Investigating Novices' In Situ Reflections on Their Programming Process

Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J. Ko
University of Washington
Seattle, Washington, USA
dloksa@uw.edu, bxie@uw.edu, kwikh@uw.edu, ajko@uw.edu

ABSTRACT

Prior work on novice programmers' self-regulation have shown it to be inconsistent and shallow, but trainable through direct instruction. However, prior work has primarily studied self-regulation retrospectively, which relies on students to remember how they regulated their process, or in laboratory settings, limiting the ecological validity of findings. To address these limitations, we investigated 31 novice programmers' self-regulation in situ over 10 weeks. We had them to keep journals about their work and later had them to reflect on their journaling. Through a series of qualitative analyses of journals and survey responses, we found that all participants monitored their process and evaluated their work, that few interpreted the problems they were solving or adapted prior solutions. We also found that some students self-regulated their programming in many ways, while others in almost none. Students reported many difficulties integrating reflection into their work; some were completely unaware of their process, some struggled to integrate reflection into their process, and others found reflection conflicted with their work. These results suggest that self-regulation during programming is highly variable in practice, and that teaching self-regulation skills to improve programming outcomes may require differentiated instruction based on students self-awareness and existing programming practices.

KEYWORDS

Programming, Metacognition, Self-regulation.

ACM Reference format:
Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J. Ko. 2020. Investigating Novices' In Situ Reflections on Their Programming Process. In *Proceedings of The 51st ACM Technical Symposium on Computer Science Education, Portland, OR, USA, March 11–14, 2020 (SIGCSE '20),* 7 pages.
https://doi.org/10.1145/3328778.3366846

1 INTRODUCTION

Self-regulation—the ability to be aware of one's thoughts and actions, exert control over them, and evaluate how well they are moving one closer towards a goal [17]—is important to successful programming. In an analysis of teaching and learning programming, Sheard et al. highlighted self-regulation as one of a vital set of skills students need to achieve success at programming [19]. Further, prior work has identified that successful learners self-regulate, generating self-explanations of material and use them to monitor for misconceptions [4]. Many studies also have shown that scaffolding aspects of self-regulation benefits novice programmers, further demonstrating the importance of self-regulation for successful programming. For instance, Loksa et al. found that explicitly teaching and scaffolding programming process can improve student productivity, independence, and self-efficacy [12]. Similarly, work by Bielaczyc, Pirolli, and Brown found that training in self-regulation strategies leads to significantly greater programming performance [3]. When it comes to expert software engineers, their systematic and self-reflective methods are a large part of what makes them experts [9] and these self-regulation skills manifest as the deliberate systematic practices which expert programmers and teams use to structure their work [15].

While self-regulation is important for programming, among novices it is infrequent and shallow. Prior work has used retrospective survey data to identify self-regulated learning behaviors such as time management, goal setting, and planning, showing that novices rarely engage in them and when they do they often do so in shallow, unsuccessful ways [6]. Other studies have used interviews, finding that low performing students use few metacognitive or resource management strategies overall [2, 10], and that this is often half of CS1 students. A think-aloud laboratory study on students' self-regulation found that learners may never engage in many self-regulation behaviors at all, and when they do it is often shallow, ineffective, and does not help avoid critical errors [11].

A key limitation of this prior work is that it is all done retrospectively, asking students to reflect on their work *after* it has occurred. Therefore, we know little about how novices self-regulate while they program, and how this self-regulation might differ from what they recall about their self-regulation. There are many reasons to suspect that retrospective data might not generalize to in situ settings. In situ programming happens in a variety of environments (office, classroom, at home), where distractions might be more abundant. It happens in settings that are not time-regulated, often unfolding over many hours or days across multiple contexts. Similarly, programming may include teaching assistants, peers, and the Internet. All of these factors may be difficult to recall retrospectively, missing nuances about self-regulation that learners might engage in, but not remember.

To investigate this generalizability gap in prior work, we studied novice students' programming self-regulation *in situ* across

SIGCSE '20, March 11–14, 2020, Portland, OR, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366846

a 10-week series of four 2-week programming assignments. We specifically investigated the following questions:

RQ1 When prompted to reflect on process in situ, what degree of in situ self-regulation do learners engage in?

RQ2 What challenges do students report encountering when attempting to reflect on their programming process in situ?

