Exploring Novices' Struggle and Progress during Programming through Data-Driven Detectors and Think-Aloud Protocols

Benyamin Tabarsi*, Heidi Reichert*, Rachel Qualls†, Thomas Price*, Tiffany Barnes*

*Department of Computer Science
*North Carolina State University
*Raleigh, NC, USA
*{btaghiz,hreiche,twprice,tmbarnes}@ncsu.edu

†Department of Computer Science
†The University of Alabama
†Tuscaloosa, AL, USA
†rlqualls@crimson.ua.edu

2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) | 979-8-3503-2946-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/VL-HCC57772.2023.00029

***Abstract*—Many students struggle when they are first learning to program. Without help, these students can lose confidence and negatively assess their programming ability, which can ultimately lead to dropouts. However, detecting the exact moment of student struggle is still an open question in computing education. In this work, we conducted a think-aloud study with five high-school students to investigate the automatic detection of progressing and struggling moments using a detector algorithm (SPD). SPD classifies student trace logs into moments of struggle and progress based on their similarity to prior students' correct solutions. We explored the extent to which the SPD-identified moments of struggle aligned with expert-identified moments based on novices' verbalized thoughts and programming actions. Our analysis results suggest that SPD can catch students' struggling and progressing moments with a 72.5% F1-score, but room remains for improvement in detecting struggle. Moreover, we conducted an in-depth examination to discover why discrepancies arose between expert-identified and detector-identified struggle moments. We conclude with recommendations for future data-driven struggle detection systems.**

***Index Terms*—CS Education, Struggle, Novice Programming, Detection Systems**

I. Introduction

It is becoming increasingly important to build systems to support students in learning computer science, especially for students from minoritized communities [1], to provide access to high-paying computing-related careers. Block-based programming (BBP) environments like Snap*!* have been designed to provide novices with fun, visual activities that support learning CS, but the nature of programming makes it a challenge for teachers to support all students, especially those who may be struggling, in solving BBP problems in a classroom. Following the definition of Dong et al. [2], we define struggle as the absence of significant progress toward a valid solution. Struggle can sometimes be productive and lead to learning, but it can also be a deeply negative experience for students, affecting their senses of self-efficacy and impacting their intent to persist [3]. When a moment of struggle is successfully detected, it can be addressed with interventions such as iterating between concepts and procedures [4] and general differentiated learning [5]. Additionally, to better understand student learning, it is often helpful to focus on these moments of struggle, but manual identification can be a highly laborious process. Scalable, automated interventions may allow instructors to better allocate limited resources. Several studies have presented new models (e.g., machine learning classifiers) for detecting student struggle during programming based on behavioral indicators [6], [7]. Recently, researchers have developed features that reflect student struggle in programming [8], and automated data-driven struggle and progress detectors, i.e. those that use historical data to build classifiers to detect the occurrence of progress or struggle during an interaction [2].

In this paper, we sought to assess the performance of a state-of-the-art struggle and progress detector (SPD) by Dong et al. [2] with real-time student data and perceptions. SPD classifies student programming trace log data according to whether the code demonstrates sufficient progress on a specific problem in a specific amount of time based on comparisons to prior correct solutions to the same problem. The paper introducing Dong's SPD system evaluated it using expert decisions about whether a human tutor should intervene to help a student while programming, using only the trace log data of student programs. We investigate whether or not the automatic SPD-identified struggle and progress moments correspond to human data during and immediately after the programming process. We triangulate this data using think-aloud protocol (TAP) audio transcripts, video recordings of the programming environment, and SPD detectors applied to programming trace logs. Our two research questions are:

- **RQ1:** To what extent did students' programming actions and verbalized thoughts align with the moments of struggle and progress identified by an automated struggle/progress detector (SPD)?
- **RQ2:** In what scenarios does the SPD fail to recognize students' actual struggle and progress moments?

