Beyond the Final Code: Understanding Student Programming Behaviors through Process Journals

Alex Chao
alexchao@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Haolin Li
hal180@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Kyla Ma
kym003@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Adrian Nazzal
anazzal3@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Ayaan M. Kazerouni
ayaank@calpoly.edu
California Polytechnic State University
San Luis Obispo, CA, USA

Leah Perlmutter
lrp87@cornell.edu
Cornell University
Ithaca, NY, USA

Adalbert Gerald Soosai Raj
asoosairaj@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Abstract

As students can now use Generative AI to easily solve introductory programming problems, there is a greater need to focus pedagogy on the programming process, not just the final product. Process journals, where students document their problem-solving steps while working on a coding problem, can encourage students to reflect on their learning and make their thinking visible to instructors. We qualitatively analyzed 48 process journals from an introductory programming course to understand student behaviors and interpret these behaviors through the theoretical lens of self-regulation. Our findings revealed ways in which students understand and approach the programming task, how they test and debug their code, how they consulted resources, and how they reflected or managed their learning. We classified these behaviors as positive, negative, or neutral based on how much self-regulation and engagement they showed. Although some students were not able to complete the basic requirements or showed little extra engagement after satisfying the base requirements, most students showed behaviors beyond the minimum requirements with the process journal and some demonstrated thinking beyond solving the problem. In addition, we found that the behaviors students showed were not strongly related to their exam score or their self-reported metacognitive self-regulation metric. Our study presents process journals as a tool that can give educators insight into both desirable and undesirable behaviors of the stages students go through when solving programming problems. This allows educators to stage interventions that support student learning and metacognitive development.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, Washington, DC, USA
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS Concepts

• **Social and professional topics → Computing education**.

Keywords

metacognition, self-regulated learning, process journals, introductory programming, student behaviors

ACM Reference Format:
Alex Chao, Haolin Li, Kyla Ma, Adrian Nazzal, Ayaan M. Kazerouni, Leah Perlmutter, and Adalbert Gerald Soosai Raj. 2026. Beyond the Final Code: Understanding Student Programming Behaviors through Process Journals. In . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Today, there is a more pressing need than ever for an emphasis on process-oriented learning. Process-oriented frameworks in computing education have been shown to improve problem-solving outcomes and metacognitive development [8, 16]. However, the general availability of AI coding tools makes it easy for students to simply one-shot programming problems, diminishing their learning [28] Moreover, feedback in commonly used automated assessment tools tends to focus on the final steps in a process such as functionality and programming standards [5, 27]. Such tools that allow multiple submissions, e.g, autograders in computing courses, can drive instructors and students to focus on solutions without attending to the process that gets one there.

Prior work has attempted to make aspects of the programming process visible for the purpose of assessment, including examples to assess problem comprehension [36], test cases to assess thoroughness and validity [5], programming plans to give early feedback on time management [17], and process replays to encourage post-hoc reflection on the programming process [37]. These types of artifacts are useful since they are checkable, but are limited since they give us insight into students' programming process at discrete time points. For example, a failing test case may indicate that the

student has a bug in their program, and in a subsequent submission we may see *that* the bug was fixed, without any insight into what the student thought or felt between those events.

A more revealing form of assessment is reflective writing, where students document their own problem-solving and learning process. Reflective work has been used to understand how students perceive their learning after a class session or how they approach a specific aspect of problem solving [13, 18, 21, 32, 38]. However, we do not yet know how students engage well with writing that asks them to document their entire problem solving process in the context of a coding problem. It is unclear what students will express in their problem-solving journey and what insights educators can gather from these reflections. This is of interest now as educators currently have concerns with encouraging and assessing student learning. To address these gaps, we analyze *process* journals that ask students to write about what they did and learned while working through a coding problem. We analyze the behaviors exhibited by students using a theoretical lens of self-regulation.

Our study answers the following research questions:

(1) **What kinds of behaviors are exhibited in student process journals in an introductory computing course?**
(2) **How are these behaviors associated with students' exam and self-regulation scores?**

Our results reveal the many ways in which students engaged with process-oriented learning. Although there was a distribution of engagement and self-regulatory behavior, in general, our students exhibited positive behavior such as reflecting on the causes of their issues and working through debug cycles. We posit that when we as educators emphasized the importance of process, students met us where we were and seriously engaged with the journals.

2 Background and Related Work

2.1 Process Journals

Reflective writing has been used in both non-computing and computing contexts to improve students' metacognition and give instructors insight into student behaviors. Students who kept a journal throughout the term and reflected on their learning after each time they engaged with course material appreciated the ownership and awareness of their learning and developed stronger metacognitive and critical thinking skills [22, 38]. However, Nückles et al. found that the benefits of prompting students in learning journals may diminish over time. Their study found that students who were explicitly prompted started out with more frequent occurrences of metacognitive strategies than students who only received general guidance; however, by the end the strategies appeared at similar frequencies for both groups [21]. Although we also gave explicit prompts in our journals, the execution was different. In Nückles et al. [21], students were asked to reflect on their learning from each class session whereas in our context we gave students prompts to respond to while solving a problem.

In contrast to writing assignments asking students to reflect on their learning in general, work in computing has prompted students to discuss their software engineering behaviors. For example, students used process journals to document how they used GitHub Copilot when programming in large code bases [32]. When using process journals to think about how to improve their software development process, many of the student plans were vague [35]. This suggests a difficult balance of how much students should be prompted in their reflective writing to be helpful for both their learning and metacognitive development. We designed our process journals to have a medium balance by providing a structure that students should follow while still leaving open flexibility for students to elaborate on their own process.

Our work uses process journals to understand not only students' reflective processes but also their behavior when solving programming problems.

2.2 Student Programming Behaviors

Prior work on programming behaviors has studied both the code writing itself and behavior surrounding the programming. Studies have shown that keystroke, compilation, and error data can be predictive of student success [4, 10, 23]. Other work has found that students who submit programs earlier tend to perform better [1, 6, 12], with some recommendations that educators provide guidelines on when/how to start assignments.