2 METHOD

To answer our research questions, we asked learners in a 10-week course to write in journals during programming sessions, reflecting on their problem solving and self-regulation. Using reflective journals is one of a few methods of measuring self-regulation [21] and have been used in CS to enhance programming skills [5]. For this study, journals served as both a prompt to self-regulate during problem solving and a record of the self-regulation the participants engaged in. Therefore, rather than just exposing how students work in the absence of being observed, our data reflects more of a best-case scenario for programming, where there is a scaffolded prompt to think about and write about their programming to support their problem solving.

2.1 Course and participants

We partnered with an instructor of one section of a required front-end web development course programming course in an information science department of a large public research university. Participants consisted of all 31 undergraduate students enrolled in the course, of which all had passed at least one prerequisite programming course covering Java and basic data structures. Of the 31 students, 25 identified as men, 6 as women. One reported being in their 2nd year of undergraduate, 13 in their 3rd year, and 17 in their 4th year.

The course required students to complete 4 projects over 10 weeks. The first project required students to create a personal website using HTML and CSS. The second project required students to create a web-based game written in JavaScript. For this project, students selected the game they wanted to create and were given a variety of suggestions from classic arcade games like Pong or Breakout to casual mobile games like Threes or Bejeweled. The third project of the course required students to create a data explorer using the React framework [8] that allowed a user to interactively explore a data set, such as that exposed by a public web API. What API to use, what data to present, and how to present that data was up to the student, but the project required that the app was both responsive and accessible (perceivable to screen readers). The fourth project for the course required students to create a messaging application using the React framework and a Firebase back-end. This project could be done individually, or in pairs, and required user accounts, authentication, and client-side routing to create a single-page application.

2.2 Data collection

To gather data about in situ self-regulation for RQ1, the instructor required students to submit a programming journal for each of their four projects. To ensure that the students had the language to describe their self-regulation in their journals, the first author taught students definitions of self-regulatory behaviors by giving a 20 minute lecture on the first day of class. This lecture introduced a framework of problem solving that depicts programming as a series of iterative problem solving behaviors drawn from prior work [11]. These behaviors, shown at the top of Table 1, included 1) reinterpreting the problem prompt, 2) searching for analogous solutions, 3) adapting previous solutions, 4) implementation, and 5) the evaluation of implemented solutions. We also described several self-regulation behaviors from this framework, shown at the bottom of Table 1: 1) planning, 2) comprehension monitoring, 3) process monitoring, 4) self-explanation, and 5) reflecting on cognition. (Hereafter, we refer to all behaviors in Table 1 as self-regulation behaviors.) We provided the definitions along with the journaling instructions on the course website for later reference.

Programming Behaviors	Definition
Interpret prompt	Statements about or demonstrating interpreting or questioning the prompt reconsidering actions in reference to the prompt or decomposing the problem into goals requirements and or sub-problems.
Search for analogous problems	Statements about or demonstrating intent to use code they have previously written or use of examples from outside sources.
Adapt a solution	Statements of or demonstrating changing or refining code.
Evaluate	Statements of or demonstrating testing or evaluating outcomes intent to test a solution or identifying why code was not meeting expectations.

Self-Regulation	Definition
Planning	Statements of intended work goals or requirements an intended order of work
Process Monitoring	Statements of start times stop times duration of coding session about work being started identifying work currently in progress when a task is complete or statements that identify actions as part of their process.
Comprehension monitoring	Statements identifying known or unknown concepts or solutions.
Self-explanation	Statements of code explanation for increased understanding.
Reflection	Statements reflecting on prior thoughts of behaviors.
Rationale	Statements that provided rational to decisions or behaviors.

Table 1: Programming and self-regulation behaviors (from prior work [11]) coded in the journals, with definitions.

We instructed students to journal about the start and stop times of each coding session, their progress through the problem solving activities in Table 1, and use the journal as a place to self-regulate in the six different ways described in Table 1. To ensure some consistency in journaling and help students understand the expectations, we provided an example journal that demonstrated how a journal might cover all of the behaviors and demonstrate the level of detail we expected. Because the emotions, struggles, and successes of programming can be very personal, students were assured that their journals would be kept anonymous and we did not enforce a structure nor did we require journals contain specific content. This allowed students to authentically journal about their programming and encouraged including what they valued and expected to be useful. To provide additional scaffolding for reflecting on their process, we provided feedback on the first journal each student submitted, identifying where they could expand on the content, clarity, and depth of their journaling for future journals.

