In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments

Kai Arakawa
Western Washington University
hicksk5@wwu.edu

Qiang Hao
Western Washington University
qiang.hao@wwu.edu

Tyler Greer
Western Washington University
greert2@wwu.edu

Lu Ding
Eastern Illinois University
lding@eiu.edu

Christopher D. Hundhausen
Washington State University
hundhaus@wsu.edu

Abigayle Peterson
Western Washington University
peter390@wwu.edu

ABSTRACT

Effective self-regulated learning (SRL) is important to student academic success. Understanding what SRL struggles students face in programming assignments is critical to guide many efforts in computing education, such as designing scalable interventions and developing effective learning technologies. Prior studies on this topic contributed to understanding what SRL strategies CS students typically use in programming assignments, and the interventions for some SRL struggles such as procrastination. However, few studies have investigated student SRL struggles in programming systematically. To fill this gap, we investigate student SRL struggles in the context of CS2 through a case study. We used multiple approaches to collect real-time data and validate our findings, such as tracking student progress, identifying potential SRL struggles, and interviewing identified struggling students to confirm our identifications. This study contributes to a deeper understanding of what SRL struggles students face in programming at a fine-grained level, and provides guidance on interventions for SRL struggles.

CCS CONCEPTS

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

KEYWORDS

self-regulated learning, learning behaviors, struggle identification, computing education, automated testing, automated feedback

ACM Reference Format:
Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D. Hundhausen, and Abigayle Peterson. 2021. In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432357

1 INTRODUCTION

Effective self-regulated learning (SRL) is important to student academic success. SRL refers to a process in which students monitor and adapt their learning progress, using strategies related to motivation, cognition, and metacognition, in order to achieve learning goals [1, 2]. Prior studies have shown that effective SRL is positively related to different aspects of learning, such as self-efficacy, academic performance, and persistence [3, 4]. When students conduct effective SRL, that is typically evidenced by their application of SRL strategies to different learning stages, such as careful estimates of the needed time for learning tasks and reflection of the learning progress based on feedback [5–7].

Understanding what SRL struggles computer science (CS) students face in programming is important for computing educators and researchers. Novice students in entry-level programming courses are typically in their first or second college year; most of these students have not fully mastered effective SRL yet [8, 9]. SRL struggles (e.g., time management) has been identified as a main factor that contributes to CS students dropout rate, and particularly for those novice students [10]. Instilling effective SRL strategies and practices can substantially increase the learning efficacy of CS courses [8, 11]. A systematic understanding of what SRL struggles students have in programming can provide a full picture of where students need help the most, and enable effective intervention designs. Additionally, such an understanding can provide practical guidance for devising learning technologies that address these challenges on a large scale [12, 13].

Prior studies on this topic contributed to multiple different aspects of SRL, such as examining the relationship between SRL and student performance in programming courses, and what SRL strategies CS students tend to use [11, 14–19]. In addition, some studies examined interventions for some specific components of SRL, such as procrastination and poor help-seeking behaviors [12, 20]. Despite the contributions of prior studies, few efforts have been made to investigate student struggles of SRL in this context systematically. More importantly, previous studies were limited by a heavy reliance on retrospective data collection methods such as surveys and self-reflection to measure student SRL. Such data can hardly reflect the authentic experience of student programming that unfolds over many hours or days across multiple contexts, missing nuances of SRL which students can not accurately recall or may not even be aware of.

To fill this gap, this study aims to investigate student struggles of SRL in programming assignments in the context of CS2. We used multiple approaches to collect data at the fine-grained level, including tracking student progress of programming, identifying students who had SRL struggles on a real-time basis, reaching out to these students, and interviewing them on what SRL struggles they have on a daily basis. The findings of this study contribute to a deeper and more systematic understanding of student struggles of SRL in programming assignments, and provide guidance for interventions and learning technologies that address these struggles.

2 BACKGROUND

SRL is defined as a process in which students actively monitor and adapt their learning progress, using regulation strategies on behaviors, motivation cognition, and metacognition, in order to achieve their learning goals [2]. Several models have been postulated to categorize SRL process, but most models of SRL are composed of at least four cyclical phases, including task analysis, planning, enactment, and self-evaluation [21, 22]. In the phase of task analysis, students first need to define a task as an easy or a hard task, and analyze the task based on both the difficulty level of the task and individual cognitive factors [23]. In the phase of planning, students set a standard for satisfactory task performance, and set goals and plan actions based on the standard [24]. In the phase of enactment, students perform the planned task, continuously monitor and utilize strategies to regulate their behaviors and emotions, and gather environmental resources (e.g., teacher feedback and help from peers) [23, 25]. In the last phase of self-evaluation, students evaluate their performance and identify the causal attributions that positively or negatively influence their performance and reflect on how to make improvements in subsequent tasks based on feedback [26].