A different line of work examines another aspect of student behavior, focusing on the process and mindset of students while problem solving. Students use a wide range of strategies to seek information and use resources to aid their programming [2, 20]. Additionally, there are differences in the behaviors of high- and low-performing students, with low-performing students more likely to not understand a debug cycle, struggle to apply material learned in class, and stop work when they have the right answer rather than understanding why it is correct [11, 14]. In addition to the comparison of high- and low-performing students, our work adds to the understanding of student behaviors when programming through analyzing their process of solving a problem and responding to any issues they face.

2.3 Defining Self-Regulation

We focus on self-regulation in this study due to the nature of our process journal. The process journal was part of every programming assignment, rather than a self-report instrument. Our goal was to ask students to document their work while solving a specific programming problem. As a result, the journals are shaped by the assignment guidelines and by what students choose to record. This makes them useful for examining the presence or absence of actions such as planning, monitoring progress, using strategies, responding to difficulties, and evaluating solutions. These observations align closely with *self-regulation*, which we define in this section and expand on in the next section.

Since there is no agreed definition among the terms self-regulation, self-regulated learning, and metacognition [3, 30], we use Prather et al.'s definition of **self-regulation** as *the process of executing cognitive control during a learning or problem-solving task*, with an attention to learners' behaviors and use of strategies as they work towards academic goals [24, 26]. Prather et al. distinguish self-regulation from metacognition, which they describe as *knowledge about one's own cognitive control*, including activities such as planning, monitoring, understanding, and evaluating progress [3, 9, 26]. Although *metacognition* and *self-regulation* both describe *cognitive*

control and have overlapping definitions, process journals help us see *visible traces* of self-regulation, rather than making claims about students' complete internal cognitive processes [3].

2.4 Self-Regulation in Learning Programming

Self-regulation is particularly relevant in programming education because programming is cognitively demanding for novices [19, 31]. As a result, researchers have examined how students regulate their learning while programming such as by focusing on students' *behaviors and strategies* as they work on programming tasks [7, 15, 33]. The work most relevant to our study in this category is Silva et al.'s study, which identified learning strategies used by 51 students in an introductory programming course [33]. In this study, students responded to reflection prompts at three points during the problem-solving process: 1) after reading the question before starting, 2) after encountering an error, and 3) after solving the problem. Our study draws on Silva et al.'s codebook but adapts it inductively based on our different context and analytic focus. In addition, whereas Silva et al.'s prompts were used solely for research purposes and administered only to study participants, our process journal was embedded as a required component of the programming assignment for all students with an educational aim.

Other studies of self-regulation education examine how to *support* students' self-regulation. In computing education, Xie et al. developed a code replay tool to support student reflections through reviewing their own programming progress [37]; Zarestky et al. studied the effects of reflective writing on a graduate computing course [38]. Although the studies above showed promising results on improving student self-regulation, studies in learning science have raised concerns about challenges in seeing growth in self-regulation skills when students are provided with prompts, or in the long-term [21, 29].

With the above discussion, this study contributes to research on self-regulation in programming in two ways. First, it examines process journals as both an instructional activity and a source of evidence about students' visible self-regulatory behaviors. We focus on observing students' behavior when they are asked to write process journals, rather than making claims about their direct effect on learning. Rather than studying self-regulation only as an individual trait or eliciting it primarily through interviews, think-aloud protocols, or case studies, we analyze how students report their problem-solving processes within a course-embedded activity with an explicit instructional purpose [7, 15, 33]. This framing motivates RQ1, which asks what behaviors are visible in students' process journals, and we use theories of self-regulation to analyze these behaviors. Second, in RQ2, we examine whether these visible behaviors are associated with students' self-assessed self-regulation and programming performance. We compare students' self-reported self-regulation with the types and frequencies of self-regulatory behaviors they show in their process journals, as well as with their performance outcomes.

3 Methods

3.1 Context

The process journal data collection was conducted in a 10-week introductory Python programming course with 286 students enrolled at a large public research-intensive university in the United States. This course was open to both computer science majors and non-majors and had no prerequisites. Six homework assignments were given throughout the course.

Beginning with the second homework, the final question was a coding question, where a problem statement was given with a function description, parameters, return type, and a sample test case. In addition to submitting a Python file for this question, students were required to submit a process journal to document their problem-solving approach alongside screenshots of their code and the errors they faced. There were no separate programming assignments besides these coding problems. This design choice was made to encourage students to focus on the process behind solving the problem rather than fixating on passing autograder outputs. Four required components were specified for the journals: 1) *Understanding the Problem*, a short explanation of the problem; 2) *First Version of Code*, a screenshot or pasted version of the initial implementation; 3) *Running and Testing*, a record of at least two additional tests that were run with their code, along with the expected and actual results; and 4) *Final Version of Code*. Optional components included: *Debugging and Adjustments*, explanations of identifying errors and implementing their fixes; *Further Testing and Refinement*, documentation of any further tests or changes made after debugging; and *Resources Consulted*, a list of any web resources, peers, or other references used. Students were provided with a sample process journal which is included in the supplemental material.

3.2 Data Collection

To obtain a more representative set of process journals, we used stratified sampling to select process journals from four quadrants based on a student's self-regulation and final exam scores. During Week 10, we administered the 12-item self-regulation factor from the Motivated Strategies for Learning Questionnaire (MSLQ) [25], a validated instrument that captures a student's metacognitive behaviors. We classified a score of at least the median of 4.58 out of 7 as high and below the median as low. For the exam scores, we used a cut-off of 27 out of 34 points as high, as this was the threshold for receiving a satisfactory pass per the specifications grading scheme we used.

We chose process journals from the fifth homework (weeks 7-8) because it was assigned later in the course, when students had more practice and familiarity with submitting the journals. The last homework asked students to debug existing code rather than write their own, so we decided to use the penultimate one instead. Choosing a later homework also had the added benefit of being closer to when we collected MSLQ data. The total number of process journals in each quadrant after filtering out empty submissions were 53 in high MSLQ/high exam, 50 in high MSLQ/low exam, 40 in low MSLQ/high exam, and 49 in low MSLQ/low exam. We randomly sampled 12 students from each quadrant to obtain a total of 48 process journals.

The problem students answered asked them to implement a 2D variant of the rainfall problem [31] where the primary concept was iterating over 2D lists using a nested for loop. Students were told to ensure that the average rainfall for an empty list should be zero, not an error. The full problem description is available in our supplement.