To understand the challenges students' encountered when trying to reflect on their programming process (RQ2), we required students to fill out a survey when submitting each of their assignments and corresponding journals. It asked two open response questions:

- *Your journal is to help you reflect on your process. Review your journal and briefly describe all points where you had trouble reflecting on your process, and/or writing parts of your journal.*
- *Think back to any time in the last two weeks where you stopped and reflected on your programming process when you were not programming or writing your journal and found it difficult. Describe why it was difficult.*

Our primary source of data was the student journals. Despite the journals being part of their grade, some students failed to submit journals. In total, we collected 106 journals with a combined total of 4,227 statements of students reflecting on their programming. Students' journals varied in level of detail, with some being quite extensive, as in this example entry: "*Initially I thought I was going to pseudo code a bunch of stuff, but instead I settled for repurposing a bunch of code from a previous exercise that takes user input and posts messages (Chirper from a previous exercise.)*" Others, in contrast, were quite terse: "*changed how I had BrowserRouter set up.*"

3 RESULTS

3.1 RQ1: What degree of in situ self-regulation do learners engage in?

To answer this question, we performed three analyses:

- We coded and computed the frequency of the behaviors in Table 1.
- We investigated whether there were distinct patterns of student behavior through clustering.
- We analyzed students survey responses about their journaling process for the "maturity" of reflection.

3.1.1 Frequency of behaviors. To understand the frequency of self-reflective behaviors, we developed the coding scheme based on a framework from prior work on programming and self-regulation [11]. We coded each statement of each journal entry to identify if it demonstrated one or more of these behaviors. To ensure the codes we applied were well-defined and consistent, we iteratively refined the code definition by having two authors apply the codes to a set of 430 (10% of the total 4,227 statements) randomly selected journal statements, using adjacent journal statements for context if necessary. To drive refinements, we discussed disagreements, refining definitions, and coded a new set of randomly selected journal statements. After four rounds of iteration, the authors reached 83% agreement on this sample data set. One author then coded all remaining statements. Table 1 shows the final code definitions for each of the self-regulation behaviors. Table 2 shows an excerpt of one students' journal, showing a session of writing, testing, and refining some CSS.

Based on these codes, we computed the frequency of each type of code in each student journal. Table 3 shows the range of the number of codes found in journals for each behavior. The "Total" column in Table 4 lists the total percentage of students who journaled about each behavior at least once across all four journals. These two tables show that most participants (>80%) journaled about the following behaviors at least once across their four journals:

- *Process monitoring* (e.g. "Day Two - Start Time - 1:30 PM | End Time 8 PM")
- *Evaluating solutions* (see Table 2.A,D,F,H for examples)
- *Searching for analogous problems* (see Table 2.E for example)
- *Reflecting on their cognition* (see Table 2.B,I for examples)
- *Self-explanations* (e.g. "*ALL THAT WAS WRONG WAS THAT I DIDN'T LINK IN THE RIGHT VERSION OF JQUERY ASDFGHJKL;[sic]*")
- *Rationale* (e.g. "*I think before I do that I should make the page look prettier and better organized, because I am reusing a lot of code for API requests*")

The least common behaviors included:

- *Interpreting* the prompt (see Table 2.J for example)
- *Adapting* previous solutions (e.g. "*The majority of this chat application will come from exercise sets so I'm taking code from those assignments and picking the components that apply to the project*")

3.1.2 Clusters of behaviors. Prior work suggests that there is large variation in self-regulation behaviors [11]; to better understand this variation, we attempted to cluster students based on which behaviors they did and did not exhibit in their journals. We began by computing a binary variable for each student and each behavior in Table 1 that was true if at least one journal exhibited that behavior, and false otherwise. Then, we performed a visual inspection of this binary data and observed that there were potentially 3 patterns of self-regulation behavior. To verify this interpretation, we applied the K-modes unsupervised clustering algorithm [7] to the student data, using the binary variables as features, specifying K=3 to separate the students into 3 clusters.