1943-6106/23/$31.00 ©2023 IEEE
DOI 10.1109/VL-HCC57772.2023.00029

II. Literature Review

Prior research has shown that learning programming is challenging for students [9]. The disproportion between the student population and teaching staff leaves struggling students more prone to challenges that arise [10]. This suggests that improving resources for students and reducing strain on teaching staff could improve students' experiences, and studies have focused on how to improve novice programmers' motivation to program [11].

Modeling important factors in the student programming experience is a prerequisite for detecting students' struggles and providing effective support. Researchers have previously used such data sources as eye trackers, web cameras, programming log data, biometrics, and surveys to determine students' cognitive states while programming, finding that performance is often positively correlated with frustration and negatively correlated with boredom, cognitive load, and memory load [12], [13]. However, less research has been done to understand how to detect these struggling moments and the reasons underlying them during novice students' programming.

A number of studies have focused mainly on students' coding log data to investigate and model important aspects of the student programming experience. Gorson et al. defined coding patterns of novice programmers based on retrospective interviews and qualitative analysis, creating detectors based on these patterns [8]. Their detectors attained F1 scores varying from 66% to 98% in the evaluation phase. In another study to understand novice programming behaviors, Dong et al. defined and classified students' tinkering behaviors in BBP assignments [14].

Others have proposed models using coding logs and assignment and exam scores of students to predict performance [7], with some using sequential pattern mining algorithms and manual inspection to identify code characteristics [15] and predict student success [16]. Dong et al. [2] proposed SPD, which is a set of data-driven detectors that used students' code similarity to historical, correct student solutions in order to find struggling and progressing moments. More specifically, SPD uses "typical time" for achieving "significant progress." The threshold for significant progress in each assignment was calculated using the 25th percentile of absolute progress values of all the student traces in that assignment. The typical time of making significant progress in each assignment was obtained based on the third quartile of the duration of the assignment's all progress chunks. If a student's time to make significant progress is longer than the 75th percentile of all students' significant progress times, SPD would consider that student as struggling. They evaluated SPD using experts' reviews on whether an intervention was needed during a random selection of the periods marked as struggle and progress.

Our work bears two differences from Dong et al.'s original study using an SPD algorithm. First, we detect struggle and progress based on novice student TAP data during programming and investigate how students' actions and verbalized thoughts align with periods flagged by SPD as struggle and progress. Second, we explore the situations where the SPD labels may not match students' TAP data and coding actions.

III. Study Context

A. Population

The participants of this study were 9th and 10th-grade students who attended an in-person introductory programming camp hosted by an American public university in the summer of 2022. This one-week camp was designed for students with no programming background. The camp's goal was to familiarize students with Snap*!*, enabling them to build simple games and collaborate with each other on a programming project. This study was performed on the second and third days of camp. At that point, the students were familiar with all of the relevant programming concepts that were involved in this study's tasks. Participants were required to have both guardian consent and their own assent. Out of twenty-four students who attended the camp, five participated in the study. The demographics of those students are presented in Table I. The demographic section of the survey did not appear for a participant, which is denoted as N/A in the table.

TABLE I: Summary of Students' Demographic Data

Demographic Category		Count
Gender	Female	1
	Male	2
	Prefer not to say	1
	N/A	1
Race	Black or African American	1
	Two or more races	1
	White	1
	Prefer not to say	1
	N/A	1
Ethnicity	Hispanic, Latinx, or of Spanish origin	1
	Not Hispanic, Latinx, or of Spanish origin	3
	N/A	1

B. Study Design

This study's primary data sources were TAP data and programming recordings, in which participants were asked to complete two programming tasks while verbalizing their thoughts. During their programming, students' actions in the programming environment were logged, and their screens and voices were recorded over Zoom.

1) Programming Tasks: Prior to starting the programming tasks, one of the researchers explained to participants how to think aloud. TAP is a method in which researchers ask participants to articulate their thoughts while working on a task [17], allowing researchers to gain insights into participants' thought processes [18]. Following this introduction, participants were offered a non-programming question to practice thinking aloud. They were then presented with the two programming tasks, Squiral and Guessing Game. Squiral is a task where students are supposed to make a square spiral that takes the number of rotations and the side length of the starting line as inputs. Guessing Game is a task where students are supposed to welcome the player and ask them to guess a secret number in a certain range. An advantage of assigning

these tasks is that previous students at the aforementioned university had completed them in different semesters, which created enough data to feed SPD. For each task, participants were allowed to move on after 25 minutes. However, they could choose to extend their time by 5 minutes. With each task prompt, participants were provided with a list of recommended blocks.