3.3 Data Analysis

The first four authors participated in the coding process. We developed our codebook through a combined deductive and inductive process, beginning with an established framework and iteratively refining it as we interpreted our data. Prior to coding, we reviewed process journals from an earlier homework to calibrate our sense of what student-written journals looked like in this course context. Then, we surveyed literature to identify a coding scheme suited to self-regulated learning behaviors in introductory programming. We identified Silva et al. [33] as the best fit given its explicit grounding in self-regulation theory and its focus on introductory programming students. We then discussed each code in Silva et al.'s codebook, filtering out some that were not relevant in our context to create an initial codebook that we used to analyze our process journals.

We divided the journals into sets of six, individually applying the initial codebook for each batch. We added codes as needed, then met to collectively produce a shared codebook and resolve any disagreements. After the first batch, we synchronously coded two process journals together to align our understanding. Then, the batch process was repeated until all journals were collectively discussed. After finishing the last batch, we revisited the previous journals to recode them with our final codebook.

Once we began our analysis, we diverged from the initial codebook more by adding, removing, and combining some codes. Because Silva et al.'s codebook was built from data spanning multiple phases of an entire course rather than a single bounded assignment, several codes did not directly map onto our context. Our process journals were tied to one homework problem which offered less room for the kinds of multi-step decomposition the original codes assumed. As we coded, we edited definitions, added codes, and resolved ambiguities in real time, treating each new journal as an opportunity to stress-test the codebook rather than a fixed instrument to apply mechanically. For instance, "divide the problem into parts" and "plan program logic" collapsed into effectively the same move at this task scale, and we merged them into a single code: *pseudocode and planning program logic*.

We give some examples of decisions we made during our coding process.

- Removing or merging codes judged too close to the assignment prompt to be informative (e.g., *tested after adjustments* was dropped as it was implied by the inclusion of the final code submission after encountering an error); *first* and *final version of code* were dropped since all submitted journals included them.
- Splitting codes for granularity (e.g., distinguishing a general *knowledge gap* from a *self-aware knowledge gap*).
- Adding codes not present in the original scheme but salient in this data source (e.g., handwritten format, AI-related help-seeking via search or in-line Copilot suggestions, screenshots of code that are not discussed in text).
- Tightening scope so codes reflected genuine variation rather than required elements of the assignment (e.g., *think about issues in advance* was redefined to exclude mention of the divide-by-zero case, since that was given in the prompt itself).

4 Results

4.1 RQ1: Student Behaviors

We discuss how our analysis captures the behaviors exhibited in student process journals by giving an overview of our codebook followed by analyzing specific codes and presenting a case study of a notable process journal.

4.1.1 Distribution of Codes. We found a wide range of behaviors shown in student process journals, with the counts of each label (out of 48 submissions) presented in Table 1. For each behavior that students exhibited, we classified the label as negative, neutral, or positive. We discuss the reasoning for our classification below.

Our four negative codes consisted of undesirable behaviors that indicated a lower level of engagement with the process journals or a lack of understanding of the material. In *unimplemented plan*, some students discussed planning for the divide by zero case and then in the very next step did not include it in their code. This could be due to an oversight, but we also suspect that some students may have written their planning portion after already writing their code. In either instance, the student displayed negative behavior. The codes *knowledge gap* and *error not discussed in text* also show gaps in understanding of the material as the student is not familiar with some content and does not recognize that. We also classified *consult in-line AI suggestions* as negative behavior as the two students with this label used the AI-generated code without demonstrating understanding of why their own code did not work.

Our neutral codes include minimally required parts of the process journal as well as behavior that did not give insight into the students' problem-solving. The codes *restating exercise statement*, *use of test-case data*, and *additional test* were given as expectations in the problem statement, so their inclusion did not indicate higher levels of engagement with the journals. Some other codes initially appear to be reflective behavior, but in practice, they were expressed minimally in the context of the students' journals. For example, *think about issues in advance* and *analysis of Python's interpreter output* were generally surface-level restatements of the divide by zero hint given in the prompt or simple syntax errors. Similarly, *define necessary programming resources* and *explaining implementation of code* did not give much more information than what was already written in their code. We also classified some more behavior that did not seem aligned with engagement nor self-regulation as neutral: *help request, collaboration,* and *give up and create new code*.

Finally, the behaviors classified as positive showed deeper engagement with the process journals and evidence of self-regulatory behavior. Some showed good coding practices, e.g., *use of debug strategies, pseudocode and planning program logic, debug cycle, presence of comments,* and *code refactoring*. Others we included as higher effort, including *use of diagrams, handwritten, intermediate steps to reach first version,* and *reasoning through implementation of fix*. Behaviors we classified as self-regulatory included an awareness of their own process and learning. These include those that were

Table 1: Codebook for Behaviors Students Expressed in Process Journals with Count (#), Positive (+) Neutral (/) Negative (-) Association, and Definition. Context for which steps were required is provided in square brackets.

Label	#	+/-	Definition
Understanding Problem [at least one in this category required]			
Restate or annotate exercise statement	43	/	Writes about or annotates the problem instructions
Pseudocode and plan program logic	30	+	Maps out how the function will be written before implementation
Define necessary programming resources	28	/	States programming data types or structures that will be used, outside of those mentioned in the write-up (e.g., if statement, variables)
Use of diagrams	8	+	Includes a diagram for outlining the problem (e.g., flowchart)
Think about issues in advance	7	/	Addresses and expands on issues that may come up in code-writing
Writing First Version of Code			
Explain implementation of code	19	/	Explains the code by providing a high-level approach or steps taken
Intermediate steps to reach first version	7	+	Shows a partial version, and acknowledges it is partial
Presence of comments	3	+	Includes comments in their function
Running and Testing			
Use of test-case data [required]	48	/	Runs the code using the provided sample test
Additional test [required]	40	/	Runs the code using self-created tests
Debugging and Adjustments			
Reflect on the issue's causes	31	+	Gives reasoning for why an error or unexpected event is triggered
Debug cycle	28	+	Describes a process of encountering an error and attempting to fix
Analysis of Python's interpreter output	21	/	Refers to an error seen in the terminal and discussing its implications
Reason through implementation of fix	16	+	Explains changes to code or mentally tracing through code
Code refactoring	5	+	Changes code structure without changing its correct functionality
Give up and create new code	3	/	Deletes lines of code to try a new approach
Use of debug strategies	3	+	Uses strategies like implementing print statements or helper functions
Read exercise statement	1	/	Refers back to the exercise statement to fix a bug
Self Regulation			
Emotional and motivational regulation	15	+	Discusses or expresses emotion or motivation
Reflect on learning	11	+	Makes a visible effort to reflect on the learning process or the content
Effort self-assessment	8	+	Discusses the level of effort spent or the difficulty level of the problem
Knowledge gap	7	-	Indicates no understanding of why something is happening
Self-aware knowledge gap	5	+	Indicates awareness of cause of confusion
Anticipate an error	3	+	Predicts or expects errors in their code
Resources Consulted			
Establish relationship with previous exercises	10	+	Refers to problem done in lecture or other lesson material
Search for complementary material	5	/	Searches the internet or uses AI
Collaboration	5	/	Collaborates with or asks for help from a classmate
Consult in-line AI suggestions	2	-	Accepts a GitHub Copilot fix or suggestion
Help request	1	/	Asks for help from a senior source (instructor, TA, or tutor)
Misc			
Handwritten	15	+	Includes any amount of handwriting
Unimplemented plan	3	-	Plans something but does not include in code
Error that is not discussed in text	3	-	Includes an error visible in a screenshot but does not address it