The resulting clusters, shown in Table 4, aligned with our visual inspection. One cluster, which we will refer to as the *high coverage* cluster, contained 12 participants whose journals had exhibited at least 9 of 10 of the behaviors in Table 1. These high coverage students typically were missing entries exhibiting the behaviors (*Interpret* the prompt and *Adapting* solutions). Table 2 shows an excerpt from a student in the high coverage cluster evaluating, reflecting, and interpreting about a series of CSS problems. The second cluster, which we will call the *moderate coverage* cluster, had 12 participants who journaled about the most common behaviors (*Process* monitoring, *Evaluating* solutions, *Searching* for analogous problems, *Reflecting* on their cognition, creating *Self-explanations*, *Rationale*), but not the less common behaviors. The final cluster, which we will call the *low coverage* cluster, had 7 participants who

...	...
A - *Evaluate*	Jumbotron inside a container seems too centered, made a manual "container"
B - *Reflection*	Project became really easy after I solidified my idea for the css layout and got it mostly done
C -	Friend suggested bootstrap navbar, that seems a lot easier than doing it with raw css
D - *Evaluate*	Bootstrap navbar was stacking everything on the right
E - *Search*	Took a while of looking through documentation, but apparently that was part of "mobile first" had to define what size it flattens at
F - *Evaluate*	Added pictures from my projects and a picture of my bird, think everything looks good
G -	Was using a stock icon, but I have a personal logo I use in other places, I guess it fits
H - *Evaluate*	Tried setting footer bottom to absolute 0, did not work, will just not mess with it
I - *Reflection*	Decided to add a gradient to make it look nicer, was very surprised how easy it was and how much it improved the look
J - *Interpret*, *Reflection*, *Evaluate*	Was looking through spec, realized I forgot highlights, they didn't work for a bit until I made them really specific

Table 2: The second half of one students' journal for the first homework, showing the codes applied from Table 1. This participant was in the *high* coverage cluster.

Behavior	#1	#2	#3	#4
Interpret	0-1	0-1	0-1	0-1
Search	0-7	0-2	0-2	0-3
Adapt	0-6	0-4	0-1	0-1
Evaluate	0-11	0-6	0-8	0-14
Planning	0-19	0-9	0-19	0-7
Process Monitoring	0-25	0-25	0-54	0-43
Comprehension monitoring	0-6	0-2	0-4	0-2
Self-explanation	0-24	0-17	0-13	0-9
Reflection	0-13	0-7	0-7	0-8
Rationale	0-17	0-8	0-8	0-7

Table 3: The range of student entries exhibiting each self-regulation or programming behaviors, by assignment, showing higher frequencies of evaluation, planning, process monitoring, and self-explanation than other behaviors and interpret only occurring once per assignment.

Code	High (12)	Moderate (12)	Low (7)	Total
Process	100%	100%	100%	100%
Evaluate	100%	100%	100%	100%
Search	100%	100%	86%	97%
Reflection	100%	100%	43%	87%
Explanation	100%	100%	29%	84%
Rationale	92%	100%	29%	81%
Planning	92%	75%	29%	71%
Comprehension	100%	33%	29%	58%
Interpret	75%	8%	0%	32%
Adapt	75%	8%	0%	32%

Table 4: The three clusters of behavior (and size of cluster). Each percentage indicates the proportion of students in the cluster who exhibited the behavior at least once in a journal.

journaled about the fewest behaviors, with *Process* monitoring and *Evaluating* solutions being the only behaviors all participants in this group journaled about.

3.1.3 Maturity of reflection. Whereas our first two analyses considered frequency of reflection and patterns in reflection activity, our third analysis of the journals considered the "maturity" of the reflection itself. We defined maturity as the degree to which self-regulation was an integrated part of students' programming process. To analyze maturity, we qualitatively coded the responses that students gave to the two journaling survey questions, analyzing how students wrote about their journaling process. Two authors inductively coded [20] the responses, identifying varying categories of detail in participant reflections on their process, developed a coding scheme, and used that scheme to independently code the survey questions. The researchers reached 88% agreement before reviewing and reconciling discrepancies, coming to 100% agreement on all assigned codes.

