisabeth V de Groot. 1990. Motivational and self-regulated learning components of classroom academic performance. *Journal of educational psychology* 82, 1 (March 1990), 33–40. https://doi.org/10.1037/0022-0663.82.1.33

[47] Paul R Pintrich, David A F Smith, Teresa Garcia, and Wilbert J Mckeachie. 1993. Reliability and Predictive Validity of the Motivated Strategies for Learning Questionnaire (Mslq). *Educational and psychological measurement* 53, 3 (Sept. 1993), 801–813. https://doi.org/10.1177/0013164493053003024

[48] James Prather, Lauren Margulieux, Jacqueline Whalley, Paul Denny, Brent N Reeves, Brett A Becker, Paramvir Singh, Garrett Powell, and Nigel Bosch. 2022. Getting By With Help From My Friends: Group Study in Introductory Programming Understood as Socially Shared Regulation. https://doi.org/10.1145/3501385.3543970

[49] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19).* Association for Computing Machinery, New York, NY, USA, 531–537. https://doi.org/10.1145/3287324.3287374

[50] Thomas W Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M Kazerouni, Brett A Becker, Andrew Petersen, Luke Gusukuma, Stephen H Edwards, and David Babcock. 2020. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20).* Association for Computing Machinery, New York, NY, USA, 356–362. https://doi.org/10.1145/3341525.3387373

[51] Alex Radermacher, Gursimran Walia, and Richard Rummelt. 2012. Improving student learning outcomes with pair programming. 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, 87–92. https://doi.org/10.1145/2361276.2361294

[52] Adalbert Gerald Soosai Raj, Jignesh M Patel, Richard Halverson, and Erica Rosenfeld Halverson. 2018. Role of Live-coding in Learning Introductory Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '18, Article 13).* Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3279720.3279725

[53] Fernando J Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. 2017. Exploring the Pair Programming Process: Characteristics of Effective Collaboration. ACM Press, 507–512. https://doi.org/10.1145/3017680.3017748

[54] Barbara Rogoff, Maureen Callanan, Kris D Gutiérrez, and Frederick Erickson. 2016. The Organization of Informal Learning. *Review of Research in Education* 40, 1 (March 2016), 356–401. https://doi.org/10.3102/0091732X16680994

[55] 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

[56] Bernhard Schmitz and Franziska Perels. 2011. Self-monitoring of self-regulation during math homework behaviour using standardized diaries. , 255–273 pages. https://doi.org/10.1007/s11409-011-9076-6

[57] Alan H Schoenfeld. 1992. On paradigms and methods: What do you do when the ones you know don't do what you want them to? Issues in the analysis of data in the form of videotapes. *Journal of the Learning Sciences* 2, 2 (April 1992), 179–214. https://doi.org/10.1207/s15327809jls0202_3

[58] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21).* Association for Computing Machinery, New York, NY, USA, 164–170. https://doi.org/10.1145/3430665.3456382

[59] Raj Shrestha, Juho Leinonen, Arto Hellas, Petri Ihantola, and John Edwards. 2022. CodeProcess Charts: Visualizing the Process of Writing Code. In *Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22).* Association for Computing Machinery, New York, NY, USA, 46–55. https://doi.org/10.1145/3511861.3511867

[60] Katta Spiel, Oliver Haimson, and Danielle Lottridge. 2019. How to do better with gender on surveys: A guide for HCI researchers. *ACM Interactions* 26, 4 (2019).

[61] Michael A Stadler. 1995. Role of attention in implicit learning. *Journal of experimental psychology. Learning, memory, and cognition* 21, 3 (May 1995), 674–685. https://doi.org/10.1037/0278-7393.21.3.674

[62] Claude Steele. 2011. Stereotype Threat and African-American Student Achievement. In *The Inequality Reader* (2 ed.). Routledge, 276–281. https://doi.org/10.4324/9780429494468-31

[63] S Stumpf, A Peters, S Bardzell, M Burnett, D Busse, J Cauchard, and E Churchill. 2020. Gender-Inclusive HCI Research and Design: A Conceptual Review. *Foundations and Trends in Human–Computer Interaction* 13, 1 (March 2020), 1–69. https://doi.org/10.1561/1100000056

[64] S Christian Wheeler and Richard E Petty. 2001. The effects of stereotype activation on behavior: A review of possible mechanisms. *Psychological bulletin* 127, 6 (Nov. 2001), 797–826. https://doi.org/10.1037/0033-2909.127.6.797

[65] 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* (Jan. 2019), 1–49. https://doi.org/10.1080/08993408.2019.1565235

[66] Lisa Yan, Annie Hu, and Chris Piech. 2019. Pensieve: Feedback on Coding Process for Novices. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19).* Association for Computing Machinery, New York, NY, USA, 253–259. https://doi.org/10.1145/3287324.3287483

[67] Qian Zhang and Teomara Rutherford. 2022. Grade 5 students' elective replay after experiencing failures in learning fractions in an educational game: When does replay after failures benefit learning?. In *LAK22: 12th International Learning Analytics and Knowledge Conference* (Online USA). ACM, New York, NY, USA. https://doi.org/10.1145/3506860.3506873

[68] Barry J Zimmerman. 2000. Attaining Self-Regulation: A Social Cognitive Perspective. In *Handbook of Self-Regulation*, Monique Boekaerts, Paul R Pintrich, and Moshe Zeidner (Eds.). Academic Press, San Diego, 13–39. https://doi.org/10.1016/B978-012109890-2/50031-7

[69] Barry J Zimmerman. 2002. Becoming a Self-Regulated Learner: An Overview. *Theory into practice* 41, 2 (May 2002), 64–70. https://doi.org/10.1207/s15430421tip4102_2

[70] Barry J Zimmerman and Anastasia Kitsantas. 1999. Acquiring writing revision skill: Shifting from process to outcome self-regulatory goals. *Journal of educational psychology* 91, 2 (1999), 241–250. https://doi.org/10.1037/0022-0663.91.2.241

[71] Barry J Zimmerman and Adam R Moylan. 2009. Self-Regulation: Where Metacognition and Motivation Intersect. In *Handbook of metacognition in education*. Routledge, 311–328.