reflective: *effort self-assessment*, *reflect on the issue's causes*, *reflect on learning* and those that showed a greater understanding of their coding process: *anticipates an error*, *self-aware knowledge gap*, and *establish relationship with previous exercises*.

4.1.2 Notable Codes. In this section, we further discuss some notable codes, compare across codes, and provide examples from process journals.

The distinction between *knowledge gap* and *self-aware knowledge gap* lies in whether a student encountering an error understands what next step they should take to resolve it. An example of a student who expressed *knowledge gap* was

> "I didn't understand where the 2.3 was coming from. It isn't the average of either row in the 2d list or both rows combined…I tried looking through the program and finding any logic errors…Still confused."

The student did not understand why their error occurred, and even after attempting to debug it, they remained stuck. In contrast, a student with *self-aware knowledge gap* stated

> "But my output looks strange, as there are too many values, there should only be 5 floats in the list. I feel like this has to do with my second for loop. I got a little lost here so I looked up the proper format for nested for loops."

Here, the student was also confused; however, they not only identified where the bug was occurring, but they also were able to make a plan for solving it. Students who were able to pinpoint specific material to review to solve their issue demonstrated a deeper understanding of programming structure and development of learning strategies through searching for material to bridge their lack of knowledge.

A notable example of self-regulatory behavior we observed was when students discussed what they thought a change to their code would do between encountering a bug and testing its fix, which we labeled as *reasoning through implementation of fix*. The most general case of this is demonstrated where a student said

> "In order to fix this error, I fixed the loop so it iterates directly over the rainfall values instead of needing to index through them."

Rather than just noting what changed, this student demonstrated an active understanding of what their code was doing and why the change resolved the issue. Another interesting case comes from a different student who stated

> "The only thing I'm expecting to get an error on is the syntax involving the length of the day…wait…length of the day? I'm supposed to be looking at the length of each month."

In this case, the act of writing about their code itself prompted the discovery of the fix. Students who received this label were not just fixing errors, but demonstrating an understanding of why their fix would work.

Whereas *reasoning through implementation of fix* captures what students do after identifying the cause of an error, *reflect on the issues' causes* captures the moment of identification itself, i.e., when a student pinpoints what went wrong and why. Students received this label when they went beyond simply noticing something is broken and actively reflected on the source of the problem. One straightforward example of this is

> "I encountered an error because I forgot to place commas between the lists in rainfall_sample"

as they identified both the error and its cause in one sentence. A different student shows a deeper version of this, where the reflection is conceptual rather than syntactic:

> "I realized that I was trying to add the total precipitation of each month to a list for no reason, probably because I didn't really think carefully enough about the steps I wanted my code to take beforehand."

Instead of spotting a surface-level mistake, this student recognized a fundamental misunderstanding in their approach. *Reflect on the issues' causes* appeared in 31 of the 48 process journals, reflecting that for most students, debugging was not a random process of trial and error but one grounded in actively identifying what went wrong. Students with this label genuinely reasoned about the cause of their bugs rather than just reporting an outcome.

4.1.3 Level of Engagement with Process Journals. We examine the level of engagement reflected in the process journals through the number of codes we applied as more codes generally meant students showed more of their process. Excluding the *Misc* codes, four students included exactly the required components (*use of test-case data*, *additional test*, and at least one code from the *Understanding Problem* category), 35 students had additional labels in addition to meeting the minimum requirements, and nine students missed one requirement. Figure 1 shows the distribution of students by the total number of labels applied. We describe the outlier with 19 labels in greater detail as a case study in Section 4.1.4.

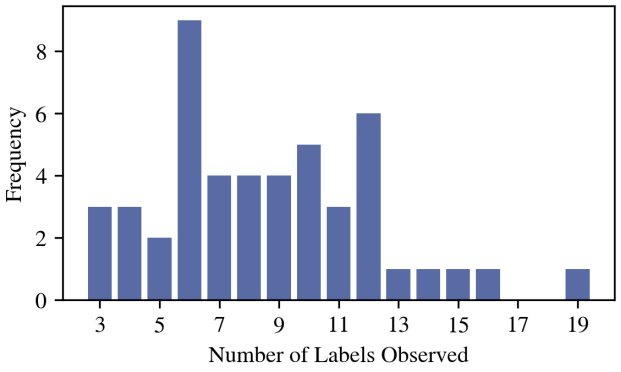


Figure 1: Frequency of Students For Each Label Count

Although nine students submitted process journals that missed a requirement, this did not indicate low overall engagement. All of these students included some additional labels in their process journals, and six of them had more than five additional labels beyond the required components. Eight of the students missed the *additional test* requirement, and one had no label under the *Understanding Problem* category. These students missed a requirement as an oversight, but their process journals were more detailed and

informative than the four students who submitted only the required components.

Finally, looking into not strictly required categories, 34 students had some label in *Debugging and Adjustments*, showing these were the students that reported encountering errors and handling them. 20 students had a label from *Writing First Version of Code*, 19 reported on *Resources Consulted*, and 27 students showed some evidence of *Self-Regulation*.