The final coding scheme consisted of three codes representing the level of reflection maturity in participant responses. Participants who demonstrated *mature* reflection were those who identified indicated that they were struggling with reflecting while programming because the reflection conflicted with their process. For example, one participant with *mature* reflection stated, "*When I hit really difficult bugs, I don't want to reflect on them or journal, I just want to look at my code and chase them down.*" These participants demonstrated a high awareness of their current process and how reflection interfered with it. They often stated that they opted not to engage in journaling during programming, choosing to journal after-the-fact to meet the journaling requirement. Another set of participants we identified was those who were actively *integrating* reflection into their process. This group provided indications that they were still developing a programming process, often reporting struggle reflecting due to the task being difficult rather than because it conflicted with any current process. For instance, one *integrating* participant expressed, "*It is [difficult] because that I might not remember all the details all the time.*" The final set of participants we identified as being *process-unaware*. These participants stated that they did not reflect on their process, that they had no difficulties reflecting while providing no additional details, or responded to the question by describing their process or a segment of code that was difficult rather than an aspect of their process. For example, one *process-unaware* participant responded, "*I never really stopped and* *thought about something being hard, I just started looking through google/documentation.*"

To understand the distribution of these different levels of maturity, we assigned participants to one of these categories based on the highest frequency of codes they received each code across all 8 (2 questions per survey, 4 surveys) survey responses. Codes across all responses for each participant were fairly stable, most participants only receiving one or two codes across all 8 responses. In cases of a conflict, we assigned participants to the group with the higher level of reflection maturity. In total 10 participants' were *mature*, 17 participants' were *integrating*, and 2 participants were *process-unaware*.

Finally, we hypothesized that there would be a relationship between students' maturity of self-regulation and the patterns of journaling behavior indicated by our clusters. To test this hypothesis, we preformed a Pearson's chi-squared analysis and did not find a significant association (p=0.2664).

3.2 RQ2: What did students report was challenging about reflecting?

To analyze the challenges that students reported writing their journals, we inductively identified themes in students survey responses. Two authors used an inductive coding approach, independently identifying themes in the open ended survey responses to the questions about what difficulties students found reflecting. The researchers used the themes to define a code book which they used to independently code the entire set of survey responses, and found 100% agreement.

Our results identified three ways that participants struggled to reflect on their programming process. One struggle students reported having while trying to reflect was that the concept of a programming process was too *abstract*. Participants appeared not to be practiced in thinking about their own thinking. For instance, one participant responded, "*It was difficult to think of how I was actually trying to solve things.*" Another challenge that many participants encountered was that it was difficult to *recall* details about their mental work. For example, one participant said, "*Many thoughts that I have about coding come by very quickly, and it's difficult for me to recall small but important influencers that cause me to change how I build something.*" Another challenge we identified was that reflecting on their process actually *conflicted* with their process. As one participant explained, "*It was really difficult to remove myself from my workflow and constantly having to switch between my journal and my code; it broke my workflow and made me work slower.*"

4 LIMITATIONS

As with any empirical study, ours had many limitations. First, all studies of this kind could benefit from more data. Our findings are limited to what we could qualitatively see from a single programming course with 31 students. While appropriate to identify patterns for students in this course, it is not enough data to identify all of the significant patterns of human self-regulation. Another limitation is the type data itself. There exists no way to directly measure cognitive processes. Attempting to expose and understand the mental work of programmers through journals and self-reported reflections is one of a few mechanisms for understanding self-regulation [21]. However, the act of presenting participants with the framework of behaviors and asking them to self-report certainly interacted with their behavior. Also, the robustness of journals as mechanism to observe self-regulation varies due to participants' ability and willingness to express their inner thoughts. Conducting this study in an authentic classroom setting also means our data is entangled with many uncontrolled variables. These include, but are not limited to, programming task properties (e.g. difficulty), student prior experience, time pressures, distractions, amount of guidance received on tasks, and access to a computer.

5 DISCUSSION

Answering RQ1, we found the following: All participants monitored their *Process* and *Evaluated* their solutions. About one third of the participants *only* engaged in those two behaviors, while another third engaged in nearly all of the self-regulation behaviors. Students almost never engaged in *Interpreting* the prompt and *Adapting* previous solutions, and when they did, they typically engaged in all self-regulation behaviors. We also found that students were either *process-unaware*, were struggling with *integrating* reflection into their process, or had a *mature* programming process that conflicted with reflection. Finally, we found that participants struggled reflecting on their process because it was too *abstract* to think about, they had problems *recalling* details about their process, or because reflecting *conflicted* with their current process.