SRL has been studied from different angles in computing education, including the relationship between SRL and CS student academic success, interventions for specific SRL elements, and how CS students at different levels conduct SRL. These studies confirm some findings of studies on SRL in other disciplines, and deepened our understanding of the specific challenges and issues CS students may face. On the one hand, computing education research on this topic largely confirmed the findings on the relationship between SRL and academic success from other disciplines. Bergin *et al.* [18] surveyed 35 students enrolled in a CS1 course, and found that students who used more metacognitive strategies performed better than those who used fewer metacognitive strategies, and higher-performing students generally tend to have a higher level of intrinsic motivation of learning than lower-performing students. Loksa and Ko [11] examined the problem-solving process of 37 students. They found that successful SRL relies on adequate programming knowledge, and students who engage in more planning and comprehension monitoring had overall fewer errors in their code than their counterparts who did not. On the other hand, many studies found that CS students, especially novice learners, need facilitations or help on performing effective SRL [15, 27]. Ko *et al.* [14] found that students had difficulties in regulating their choices of problem-solving strategies, despite knowing which one is more effective, and recommended explicit teaching and learning on SRL to strengthen student capabilities in problem-solving.

A handful of studies investigated what the strategies that students apply when working on their programming assignments are. Pedrosa *et al.* [28] interviewed 38 students in two programming course, and found that the common SRL strategies that students used were information searching, work reviewing, and time management (planning). When students self-monitored that they encountered difficulties in solving a problem, they often searched online for related information or self-evaluated their work for errors (e.g., misspellings, sentence construction, theoretical content). A similar study was conducted with 85 college students taking an introductory software development course [8]. Data was collected through reflection essays, and the authors identified a set of SRL strategies CS students use, such as task difficulty assessment, designing before coding, and problem decomposition. In the same line, Falkner *et al.*[29] further found that expert programmers used substantively more planning (design before coding, time management) and self-evaluation (testing) strategies than novice programmers. Prather *et al.* [30] studied whether providing an explicit metacognitive prompt assisted novice programmers in overcoming the metacognitive difficulties. Marin *et al.* [20] studied using email alert intervention to help students overcome procrastination problems. The efforts to investigate how students perform SRL focused on understanding what SRL strategies students tend to use when working on programming assignments.

Despite the contributions, there are a few limitations and gaps in the literature on this topic. First, the majority of prior studies were conducted retrospectively through surveys or by asking students to reflect on the work they completed a long time afterward [31]. The retrospective data might not generalize to authentic settings, because of missed nuances and the lack of verification of the reliability of this method. Second, although these methods that relied on self-reported data can provide some information about what students did and how they thought, the results are substantially subject to individual interpretations and understanding. When students lack strong self-reflection capabilities, they may not accurately depict their process of SRL, resulting in biased or inaccurate data [32, 33]. Third, few studies focused on investigating what SRL struggles students typically face when programming. A systematic understanding of SRL struggles in programming is critical to provide a full picture of where students need the help the most, and provide guidance for the development of interventions and learning technologies. This study aims to fill the gaps by investigating what SRL struggles students face in programming through multiple data collection approaches, including observing student programming at a fine-grained level, identifying students who are potentially struggling in SRL, and communicating with them through emails and invited interviews. To overcome the scalability challenge of observation in situ, we will utilize the learning technologies such as automated testing and automated feedback, and track student programming activities daily.

3 RESEARCH DESIGN

This study is guided by one research question: *What SRL struggles do CS students have when working on programming assignments?.*

3.1 Experiment Design

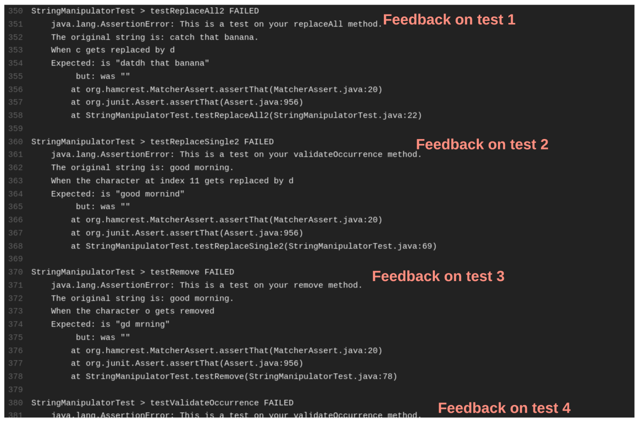
To answer this question, we explore student SRL struggles in programming through a case study. An instrumental case study was a suitable approach to answer this research question because it captures the complexities of a phenomenon at a substantially detailed level that can not be achieved by surveys or controlled experimental designs [34]. Our case study is on a complex programming assignment in the context of CS2. 135 students from a large university in the northwestern United States participated in this case study. The assignment asked students to code a command-line interface that allows users to manipulate any string input in various ways, such as character replacement, insertion, and removal. It takes about 250 to 300 lines of Java code to complete the assignment. The assignment lasted for two weeks; during the two weeks, six lab sessions were arranged for students to work on the assignment with the support and facilitation from two teaching assistants. To strengthen the idea of decoupling, we designed the assignment in a way that required students to code seven independent unit functions and to utilize them in the main function.