4.1.4 Case Study. We present a case study of our outlier from Figure 1, who had a total of 19 labels out of 32, to show an example of a high effort, high engagement process journal. Journals of this caliber are rare, and we present it to give richer insight into the potential behaviors a student can exhibit. Although most students submitted a process journal between one and three pages, this student submitted a very detailed and reflective 12-page journal. We discuss some notable features of the journal below.

The student made explicit references to their learning, both from prior experiences and planning for the future. When discussing solving the problem, they reflected on what they have learned from past assignments, stating "I'm so incredibly grateful that I remembered how to append lists and dictionaries" and "I am a lot more aware of how python works with loops now." While working on the current problem, they were also thinking about what they would take away from it, writing "I should utilize comments more as I learn more, though." Unlike some other reflective activities, the process journals in this class are modular and do not explicitly build on each other, so it was even more meaningful when students drew comparisons across their different process journals.

The student not only used incremental development but was also aware of its benefits. At the beginning, they stated "Rather than just trying to write the code right away, I will try testing before anything happens so I have a good idea of what the for loop does." By taking the time to write out their entire process, they were also reflecting on why they took certain steps. Later, they continued with "I am so glad that I printed the values though. This has made me so much more confident knowing what the list does." Here, the student understood how their motivation was impacted by their development process.

In addition to motivational regulation, the student was also very emotionally expressive throughout the journal. A few examples include "The time I spent on the last process journal TRAUMATIZED me", "OH!", "Silly me...What am I doing?", "YES! It works!". Other students expressed emotions in their process journals as well but typically as a single comment.

Finally, the student went above and beyond by both refactoring and writing extra code. During the problem solving process, the student rewrote parts of their code for readability. At the end of the process journal, they also wrote their own automated test generator.

Notably, this student took several debug cycles to complete the problem, which allowed us to observe deeper engagement in the process journal. The student took full advantage of one of the goals of the process journal, which was to provide a valuable learning opportunity for students who do not immediately solve the problem.

4.2 RQ2: Association with Exam and MSLQ Scores

As our work is exploratory in nature towards capturing student behaviors and we only have 12 students per quadrant, we report descriptive rather than inferential statistics. We use this section to give more context into understanding how behaviors shown on the process journals might tie into other metrics.

Although we hypothesized that students with higher exam scores and/or higher MSLQ scores might submit higher quality process journals, we did not find this to be the case. When we looked at the codes in aggregate, we found a relatively equal distribution of students in each quadrant (high/low MSLQ/exam score) for level of detail and presence of extra labels per category.

To see if there were differences for individual labels, we examined the distribution of students with high/low MSLQ and exam scores for each label. Figure 2 shows notable labels we discussed in Section 4.1.2 as well as labels for which there was an uneven distribution of students for either exam score or MSLQ.

Some behaviors that we characterized as positive/negative were associated with high/low exam scores. For example, six of eight students with *effort self-assessment* had high exam scores while all three students with an *error that is not discussed in text* had low exam scores. At the same time, six of seven students with *think about issues in advance* had low exam scores. From the definition of the code, this seems surprising, but in our context the problem statement explicitly instructed students to handle the zero rainfall months. Although the students did expand on the divide by zero error, they may not have foreseen the issue on their own.

For MSLQ scores, we did not find evidence that a student's MSLQ score was associated with exhibiting self-regulatory behavior in their process journal. Although all three students with *unimplemented plan* had low MSLQ scores which may be indicative of lower self-regulatory behavior, this pattern did not persist across other codes. For example, 11 of 15 students who discussed emotions or motivation had low MSLQ scores even though we characterize this as self-regulatory behavior. Moreover, for some other codes which we consider to be self-regulatory, there was a relatively even split among the low/high MSLQ scores, including *reflect on learning*, *self-aware knowledge gap*, and *reasoning through implementation of fix*.

5 Discussion

5.1 Interpretation of Student Behaviors

Although students were given a set of requirements for their process journals, there was a wide variety in the extent to which they followed them. We saw students who recorded many of details and reflections, students who only delivered the minimal requirements, and students who missed requirements. From the label counts, we see that the requirement students were most likely to miss was the *additional test*. As a common pattern, students would run and pass the provided test case and forget the requirement of at least two additional tests. Although additional tests have been a requirement for all programming questions, these students may not have developed a habit of coming up with their own tests, potentially neglecting the value of thoroughly testing their code.

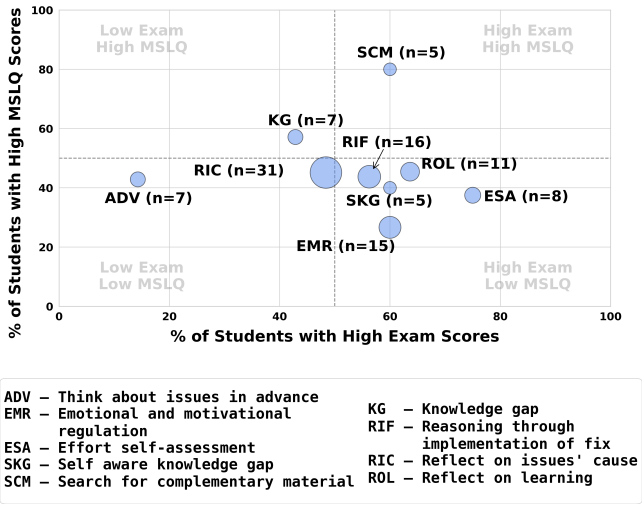


Figure 2: Mapping Labels Across Exam & MSLQ Scores. A circle that is far right along the X axis represents a behavior that tends to show up in students with high exam scores, while higher up on the Y axis is associated with higher MSLQ scores. The area of a circle is proportional to the number of students who exhibited that behavior.

We hypothesize that the positive, negative, or neutral labels that are associated with students' process journals may not be indicative of a student's proficiency. Process journals that were coded with negative labels indicate that the student had areas that needed improvement. The negative labels generally alluded to a lack of understanding, where there were parts within the student's process where they got stuck. These students may benefit from additional support from teaching staff, collaboration, and other resources to strengthen their understanding. However, the converse is not necessarily true.