5.1 Interpretation of results

One of the challenges participants reported when reflecting was that the concept of a programming process was too *abstract* for some participants to meaningfully reflect on. This could be because the training we provided, which included definitions and examples for all of the behaviors, was simply inadequate for some students. Another interpretation is that participants where simply not very aware of their process and, thus, had problems identifying and understanding the mental behaviors they were engaging in. Because metacognitive awareness is something that must be developed over time [18], we suspect that it was likely a combination of these two interpretations. We suspect that the challenge of *recalling* detail of their mental work was also due to a lack of experience thinking about their own thinking. The challenge that reflecting *conflicted* with their process, however, we believe the self-reports that participants already had a programming process they have found to be successful which was hindered by reflecting. Together, these challenges suggest that reflecting on programming process may be a method best used for rank novices, before they have begun to develop a process of their own, and that reflection of this sort requires careful training and scaffolded practice.

While we were able to identify and rank behaviors by how frequently students engaged in them, this says little about what this ranking actually means. One interpretation is that the rank reflects the order in which behaviors are developed or relied upon meaningfully to solve programming problems. It could be that students must first develop their skills of *Process* monitoring and *Evaluating* their solutions, which all participants engaged in, in order to gain enough awareness of their other process behaviors. Alternatively, just because a behavior is engaged in does not mean it is done so meaningfully. Prior work on self-regulation identified that the *Planning* and *Comprehension* monitoring behaviors may be the first self-regulation behaviors to help students avoid errors [11]. In our data, however, *Planning* and *Comprehension* monitoring were *not* among the most engaged in behaviors. This could mean that, while these behaviors are undoubtedly important, students often do not rely upon them; they may need to develop other skills first. Another interpretation of the behavior rankings is that it simply highlights the behaviors that are the easiest to be *aware* of engaging in. This would mean that novices are simply not aware of many behaviors, or do not engage in them at all. For instance, few participants reflected on *Interpreting* or *Adapting*. From this we can take that novices simply do not take the time to understand the problem they are attempting to solve and do not, or are not aware of, adapting prior knowledge or examples to help craft a solution. This interpretation aligns with prior work finding that novices often begin to code before understanding the problem [1] and that they have a difficult time leveraging solutions to similar problems to solve their current problem [4, 13].

There are many reasons why the maturity levels (*mature*, *integrating* and *process-unaware*) might not have been associated with the clusters. One explanation is that it was an instrument failure, since the survey questions from which we derived this measure were not originally intended to measure process awareness. However, another explanation is that one's ability to reflect on, and journal about, programming process is highly contingent on a second mechanism; metacognitive *awareness* of process. Following this interpretation, we might expect that it could require little-to-no process awareness to simply monitor mental work and identify when a behavior is being engaged in. A participant who is skilled in monitoring mental work in situ, but lacking a greater awareness of their process might, then, be classified as a *high* coverage participant *and* be *process-unaware*. Conversely, a participant who has *mature* process awareness, even if that awareness concludes that they have a poor process, may lack practice monitoring mental work, or may not be able to expend the mental effort to monitor their process in situ, and thus may exhibit *low* coverage of self-regulation behaviors.

Our data confirms prior work on novice programmers' self-regulation. Prior work conceptualizes developing programming expertise as a series of "levels" [14]. This prior work argues that "level 2" students should value decomposing program goals (which we identify as *Interpreting* the prompt) but that their process is insufficient for larger programs and that their primary focus is getting a program to work. The authors indicate that "Level 3" is when students develop some appreciation for the process of designing a successful program, and thus, more robust process awareness. In the context of this prior work, our participants would be categorized as attempting to move from level 2 (code generators), to level 3 (program generators). This is a fairly appropriate characterization of our participants, who may have only taken one previous programming course and are now learning to develop larger front-end web applications. Our results, too, support this classification. The type and variation of self-regulation and awareness of programming process in our results are appropriate for students transitioning between level 2 and level 3.

Our results also provide further insights into prior work. Liao et al. identified that high preforming CS students use more metacognitive strategies [10]. While this work provides strategy differences between high and low students, such as exam studying and help seeking strategies, our results suggest what self-regulation behaviors might be driving those strategies.