To achieve observing student progress towards completing the assignment in situ, we deployed an automated feedback system for this programming assignment using version control and continuous integration tools [i.e., GitHub and Travis-CI(travis-ci.com)]. All students participating in this case study had sufficient knowledge in using version control tools (e.g., Git) through command-line tools before taking this course. They were all encouraged to push their code to GitHub whenever they got a chance to work on the assignment every day. Every push to GitHub would trigger automated testing on Travis-CI. Sequentially, students would be notified of the feedback from the automated testing immediately after they pushed their code to GitHub. **By tracking the log information from the automated feedback system, we were able to observe student coding behaviors in situ on a daily basis, such as lines of changed code, which test cases were passed successfully, and snapshots of student code per push to GitHub**.

To tailor the automated feedback for the programming assignment, we developed unit testing for each of the required functions. Our efforts aimed to (1) provide multiple test cases reaching sufficient test coverage, and (2) provide customized feedback that addresses the gap between the expected and actual output per individual function [6, 13]. An example of what students receive for automated feedback is demonstrated in Figure 1. Students' assignments were graded based on (a) the percentage of test cases that were passed through the automated testing system by the deadline and (b) manual testing with more randomized testing cases on all required functions, including both the unit and main functions. To achieve a reliable result on the manual testing, each assignment was graded by two graders. The inter-rater reliability of grading (Cohen's Kappa) was .96.

3.2 Data collection and analysis

We collected all student performance on the programming assignment, and tracked the data associated with each student's push of code to Github, including the number of lines of changed code, which test cases were passed successfully, and snapshots of student code. To identify the potential struggles of SRL, we set up an alert microservice that sent us a list of potentially struggling students on a daily basis. The microservice identified potential struggles based on two criteria. We identified a student as potentially struggling in SRL when a student:



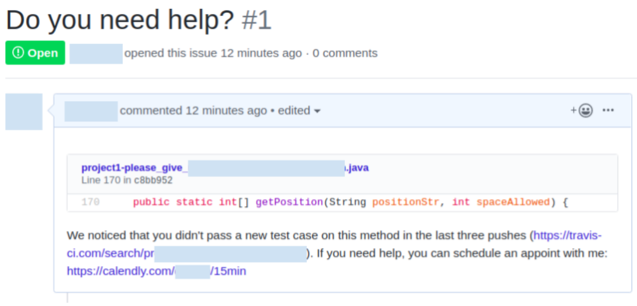
```
StringManipulatorTest > testReplaceAll2 FAILED
    java.lang.AssertionError: This is a test on your replaceAll method.
    The original string is: catch that banana.
    When c gets replaced by d
    Expected: is "datdh that banana"
         but: was ""
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
        at org.junit.Assert.assertThat(Assert.java:956)
        at StringManipulatorTest.testReplaceAll2(StringManipulatorTest.java:22)

StringManipulatorTest > testReplaceSingle2 FAILED
    java.lang.AssertionError: This is a test on your validateOccurrence method.
    The original string is: good morning.
    When the character at index 11 gets replaced by d
    Expected: is "good mornind"
         but: was ""
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
        at org.junit.Assert.assertThat(Assert.java:956)
        at StringManipulatorTest.testReplaceSingle2(StringManipulatorTest.java:69)

StringManipulatorTest > testRemove FAILED
    java.lang.AssertionError: This is a test on your remove method.
    The original string is: good morning.
    When the character o gets removed
    Expected: is "gd mrning"
         but: was ""
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
        at org.junit.Assert.assertThat(Assert.java:956)
        at StringManipulatorTest.testRemove(StringManipulatorTest.java:78)

StringManipulatorTest > testValidateOccurrence FAILED
    java.lang.AssertionError: This is a test on your validateOccurrence method.
```

Figure 1: A partial screenshot of the automated feedback that students received.