Process journals receiving positive codes did not always mean that the student did not need help understanding the programming concepts needed to solve the problem. A student may be thorough in pointing out their issues and expressing their thought process throughout their journal, but they may still need assistance from other resources to supplement their knowledge to fully solve the problem. Additionally, a lack of positive labels may not translate to the student needing support to understand the problem. A case where a student may receive few to no positive labels would be if they solved the problem on their first attempt. These students would not have a chance to display positive self-regulatory behavior, as they do not have any debugging cycles where they can reflect on challenges or learning. Because such process journals, with a linear process of immediately reaching a fully working product and a lack of reflection, do not strongly indicate the student's level of understanding, we find that the most informative process journals come from students who find the problem challenging and need to iterate on their solution.

5.2 Relationship to Prior Work

5.2.1 Programming Behaviors. Unlike prior work that found that low-performing students tend to exhibit more negative programming behaviors [11, 14], we did not find strong evidence of a difference in behavior between students with high and low exam scores. This may be due to differences in the nature of the data. We did not directly observe student behaviors nor ask them directly about their programming habits. High performing students may tend to document their process less as they are able to solve their programs with more ease. Additionally, low-performing students may exclude to mention that they still did not understand the code that they submitted.

5.2.2 Self-Regulatory Behaviors. We find behaviors in our students' process journals not present in work by Silva et al. [33] that give us both more detail into how they solved the problems and greater insight into their self-regulatory behavior. In Silva et al.'s paper, students were asked three questions during the coding process: before they started, how they planned to solve the exercise; after encountering an error, what can be done to solve the problems; and after finishing the test cases, what went right and wrong. In contrast, although our process journals also had guidelines, students were given more freedom to openly document their entire problem solving process. This allowed us to capture the following positive behaviors that were not in the scope of how a student would respond to questions from Silva et al.'s work: *intermediate steps*, *debug cycle*, *reasoning through implementation of fix*, *anticipates an error*, *reflect on learning*, and *self-aware knowledge gap*.

All of the negative codes we found were not present in Silva et al. [33]. Although Silva et al. ask students what went wrong, being able to identify that as a student is a metacognitively challenging task. Therefore, through analyzing the student's process, we can find undesirable behavior that they are not aware of themselves. This gives us an alternative way of understanding a student's self-regulation by giving insight into different levels of their regulation, e.g., the difference between a student expressing *knowledge gap* versus *self-aware knowledge gap*.

Although there were several behaviors present in our study that were not in earlier work, we did not find the converse to be true. Some codes from Silva et al. did not appear in our final codebook, but the associated behavior was collapsed into one of our codes. For example, *divide problem into parts* was included under *pseudocode and plan program logic* as our problem did not lend itself to a natural division, and only a few students tried to divide the problem into parts. We excluded two codes from our codebook: *input and output definition*, because the inputs and outputs were provided in our context, and *read annotations*, because it appeared only once in their study.

Our process journals are able to capture the same self-regulatory behavior as earlier work while also providing greater insight into other student behaviors, including negative ones. In prompting students to document their entire problem-solving process, we gain a more 'authentic' view into their behavior. This opens up more possibilities for educators to stage interventions to support student learning. In identifying negative behavior that a student is unaware of, educators can teach students both the content and better self-regulatory habits.

5.3 Process Journals and Self-Regulation

Returning to our theoretical framing, we interpret the process journal as a course artifact that both encourages and reveals traces of self-regulation. Because process journals ask students to document their problem-solving process, we can observe behaviors related to planning, monitoring progress, responding to errors, using strategies, evaluating solutions, and reflecting on personal learning. At the same time, given that process journals were completed as part of an assignment with specific requirements and constraints, we do not treat them as complete representations of students' internal cognitive processes or as direct measures of self-regulation.

Nevertheless, examining how students chose to externalize their problem-solving process connects to self-regulation in several ways. First, it supports our positive, negative, and neutral label classification, which combines the process journal context with self-regulation theory to interpret the behaviors students exhibited. This classification allows us to identify visible evidence of self-regulatory behaviors in process journals, ranging from concrete strategies such as *use of diagrams* to higher-level reflection such as *reflect on learning*. It also allows us to identify potential gaps in self-regulation, such as when students produced an *unimplemented plan* or showed an error in a screenshot without discussing it in text. Second, beyond isolated labels, we also considered the overall effort students invested in their process journals. Our distinction among incomplete, necessary-only, and beyond-necessary process journals, as well as our analysis of the number of labels observed, was motivated by the wide variation in engagement we saw while analyzing the journals. Although effort is not equivalent to self-regulation, we found that greater engagement with the journal may create more opportunities for students to demonstrate and practice self-regulatory behaviors.

This interpretation also helps explain why MSLQ self-regulation scale scores were not strongly associated with the label counts, despite our consideration that many labels are strong reflections of self-regulation. The MSLQ captures students' self-reported self-regulation at a broad level, whereas our labels capture visible, task-specific behaviors in a single programming assignment. At the individual student level, a process journal is therefore not a direct reflection of self-regulatory capacity. A student may report strong self-regulatory habits on the MSLQ but write a less detailed journal because they solved the problem quickly, encountered few errors, or did not see value in documenting steps beyond the requirements. Conversely, a student with a lower MSLQ score may produce a richly coded process journal because they encountered more errors, expressed more uncertainty, or used the journal as a space to work through confusion. Thus, label counts should be interpreted as measures of documented engagement with the process journal, rather than as direct measures of students' self-regulatory capacity.

5.4 Implications for Practice

A core question for educators is whether process journals are useful instructional artifacts in programming courses. Although our study does not directly measure the instructional effectiveness of process journals, our findings shed light on when and how they may be valuable.

Our results suggest several potential benefits of introducing process journals. Some students used the journals as an opportunity to actively engage with and reflect on their learning. The student described in our case study, for example, used the process journal to document planning, debugging, emotional responses, strategy use, and reflection on prior learning. For students who engage with the artifact in this way, process journals may provide a routine opportunity to practice making their problem-solving process explicit. Additionally, process journals can give instructors insight into how students approach programming problems, including aspects of novice problem-solving that experienced course staff may otherwise take for granted. During the course, staff used process journals to identify and discuss students' knowledge gaps, suggesting that journals can help instructors monitor not only the final product of student learning, but also the process that leads to it. Similarly, our results point to specific areas where students may need additional support. For example, students frequently omitted the required *additional tests*, suggesting that some students may not yet understand the purpose or value of creating their own test cases beyond the provided examples.