5.2 Implications and future work

Our results have important implications for future self-regulation interventions and research. Our results suggest that educators seeking to scaffold the development of self-regulation skills should strongly consider the robustness of students' current self-regulation skills. The challenges reported by our participants demonstrate that interventions intended to help build self-regulation skills may act to needlessly slow down and hinder students' ability to be productive. Alternatively, students that would fall into the *high coverage* or *moderate coverage* clusters may disregard the intervention in favor of their current workflows resulting in wasted efforts with no benefits. Additionally, without careful training and scaffolded practice *low coverage* students may struggle to begin to develop necessary self-regulation skills at all, remaining in a state of *low coverage*. Instead, educators may want to have tiered, faded scaffolding, systems that first carefully train *low coverage* students in self-regulation skills and the ability to reflect on them, helping them to achieve *moderate awareness*. Additionally, educators might want to take into account the self-regulation behaviors that students are more and less apt to engage in. Some work is already attempting to emphasize the importance of developing the *Interpreting the prompt* from the beginning [16] and similar efforts might be needed for *Adapting* to help student be more aware of their process and help them more quickly achieve *high coverage* levels of self-regulation.

We believe further research into understanding the development of novice programmers' self-regulation is warranted. First, future work should replicate our findings in other authentic programming settings, using other programming languages, and in other cultures. Future work should refine the training and instruments used in our study to more accurately measure self-regulation in situ. Future work should leverage our awareness cluster findings as a basis for further investigations on the development of self-regulation skills in programming. Future work should also explore the order in which novices develop particular self-regulation behaviors. Similarly, future work should investigate any connection between the categories of reflection and the clusters of behavior coverage.

Despite the limitations on the validity and generalizability of our results, our findings are an important step in understanding the in situ self-regulation of novice programmers. With further research, improved instruments, and refined theories, we hope for a future where educators understand self-regulation development and leverage that understanding to support the development of robust self-regulation skills for all students.

6 ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (NSF) under grants 1703304, 1735123, and 12566082. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Carl Martin Allwood. 1986. Novices on the computer: a review of the literature. *International Journal of Man-Machine Studies* 25, 6 (1986), 633–658.

[2] Susan Bergin, Ronan Reilly, and Desmond Traynor. 2005. Examining the role of self-regulated learning on introductory programming performance. In *Proceedings of the first international workshop on Computing education research*. ACM, 81–86.

[3] Katerine Bielaczyc, Peter L Pirolli, and Ann L Brown. 1995. Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and instruction* 13, 2 (1995), 221–252.

[4] Michelene TH Chi, Miriam Bassok, Matthew W Lewis, Peter Reimann, and Robert Glaser. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science* 13, 2 (1989), 145–182.

[5] Ryan Chmiel and Michael C Loui. 2003. *An integrated approach to instruction in debugging computer programs*. Vol. 3. IEEE.

[6] Katrina Falkner, Rebecca Vivian, and Nickolas JG Falkner. 2014. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. ACM, 291–296.

[7] Zhexue Huang. 1997. A fast clustering algorithm to cluster very large categorical data sets in data mining. *DMKD* 3, 8 (1997), 34–39.

[8] Facebook Inc. [n. d.]. React: A JavaScript library for building user interfaces. ([n. d.]). http://reactjs.org/

[9] Paul Luo Li, Amy J Ko, and Jiamin Zhu. 2015. What makes a great software engineer? In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 700–710.

[10] 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 (ITiCSE '19)*. ACM, New York, NY, USA, 196–202. https://doi.org/10.1145/3304221.3319740

[11] 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*. ACM, 83–91.

[12] 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*. ACM, 1449–1461.

[13] Briana B Morrison, Lauren E Margulieux, and Mark Guzdial. 2015. Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research*. ACM, 21–29.

[14] Roy D Pea and D Midian Kurland. 1984. On the cognitive effects of learning computer programming. *New ideas in psychology* 2, 2 (1984), 137–168.

[15] Marian Petre. 2009. Insights from expert software design practice. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 233–242.

[16] 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*. ACM, 531–537.

[17] R Keith Sawyer. 2006. The new science of learning. *The Cambridge handbook of the learning sciences* 1 (2006), 18.

[18] Gregory Schraw. 1998. Promoting general metacognitive awareness. *Instructional science* 26, 1-2 (1998), 113–125.

[19] Judy Sheard, S Simon, Margaret Hamilton, and Jan Lönnberg. 2009. Analysis of research into the teaching and learning of programming. In *Proceedings of the fifth international workshop on Computing education research workshop*. ACM, 93–104.

[20] David R Thomas. 2003. A general inductive approach for qualitative data analysis. (2003).

[21] Barry J Zimmerman. 2008. Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects. *American educational research journal* 45, 1 (2008), 166–183.