Do you need help? #1

Open [redacted] opened this issue 12 minutes ago · 0 comments

[redacted] [redacted] commented 12 minutes ago • edited

project1-please_give_[redacted].java
Line 170 in c8bb952

```
    public static int[] getPosition(String positionStr, int spaceAllowed) {
```

We noticed that you didn't pass a new test case on this method in the last three pushes (https://travis-ci.com/search/pr[redacted]). If you need help, you can schedule an appoint with me: https://calendly.com/[redacted]/15min

Figure 2: An example of how we reached out to potentially struggling students.

- failed to start working on the assignment in time (e.g., not starting before 1/2 of the allowed time passed) or make little efforts in an extended period of time (e.g., no commits in seven days)
- failed to pass the same test case in three sequential commits & pushes to GitHub

To confirm if our "observation" is accurate and what SRL struggles these students have, we reached out to them individually on a daily basis (see Figure 2) to (1) address their knowledge or problem-solving gaps if there are any, (2) confirm if they were struggling, and (3) invite them to have a brief face-to-face interview within two days, focusing on a simple question "*What do you think you were struggling in terms of SRL?*". We documented all the email communications and transcribed all face-to-face interviews.

To answer the proposed research question, we applied grounded theory analysis to the collected data. Different from other qualitative analysis, the grounded theory does not start with a structured coding framework. Instead, the grounded theory involves the development of a coding framework from the data itself [35]. There are two steps in the grounded theory analysis: The first step is open coding, where the data is broken down into distinct segments in

Table 1: Students' SRL struggles in programming assignments.

Struggles of SRL	Freq	%Freq
Task Analysis	18	35%
Misunderstand requirements	7	13%
Lack of fundamental skills	4	8%
Coding before thinking	4	8%
Fail to decompose problems	3	6%
Self-Control	20	38%
Procrastination	12	23%
Underestimate time	8	15%
Self-Reflection	14	27%
Lack of reflections on feedback	10	19%
Fail to seek help	4	8%

An individual student may encounter more than one type of SRL struggles.

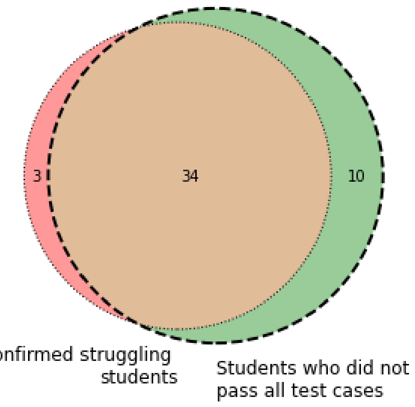


Figure 3: A Venn diagram of failures to pass all the test cases and struggles in SRL.

order to obtain the full collection of concepts in the data, so a coding framework can be generated. The second step is axial coding, where the coding framework is further refined based on theoretical frameworks and comparison within the data. The grounded theory analysis avoids force-fitting observations into rigid categories and misclassification [36].

4 RESULTS

4.1 Coding framework

The development of a coding framework for student SRL struggles in programming is a multi-step process. First, we identified and reached out to 47 students who demonstrated behaviors of struggles by two criteria (see Section 3.3), and confirmed that 37 students were struggling through either email communications or face-to-face meetings. Second, we analyzed all collected data of these 37 students, including both the tracked data and other qualitative data, such as email responses and transcribed interview texts through open coding. Two coders worked independently at this step, either coding the data to existing codes, or creating a new code, identifying a description of the newly created code and examples. The open coding process yielded a total of 23 distinct codes. Third, at the step of the axial coding, the two coders worked collaboratively to iteratively rene the established codes into categories, merging codes where appropriate. Overall, our coding process is informed by the existing SRL framework developed by Zimmerman [37] and CS-specific SRL strategies framework developed by Falkner *et al.* [8]. Our final framework is presented in Table 1.

4.2 Quantitative analysis results

Our analysis reveals that students encountered a range of SRL struggles when working on programming assignments, such as lack of reflections, underestimating the needed time, and failure to decompose problems. Using the grounded theory analysis, we classified these struggles into three main categories, including (1) task analysis, (2) self-control, and (3) self-reflection. Table 1 presents the three categories in our finalized coding framework. Among our quantitative analysis results, we would like to highlight three findings.

First, students who demonstrated SRL struggles tend to have weaker performance than their counterparts. 91.9% of (34 out of 37) confirmed struggling students did not pass all the test cases of this programming assignment. Vice versa, students who have weaker performance in the programming assignment also tend to encounter SRL struggles. 77.2% (34 out of 44) of students who did not pass all the test cases were confirmed as struggling in SRL. This relationship is presented in Figure 3.