2) Programming Log Data Structure: Participants programmed in the iSnap environment [19], which logs students' actions when programming in Snap*!*. This includes coding actions, like creating variables, and non-coding actions, such as running the code. Each of these actions is called a record, and a set of records belonging to a student in a session of programming is called a trace.

IV. Methodology

Prior to looking at the results of the automatic detectors, two researchers watched the recordings of students' programming sessions. While watching the recordings of students' programming screens and listening to their verbalized thoughts, the researchers had the log data of students' programming assignments open. The researchers then tagged periods when they perceived students struggling and progressing. Their tags consisted of a short description of why it was tagged as struggle or progress, along with the ID of the trace log record showing that specific observation. For example, one student's moment of struggle was tagged as "Changing categories while not knowing where the desired block is. ID72", representing that the student was unhelpfully clicking around in search of a block around trace log ID 72. The researchers then collected and compared their tags to each other's. All tags were discussed, following which a single set of complete and agreed-upon tags was produced for each participant's transcript for each of the two assignments. During times of disagreement or ambiguity, the researchers reassessed the students' log data and videos. As the researchers discussed and re-watched recordings and all disagreements were resolved, no similarity coefficients were calculated.

Finally, the researchers examined how the human-identified struggle and progress moments aligned with SPD results. SPD returns dichotomous periods of struggle and progress defined by beginning and ending trace log IDs–for example, a section from 0 to 150 may be labeled as progress. Accordingly, researchers compared the trace log IDs of their own observations against SPD's identified trace log IDs. Specific emphasis was placed on noting where and why moments of disagreement with SPD happened. Also, some SPD-identified periods of struggle or progress included experts' both struggle and progress labels, which made experts add the "mixed" category to the analysis confusion matrices. SPD's confusion matrices, demonstrating the comparison between researcher tags and SPD predictions, are presented in Tables IIIa and IIIb, which are in the appendix.

V. Results

A. RQ1: The Performance of Data-Driven Detectors (SPD)

Tables IIa and IIb show SPD's performance metrics for Squiral and Guessing Game tasks. Based on Table IIa, the precision of SPD is above 75% for both struggle and progress detection, which means SPD's decisions on struggle and progress are reliable for the Squiral assignment. However, the 56% recall of struggle detection indicates that many struggle moments were flagged as progress, which underscores the necessity for SPD's refinement. Overall, the macro-averaged F1-score for Squiral is 76%, which is favorable.[1]

SPD results for Guessing Game progress have the same attributes but recall and precision of struggle detection reveal room for improvement. The current precision suggests that in the 60% of moments flagged as struggle, students were actually struggling, but the recall implies that nearly half of the struggle moments were labeled as progress. The macro-averaged F1-score for Guessing Game is 70%. Hence, SPD's overall F1-score was 72.5%.

TABLE II: Performance Metrics

(a) Squiral

	Precision	Recall	F1-Score
Struggle	0.75	0.56	0.64
Progress	0.81	0.94	0.87

(b) Guessing Game

	Precision	Recall	F1-Score
Struggle	0.6	0.52	0.56
Progress	0.78	0.9	0.84

B. RQ2: Scenarios of SPD Misclassifications

1) Incorrect Struggle:

a) Atypical programming strategies.: Since the notions of progress are rooted in previous students' implementations of problems, certain unique programming strategies that resulted in correct (albeit unorthodox) solutions were treated as a struggle. For example, a participant's (P1) mental model and implementation of the Guessing Game was correct. However, their usage of the broadcast block, which was atypical but not incorrect for this assignment, caused SPD to label their work as "struggle."

b) Slow progress: Due to SPD's data-driven nature, students who attain the same progress over a longer period of time, in spite of their correct mental models and steady progress, may be labeled as struggling. For example, a student's (P2) Guessing Game implementation was considered by both researchers to be full of progress periods, but SPD labeled that entire session as a struggle.