At the same time, our findings suggest that the effectiveness of process journals depends on how they are implemented. Because our process journals were tied to specific programming questions, the design of those questions shaped the kinds of self-regulatory behaviors students could display. Our codebook differed from Silva et al.'s codebook in part because our data came from a single bounded programming assignment rather than a broader set of prompted reflections across a course [33]. This suggests that if educators want students to use process journals to practice self-regulation, the accompanying programming tasks should be designed to support richer engagement with planning, monitoring, debugging, strategy use, and reflection.

Educators should also consider how process journals are designed, assigned, and assessed. In our context, the required components gave students a clear structure, but they may also have encouraged a "check the box" approach in which students completed only the minimum required elements. One possible alternative is to place constraints on the form or scope of the journal, such as page limits or time-based reflection windows, rather than prescribing every required content category. This may give students more freedom to document the aspects of their process that were most meaningful, while reducing the tendency to treat the journal as a checklist. Another consideration is to have the prompts be adjusted to the specific problem they are accompanying. These alternatives align with Nückles et al.'s discussion of prompted journal tasks, where repeated prompted writing tasks appeared less effective for long-term self-regulation [21].

Overall, our findings suggest that process journals are promising but not automatically effective instructional artifacts. They can help students externalize their problem-solving process and help instructors identify where students encounter difficulties. However, their value depends on careful alignment among the programming task, the journal prompt, and the grading structure. Instructors who adopt process journals should therefore consider not only asking students to document their process, but also designing tasks and assessments that make genuine planning, monitoring, debugging, and reflection worthwhile.

6 Limitations and Threats

Our study was conducted in a single course at a single institution, so the results may not be applicable to other contexts. However, there was significant overlap in self-regulatory behaviors with work from Silva et al. [33] which operated in a much different context (an online course in Brazil with mainly adult learners) which adds confidence to the generalizability of these behaviors.

The format of the process journals may have limited the behaviors we observed. As there was a structured list of requirements and an example process journal given, students may have expressed different behaviors in a different format. Even if a student had an additional thought or tried a different method, because it did not fit well under the elements of a process journal specified in the instructions, they may have felt discouraged from noting the thought down. To help mitigate this, we chose to analyze the latest process journal where students could have solidified their own style. At the same time, however, we only analyzed one out of five process journals, and the quality of process journals may have changed over time. Finally, since the process journals are self-reported, they may not be representative of what the students actually did.

7 Conclusion

Our initial goals when designing the process journal were to reemphasize the humanistic aspects of programming and problem solving such as productive struggle, recovering from broken code, and emotional releases after successful debugging, all of which can be circumvented by just outsourcing the problem to GenAI. We aimed to encourage students to slow down and embrace the entire problem solving process and show them that as educators, we cared more about their process than just their final code solution.

Although a possible result of our work was that students approached the journals with minimal effort and treated it as busy work, we did not find this to be the case. Even at the end of the course, students still engaged with the process journals and possibly understood their value as a learning tool. Most of our students exhibited behaviors that went above and beyond the requirements. Some students even went beyond the scope of answering the optional prompts and put extra effort into their journals by adding diagrams and handwritten components. Even for the students who submitted minimal process journals, we believe that working through the reflective steps in the prompt is helpful for students.

Our work further opens up process journals as both a research and pedagogical artifact that can give deep insight into a student's thinking, enable interventions based on student responses, and correlate student behaviors with student performance and other metrics.

As CS1 learning objectives are shifting due to GenAI [34], a student's ability to understand what they do and do not know remains ever important. We found that students who engaged deeply with their process journals showed self-regulatory behavior that supported this skill. Guiding students through this process can be a scaffolded way of developing this increasingly important skillset.

Acknowledgments

We thank the anonymous reviewers as well as Sona Chitchyan and Audria Montalvo for their helpful feedback in improving earlier drafts of this paper. This work was supported by the Charles Lee Powell Foundation, the National Science Foundation under Grant Nos. 2417787, 2506612, and 2417531 and the GenAI in CS Education Consortium.

References

[1] Francisco Enrique Vicente Castro, Juho Leinonen, and Arto Hellas. 2022. Experiences With and Lessons Learned on Deadlines and Submission Behavior. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '22)*. Association for Computing Machinery, New York, NY, USA, Article 25, 13 pages. doi:10.1145/3564721.3564728

[2] Elizabeth B. Cloude, Jiayi Zhang, Ryan S. Baker, and Eric Fouh. 2024. Procrastination vs. Active Delay: How Students Prepare to Code in Introductory Programming. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 214–220. doi:10.1145/3626252.3630907

[3] Daniel L. Dinsmore, Patricia A. Alexander, and Sandra M. Loughlin. 2008. Focusing the Conceptual Lens on Metacognition, Self-regulation, and Self-regulated Learning. 20, 4 (2008), 391–409. doi:10.1007/s10648-008-9083-6

[4] John Edwards, Juho Leinonen, and Arto Hellas. 2020. A Study of Keystroke Data in Two Contexts: Written Language and Programming Language Influence Predictability of Learning Outcomes. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 413–419. doi:10.1145/3328778.3366863

[5] Stephen H. Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE04)*. ACM, 26–30. doi:10.1145/971300.971312

[6] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (Berkeley, CA, USA) *(ICER '09)*. Association for Computing Machinery, New York, NY, USA, 3–14. doi:10.1145/1584322.1584325

[7] Katrina Falkner, Rebecca Vivian, and Nickolas J.G. Falkner. 2014. Identifying Computer Science Self-Regulated Learning Strategies. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (New York, NY, USA, 2014-06-21) *(ITiCSE '14)*. Association for Computing Machinery, 291–296. doi:10.1145/2591708.2591715

[8] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2018. *How to design programs: an introduction to programming and computing*. Mit Press.