Second, more students are identified struggling in SRL as the deadline approaches. The assignment lasted for two weeks, and over 80% of the confirmed struggling students were identified in the second week. As the deadline is approaching, substantially more students were confirmed as struggling. Regardless of the relationship between student performance and struggles in SRL, it is interesting to note that the conventional metrics of automated testing systems are less useful to capture these struggles (see Figure 4). Student average performance based on automated testing showed consistent improvement over time, but failed to capture any SRL struggles.

Third, self-control is the most frequent SRL struggle that students encounter. There are two subcategories of struggles in self-control: procrastination and underestimating the needed time. This is evidenced by (1) failing to participate in the required lab sessions and (2) starting the assignment too late. Among the 37 individual students confirmed as struggling, 11 failed to participate in the required lab sessions. When we applied the coding frame to these 11 students, we found that they also tend to have other types of SRL struggles (see Table 2).

The assignment lasted for two weeks; among the 37 individual students, 19 did not start working on the assignment until the first week passed. When we plot the time when students started working on the assignment against the ratio of passed test cases, it is apparent that students who began late tended to do less well than their counterparts (see Figure 5). To quantify this observation, we applied correlation analysis to the starting time and student

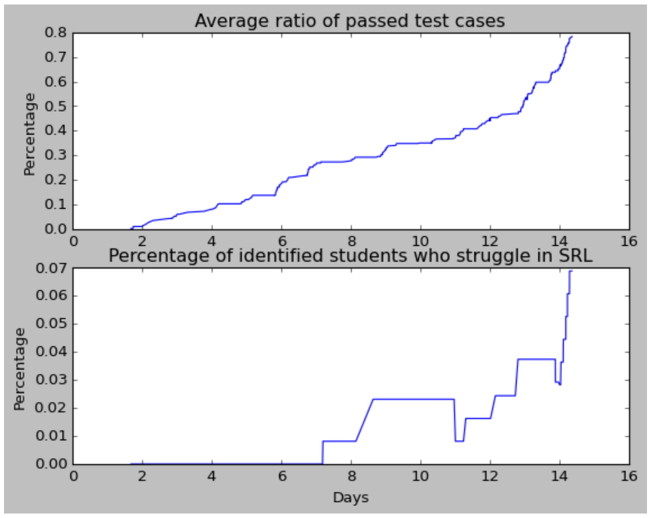


Figure 4: Average passed test cases and confirmed struggling students over time.

Table 2: SRL struggles in programming assignments of students who (1) were confirmed struggling and (2) failed to participate in the required lab sessions

Struggles of SRL	Freq	%Freq
Task Analysis	12	40.0%
Misunderstand requirements	5	16.7%
Lack of fundamental skills	3	10.0%
Coding before thinking	4	13.3%
Fail to decompose problems	0	0.0%
Self-Control	11	36.7%
Procrastination	7	23.3%
Underestimate time	4	13.3%
Self-Reflection	7	23.3%
Lack of reflections on feedback	6	20.0%
Fail to seek help	1	3.3%

An individual student may encounter more than one type of SRL struggles.

performance on this assignment, and found that starting time to be negatively correlated with performance, r(133) = -.38, p < .05. In other words, the later a student begins working on the assignment, the less likely the student is to perform well.

4.3 Qualitative analysis results

Our qualitative analysis of our communications with struggling students indicates that such students lack knowledge about the effective practice of SRL, and may face challenges that are beyond academics. Among all of our findings in qualitative analysis, we would like to highlight three results. First, students struggle in task analysis in various ways that they may not be aware of. A thorough understanding of the task is always necessary for effective programming and problem-solving, yet we found multiple evidence in the lack of understanding in the assignment requirements, such as putting all code under the main function, and making changes to function parameters that were not allowed to be changed. For instance, several students expressed doubts, confusion about coding the required unit functions at one point in time, and expressed regrets of not following the requirements later during the interviews. One student said: "*(Although) I coded everything in the main method, I thought I was being systematic and broke down the problem to pieces that I can tackle. I know what decoupling is ... I just didn't realize why that can be useful until I started debugging my code by the feedback on Travis(-CI).*"