2) Incorrect Progress:

a) Suggested code: In many cases, SPD was confused by the addition of the recommended starting blocks featured in the assignments. As the correct blocks were added to the coding workspace, SPD noted that all students demonstrated

[1]The mixed class is excluded from macro-averaged F1, as it was not a class that SPD predicted.

progress at first. However, students' TAP data, as well as their actions, revealed that they often did not have a strong mental model or strategy for approaching the problem. For instance, P1, immediately after receiving and skimming the prompt for Squiral, began copying recommended blocks. While SPD saw this as a period of progress, the experts recognized P1 was struggling conceptually. Similarly, there is an issue when blocks are placed on the screen but not connected together. This often comes from students tinkering, as seen in P2, but does not indicate conceptual understanding or future progress.

3) Mixed: The mixed category was created when there was some, but not total, disagreement with the results of SPD, predominantly relating to conceptual progress. For example, during Guessing Game, there was a period during which P2's efforts were labeled as a struggle, but the student's actions (re-checking the instructions, slowly exploring available blocks, and noting they "[j]ust needed to put it all together," in reference to their multiple stacks of if statement blocks) suggested an improved understanding of the problem, although their programming did not indicate progress. While programming for Squiral, P2 recognized that they needed to use an editor block, which was more progress than what other students in the study had made, but the period was still labeled as a struggle.

VI. Discussion

A. RQ1: The Performance of Data-Driven Detectors (SPD)

Although our results are not entirely in line with the SPD's original evaluation [2], both analyses indicated the great potential of this type of detector. Still, SPD's recall and precision are lower in struggle detection than in progress detection. Since SPD's initial goal was to find moments of struggle, the number of instances in the progress classes is higher. This imbalance, as shown in Tables IIIa and IIIb, may be a reason why the struggle detection is weaker. Additionally, of six mixed moments, five were labeled as a struggle by SPD, which were mainly due to SPD's time and progress thresholds. In our analysis, we observed that students' minor progress periods sometimes took longer than SPD's threshold, but they were not recognized as they would not meet the minimum progress metric or maximum time for progress. Although these numbers are chosen based on a data set of 104 instances in the original work, they may not work for students of other ages or learning paces. Being adaptive is an important factor to include in the future implementation of struggle detectors.

B. RQ2: Scenarios of SPD Misclassification

Some of SPD's misclassifications were inherent to being data-driven, like being unable to assess the progress of unique programming strategies, but some cases, like students making slow progress, may be remedied. In general, SPD was better at assessing struggle in Guessing Game compared to Squiral, which might be due to the better performance of students in the Guessing Game. In Squiral, most students never moved past the tinkering phase.

C. Design Implications for Data-Driven Detectors

In some cases, the training data used for data-driven detectors (such as SPD) may not align with the actual classroom context for a number of reasons. For instance, our participants were 9th and 10th-grade students, whereas the SPD values for determining progress were based on an introductory programming course for university students. Also, there was a slight difference in the study's suggested blocks for Squiral from the conventional instructions, which made the problem more challenging. These are limitations of data-driven methods that may be alleviated by using fully comparable data and populations. Additionally, detectors may benefit from ignoring blocks that are not connected to block chunks. This would rule out misclassifications related to tinkering or using starter blocks. Finally, we propose a modification in which detectors' progress parameters are tuned based on the individual students. For example, a student's programming rate (as measured by blocks per minute) may reveal whether a student is a fast or slow programmer, which can help personalize the model.

VII. Future Work and Limitations

This study was limited in several respects. First, it utilized an extremely small sample of very novice programmers. To validate results, an assessment of more students' TAP data would be beneficial. Second, due to the study's exploratory nature, inter-rater reliability was not calculated, which will be addressed in future studies. Third, due to space and time limitations and IRB considerations for minors attending a summer camp, participants had to complete their tasks while some other participants were also being studied. Researchers tried to handle this issue by seating participants at a distance from each other. However, overhearing others thinking aloud could happen and might have caused the distraction.