[9] John H. Flavell. 1979. Metacognition and Cognitive Monitoring: A New Area of Cognitive–Developmental Inquiry. 34, 10 (1979), 906–911. doi:10.1037/0003-066X.34.10.906

[10] Marcus E. Gubanyi and Leen-Kiat Soh. 2025. Evaluating Programming Assignments and Compile-and-Run Prompts in CS1 by Observing Student Programming Behaviors. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1 (ICER '25)*. Association for Computing Machinery, New York, NY, USA, 210–226. doi:10.1145/3702652.3744204

[11] Amanpreet Kapoor, Paul Denny, Leo Porter, Stephen MacNeil, and Marc Diaz. 2026. Exploring Student Behaviors and Motivations when using AI Teaching Assistants with Optional Guardrails. In *Proceedings of the 28th Australasian Computing Education Conference (ACE '26)*. Association for Computing Machinery, New York, NY, USA, 22–31. doi:10.1145/3786228.3786233

[12] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2021. Does the Early Bird Catch the Worm? Earliness of Students' Work and its Relationship with Course Outcomes. 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, 373–379. doi:10.1145/3430665.3456383

[13] Gary Lewandowski. 2003. Using Process Journals to Gain Qualitative Understanding of Beginning Programmers. *Journal of Computing Sciences in Colleges* 19, 1 (2003), 299–310. https://dl.acm.org/doi/10.5555/948737.948779

[14] Soohyun Nam Liao, Sander Valstar, Kevin Thai, Christine Alvarado, Daniel Zingaro, William G. Griswold, and Leo Porter. 2019. Behaviors of Higher and Lower Performing Students in CS1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 196–202. doi:10.1145/3304221.3319740

[15] Jonathan Liu, Erica Goodwin, and Diana Franklin. 2025. Student Utilization of Metacognitive Strategies in Solving Dynamic Programming Problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2025-02-18) *(SIGCSETS 2025)*. Association for Computing

Machinery, 687–693. doi:10.1145/3641554.3701888

[16] 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. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, 1449–1461. doi:10.1145/2858036.2858252

[17] Joshua Martin, Stephen H. Edwards, and Clifford A. Shaffer. 2015. The Effects of Procrastination Interventions on Programming Project Success. In *Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15)*. ACM, 3–11. doi:10.1145/2787622.2787730

[18] Claire McGuinness and Michelle Brien. 2007. Using reflective journals to assess the research process. *Reference Services Review* 35, 1 (Feb. 2007), 21–40. doi:10.1108/00907320710729346

[19] I.T. Chan Mow. 2008. Issues and Difficulties in Teaching Novice Computer Programming. In *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education* (Dordrecht, 2008), Magued Iskander (Ed.). Springer Netherlands, 199–204. doi:10.1007/978-1-4020-8739-4_36

[20] Silvia Muller, Monica Babes-Vroman, Mary Emenike, and Thu D. Nguyen. 2020. Exploring Novice Programmers' Homework Practices: Initial Observations of Information Seeking Behaviors. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 333–339. doi:10.1145/3328778.3366885

[21] Matthias Nückles, Sandra Hübner, and Alexander Renkl. 2008. Short-Term versus Long-Term Effects of Cognitive and Metacognitive Prompts in Writing-to-Learn. International Society of the Learning Sciences, Inc. https://repository.isls.org//handle/1/3143

[22] Chris Park. 2003. Engaging Students in the Learning Process: The learning journal. *Journal of Geography in Higher Education* 27, 2 (2003), 183–199. arXiv:https://doi.org/10.1080/03098260305675 doi:10.1080/03098260305675

[23] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '15)*. Association for Computing Machinery, New York, NY, USA, 77–86. doi:10.1145/2828959.2828966

[24] Paul R. Pintrich and Elisabeth V. De Groot. 1990. Motivational and Self-Regulated Learning Components of Classroom Academic Performance. 82, 1 (1990), 33–40. doi:10.1037/0022-0663.82.1.33

[25] Paul R. Pintrich and Elisabeth V. De Groot. 1990. Motivated Strategies for Learning Questionnaire (MSLQ). doi:10.1037/t09161-000 Database record.

[26] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (New York, NY, USA, 2020-08-07) *(ICER '20)*. Association for Computing Machinery, 2–13. doi:10.1145/3372782.3406263

[27] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)*. ACM, 41–50. doi:10.1145/3230977.3230981

[28] Jaromir Savelka, Arav Agarwal, Marshall An, Chris Bogart, and Majd Sakr. 2023. Thrilled by Your Progress! Large Language Models (GPT-4) No Longer Struggle to Pass Assessments in Higher Education Programming Courses. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 78–92. doi:10.1145/3568813.3600142

[29] Clara Schumacher and Dirk Ifenthaler. 2021. Investigating Prompts for Supporting Students' Self-Regulation – A Remaining Challenge for Learning Analytics Approaches? 49 (2021), 100791. doi:10.1016/j.iheduc.2020.100791

[30] Dale H. Schunk. 2008. Metacognition, Self-Regulation, and Self-Regulated Learning: Research Recommendations. 20, 4 (2008), 463–467. doi:10.1007/s10648-008-9086-3

[31] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do we know how difficult the rainfall problem is?. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '15)*. Association for Computing Machinery, New York, NY, USA, 87–96. doi:10.1145/2828959.2828963

[32] Anshul Shah, Anya Chernova, Elena Tomson, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2025. Students' Use of GitHub Copilot for Working with Large Code Bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. ACM, Pittsburgh PA USA, 1050–1056. doi:10.1145/3641554.3701800

[33] Leonardo Silva, António Mendes, Anabela Gomes, and Gabriel Fortes. 2024. What Learning Strategies are Used by Programming Students? A Qualitative Study Grounded on the Self-regulation of Learning Theory. *ACM Trans. Comput. Educ.* 24, 1, Article 9 (Feb. 2024), 26 pages. doi:10.1145/3635720

[34] Annapurna Vadaparty, Daniel Zingaro, David H. Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 297–303. doi:10.1145/3649217.3653584

[35] Tammy VanDeGrift, Tamara Caruso, Natalie Hill, and Beth Simon. 2011. Experience report: getting novice programmers to THINK about improving their software development process. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 493–498. doi:10.1145/1953163.1953307

[36] John Wrenn and Shriram Krishnamurthi. 2019. Executable Examples for Programming Problem Comprehension. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. ACM, 131–139. doi:10.1145/3291279.3339416

[37] 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 - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 298–313. doi:10.1145/3568813.3600127

[38] Jill Zarestky, Michelle Bigler, Mollie Brazile, Tobin Lopes, and Wolfgang Bangerth. 2022. Reflective Writing Supports Metacognition and Self-regulation in Graduate Computational Science and Engineering. *Computers and Education Open* 3 (Dec. 2022), 100085. doi:10.1016/j.caeo.2022.100085