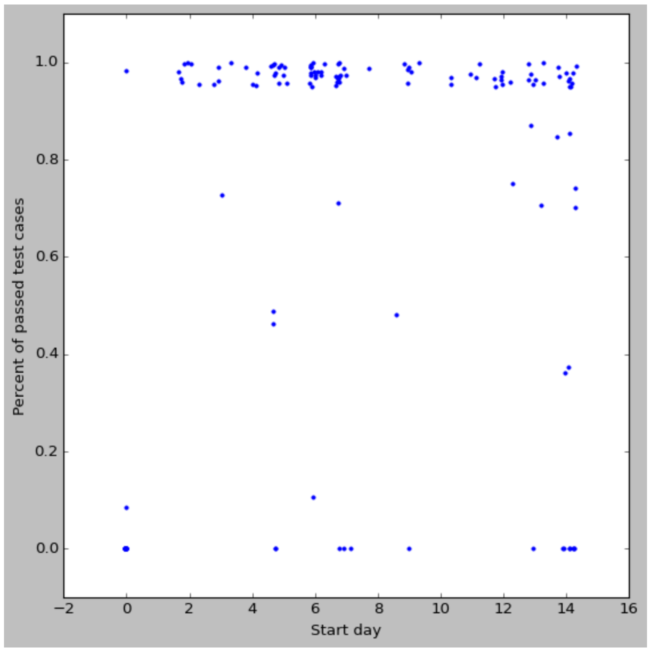


Figure 5: The time when a student started working on the assignment vs. the ratio of passed test cases (*using jittering to avoid overlapping*).

Second, the lack of self-reflection slowed down students' progress substantially in completing the assignment. As is indicated in Section 3.2, students were provided instant automated feedback that breaks down to each of the required unit functions and main function. Multiple students repeatedly told us that they were doing fine despite failing the same test cases in more than three sequential commits & pushes to GitHub, and eventually did not succeed in passing all the test cases. Surprisingly, many of these students did not see the automated feedback as the feedback that they should reflect upon and take advantage of. Instead, they saw it as a reference that is useful only when they need it. For instance, one student said: "*I see the Travis(-CI) feedback as just a reference (that) shouldn't interfere with my progress. (I believe) I have my own thinking, and I will pass all test cases when I finish everything.*" As for not seeking help in time, several students expressed hesitation to admit that they need help or let other people know that they need help. This led them to change a few lines of code, make frequent commits and pushes to GitHub, and check to see if the automated testing results have changed. One student who failed to pass the same test case for five times mentioned: *"I don't know ... I just think all other people experienced the same, so it's not the right time for me to reach out for help yet. Maybe if I try a few more times, I will pass that test case ..."*

Third, struggles in self-control have various contributing factors. In many cases, procrastination and underestimating the needed time are due to academic reasons. Two main reasons are (a) the lack of skills in assessing difficulty and (b) bad learning habits. Several students indicated that they always started their assignments when deadlines were approaching, while some others thought the assignment would take them a day or two, but realized more time was needed when it was too late. In some extreme cases, self-control struggles were actually due to non-academic reasons, such as unexpected heavy study load and the need to balance academic and personal life. For instance, one student had to take five courses for personal reasons and barely could give enough time and effort to any course he/she was taking. For another instance, a student had to take two part-time jobs to support his/her academic life, and had no choice but to miss all the lab sessions where his classmates worked on the assignment with the support from teaching assistants.

5 DISCUSSION

Using in situ data collection technique, we identified a set of SRL struggles CS students tend to have in programming assignments, confirming some previous findings as well as revealing new insights into student SRL struggles. Several previous studies have identified challenges for students to perform effective task analysis. The identified issues range from assessing problem difficulties, assessing the requirements of the given problems, and composing a given problem [8, 15, 27]. Our combinations of observation and interviews confirmed these findings. These findings highlight the need to explicitly teach task analysis skills and reinforce the teaching through purposeful practice in programming courses [11, 18, 30]. Given that students even have problems understanding problem requirements, a simple intervention that requires students to summarize the requirements may force them to put more effort and attention to task analysis, thereby reducing mistakes and struggles at this step [30].

Self-control is the other major SRL struggle students face. Students may put little time and effort into working on the assignment due to underestimating the needed time and bad learning habits [8]. We found that as it got closer towards the deadline, the more likely such struggles will be observed. It may not be the responsibility of CS instructors to cultivate good learning habits in college students, but innovative intervention approaches can undoubtedly be used to help students do a better job in monitoring their progress. Marin *et al.* [20] found that email alert reminding students of lacking progress was hated, but effectively pushed them to put effort into their assignments. Prior educational psychology studies may shed light on how we design and implement effective interventions that strengthen student self-control, such as precommitment and subgoals [19, 38].

Different from some previous studies, we found that many students who had SRL struggles lack the knowledge or skills to perform effective self-reflection, even with the support from automated feedback. When surveys and self-reflection journals are used to collect data only retrospectively, students can only recall what happened, and provide a perspective that is limited by the memory and their own understanding [8, 28]. When examining student self-reflection in programming in situ, Loksa *et al.* [31] found that many students struggled in performing basic self-reflection, and some of them were even not aware of this process. Surprisingly, even as we made the feedback immediately available to students, some of them still chose not to use or largely ignored it. There are two possible explanations for this finding: (1) the lack of adaptivity of automated feedback that makes it difficult to understand, and (2) there is an urgent need to help students recognize the importance of self-reflection, such as how to use feedback and seek help when needed [39, 40].

We believe further research into understanding the SRL struggles novice learners face in programming is warranted. Replication studies are critical to achieving a thorough and systematic understanding of what struggles novice learners tend to have in programming [41]. Future research may consider replicating our data collection methods that combine observation, identification of struggles, and interviews with struggling students in a different context, using a different programming language. Future work may consider comparing and exploring the strengths and weaknesses of the coding frameworks developed by Falkner *et al.* [8] and in our study. Falkner *et al.* [8] used self-reflection journals as the main instrument to measure student SRL while we mainly relied on tracking and observing the data generated from automated testing tools. A comparison like this may further demonstrate the effectiveness of the different approaches to addressing the same problem.

Our results suggest that educators seeking to scaffold the development of self-regulation skills should be more targeted. Students have different SRL struggles to a vastly different extent, and some of these struggles can be inferred based on tracking student learning progress, such as procrastination or failure of self-reflection. The advances in learning technologies, especially automated testing and feedback, provide a valuable opportunity to observe student progress (or lack of) in situ at a fine-grained level. Interventions driven by such data may be more effective in delivering the most needed help to the right individuals, promoting just-in-time learning. Prior studies found that SRL interventions intended to help build SRL skills may needlessly slow down and hinder students' ability to be productive, especially for students with little to no struggles [31]. Future studies may consider integrating data-driven approaches in their design of SRL interventions.

6 CONCLUSION

We investigated student SRL struggles in the context of CS2 through a case study using multiple in situ data collection approaches, such as tracking student progress, identifying struggling students, and confirming the struggles through emails and follow-up interviews. We performed data collection on a daily basis. We identified and ranked common student SRL struggles, including task analysis, self-control, and self-reflection, and revealed findings on each type of struggle through quantitative and qualitative data analysis. This study contributes to a deeper understanding of what SRL struggles students face in programming, and provides guidance on intervention designs for the identified SRL struggles.

REFERENCES

[1] Paul R. Pintrich. A conceptual framework for assessing motivation and self-regulated learning in college students. *Educational Psychology Review*, 16(4): 385–407, 2004. doi: 10.1007/s10648-004-0006-x.

[2] Barry J Zimmerman. From cognitive modeling to self-regulation: A social cognitive career path. *Educational psychologist*, 48(3):135–147, 2013.

[3] Paul R Pintrich and Elisabeth V De Groot. Motivational and self-regulated learning components of classroom academic performance. *Journal of educational psychology*, 82(1):33, 1990.

[4] Maria K DiBenedetto and Barry J Zimmerman. Differences in self-regulatory processes among students studying science: A microanalytic investigation. *The International Journal of Educational and Psychological Assessment*, 5(1):2–24, 2010.

[5] Bernardo Tabuenca, Marco Kalz, Hendrik Drachsler, and Marcus Specht. Time will tell: The role of mobile learning analytics in self-regulated learning. *Computers & Education*, 89:53–74, 2015.

[6] Qiang Hao and Michail Tsikerdekis. How automated feedback is delivered matters: Formative feedback and knowledge transfer. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–6. IEEE, 2019.

[7] Barry J Zimmerman and Dale H Schunk. Reflections on theories of self-regulated learning and academic achievement. In *Self-regulated learning and academic achievement*, pages 282–301. Routledge, 2013.

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

[9] Qiang Hao, Bradley Barnes, and Mengguo Jing. Quantifying the effects of active learning environments: separating physical learning classrooms from pedagogical approaches. *Learning Environments Research*, pages 1–14, 2020.

[10] Ilias O Pappas, Michail N Giannakos, and Letizia Jaccheri. Investigating factors influencing students' intention to dropout computer science studies. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 198–203, 2016.

[11] Dastyni Loksa and Andrew J Ko. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM conference on international computing education research*, pages 83–91, 2016.

[12] Ayaan M Kazerouni, Stephen H Edwards, T Simin Hall, and Clifford A Shaffer. Deveventtracker: Tracking development events to assess incremental development and procrastination. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 104–109, 2017.

[13] Qiang Hao, Jack P Wilson, Camille Ottaway, Naitra Iriumi, Kai Arakawa, and David H Smith. Investigating the essential of meaningful automated formative feedback for programming assignments. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 151–155. IEEE, 2019.

[14] Andrew J Ko, Thomas D LaToza, Stephen Hull, Ellen A Ko, William Kwok, Jane Quichocho, Harshitha Akkaraju, and Rishin Pandit. Teaching explicit programming strategies to adolescents. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 469–475, 2019.

[15] Murali Mani and Quamrul Mazumder. Incorporating metacognition into learning. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 53–58, 2013.

[16] Davin McCall and Michael Kölling. A new look at novice programmer errors. *ACM Transactions on Computing Education (TOCE)*, 19(4):38, 2019.

[17] Christopher Hundhausen, Anukrati Agrawal, Dana Fairbrother, and Michael Trevisan. Integrating pedagogical code reviews into a cs 1 course: an empirical study. In *ACM SIGCSE Bulletin*, volume 41, pages 291–295. ACM, 2009.

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

[19] Qiang Hao, Robert Maribe Branch, and Lucas Jensen. The effect of precommitment on student achievement within a technology-rich project-based learning environment. *TechTrends*, 60(5):442–448, 2016.

[20] Joshua Martin, Stephen H Edwards, and Clfford A Shaffer. The effects of procrastination interventions on programming project success. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 3–11, 2015.

[21] Philip H Winne. Self-regulated learning viewed from models of information processing. *Self-regulated learning and academic achievement: Theoretical perspectives*, 2:153–189, 2001.

[22] Barry J Zimmerman. Attaining self-regulation: A social cognitive perspective. In *Handbook of self-regulation*, pages 13–39. Elsevier, 2000.

[23] Dale H Schunk. Goal setting and self-efficacy during self-regulated learning. *Educational psychologist*, 25(1):71–86, 1990.

[24] Anastasia Efklides. Interactions of metacognition with motivation and affect in self-regulated learning: The masrl model. *Educational psychologist*, 46(1):6–25, 2011.

[25] Qiang Hao, Brad Barnes, Robert Maribe Branch, and Ewan Wright. Predicting computer science students' online help-seeking tendencies. *Knowledge Management & E-Learning: An International Journal*, 9(1):19–32, 2017.

[26] David H Smith IV, Qiang Hao, Vanessa Dennen, Michail Tsikerdekis, Bradly Barnes, Lilu Martin, and Nathan Tresham. Towards understanding online question answer interactions and their effects on student performance in large-scale stem classes. *International Journal of Educational Technology in Higher Education*, 18, 2020.

[27] Anneli Eteläpelto. Metacognition and the expertise of computer program comprehension. *Scandinavian Journal of Educational Research*, 37(3):243–254, 1993.

[28] Daniela Pedrosa, José Cravino, Leonel Morgado, and Carlos Barreira. Self-regulated learning in higher education: strategies adopted by computer programming students when supported by the simprogramming approach. *Production*, 27(SPE), 2017.

[29] Katrina Falkner, Claudia Szabo, Rebecca Vivian, and Nickolas Falkner. Evolution of software development strategies. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 243–252. IEEE, 2015.

[30] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. First things first: providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 531–537, 2019.

[31] Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J Ko. Investigating novices' in situ reflections on their programming process. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 149–155, 2020.

[32] Wayne T Roberts and Philip A Higham. Selecting accurate statements from the cognitive interview using confidence ratings. *Journal of Experimental Psychology: Applied*, 8(1):33, 2002.

[33] Elizabeth J Halcomb and Patricia M Davidson. Is verbatim transcription of interview data always necessary? *Applied nursing research*, 19(1):38–42, 2006.

[34] Pamela Baxter, Susan Jack, et al. Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, 13(4): 544–559, 2008.

[35] Kathy Charmaz. *Constructing grounded theory*. Sage, 2014.

[36] Kathy Charmaz, Liska Belgrave, et al. Qualitative interviewing and grounded theory analysis. *The SAGE handbook of interview research: The complexity of the craft*, 2:347–365, 2012.

[37] Barry J Zimmerman. A social cognitive view of self-regulated academic learning. *Journal of educational psychology*, 81(3):329, 1989.

[38] Carola Grunschel, Malte Schwinger, Ricarda Steinmayr, and Stefan Fries. Effects of using motivational regulation strategies on students' academic procrastination, academic performance, and well-being. *Learning and Individual Differences*, 49: 162–170, 2016.

[39] David J Nicol and Debra Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education*, 31(2):199–218, 2006.

[40] Qiang Hao, Ewan Wright, Brad Barnes, and Robert Maribe Branch. What are the most important predictors of computer science students' online help-seeking behaviors? *Computers in Human Behavior*, 62:467–474, 2016.

[41] Qiang Hao, David H Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J Ko. A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–18, 2019.