For future research, one avenue is to build hybrid detectors that capture students' struggles and progress in both assignment-specific activities and fully open-ended projects. Additionally, creating a system identifying struggle and progress moments in real-time could help instructors provide targeted, timely interventions.

VIII. Conclusion

In this study, we conducted a small-scale assessment of an existing struggle-progress detection system, which we called SPD. SPD measured students' codes against their similarity with previously identified correct solutions. To evaluate SPD, we asked novice programmers to verbalize their thoughts as they attempted two programming tasks. We then assessed their struggling and progressing moments and compared our results to SPD. We found that, while SPD was generally accurate, there is still room for improvement, particularly in detecting struggle. We ended with a few recommendations for potentially improving data-driven detectors, such as designing assignment-agnostic detectors and differentiating according to a student's demonstrated speed of programming. We intend to use these results to further refine and improve struggle detection and feedback systems for real-time implementation.

REFERENCES

[1] G. Inc., "Developing Careers of the Future: A Study of Student Access to, and Interest in, Computer Science," 2021. [Online]. Available: https://www.gallup.com/analytics/354740/study-of-student-interest-in-computer-science.aspx

[2] Y. Dong, S. Marwan, P. Shabrina, T. Price, and T. Barnes, "Using Student Trace Logs to Determine Meaningful Progress and Struggle during Programming Problem Solving," *International Educational Data Mining Society*, 2021. [Online]. Available: https://eric.ed.gov/?id=ED615663

[3] J. Gorson and E. O'Rourke, "How Do Students Talk About Intelligence? An Investigation of Motivation, Self-efficacy, and Mindsets in Computer Science," in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ser. ICER '19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 21–29. [Online]. Available: https://dl.acm.org/doi/10.1145/3291279.3339413

[4] "Iterating between lessons on concepts and procedures can improve mathematics knowledge - Rittle-Johnson - 2009 - British Journal of Educational Psychology - Wiley Online Library."

[5] C. Cusumano and J. Mueller, "How Differentiated Instruction Helps Struggling Students," *Leadership*, vol. 36, no. 4, pp. 8–10, 2007, publisher: Association of California School Administrators ERIC Number: EJ771718.

[6] A. Estey, H. Keuning, and Y. Coady, "Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 189–194. [Online]. Available: https://dl.acm.org/doi/10.1145/3017680.3017790

[7] J. P. Munson and J. P. Zitovsky, "Models for Early Identification of Struggling Novice Programmers," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 699–704. [Online]. Available: http://doi.org/10.1145/3159450.3159476

[8] J. Gorson, N. LaGrassa, C. H. Hu, E. Lee, A. M. Robinson, and E. O'Rourke, "An Approach for Detecting Student Perceptions of the Programming Experience from Interaction Log Data," in *Artificial Intelligence in Education*, ser. Lecture Notes in Computer Science, I. Roll, D. McNamara, S. Sosnovsky, R. Luckin, and V. Dimitrova, Eds. Cham: Springer International Publishing, 2021, pp. 150–164.

[9] M. Mladenović, I. Boljat, and Žanko, "Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level," *Education and Information Technologies*, vol. 23, no. 4, pp. 1483–1500, Jul. 2018. [Online]. Available: https://doi.org/10.1007/s10639-017-9673-3

[10] S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter, "A Robust Machine Learning Technique to Predict Low-performing Students," *ACM Transactions on Computing Education*, vol. 19, no. 3, pp. 1–19, Sep. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3277569

[11] M. J. Lee and A. J. Ko, "Personifying programming tool feedback improves novice programmers' learning," in *Proceedings of the seventh international workshop on Computing education research*, 2011, pp. 109–116.

[12] Z. Atiq and M. C. Loui, "A Qualitative Study of Emotions Experienced by First-year Engineering Students during Programming Tasks," *ACM Transactions on Computing Education*, vol. 22, no. 3, pp. 32:1–32:26, Jun. 2022. [Online]. Available: https://doi.org/10.1145/3507696

[13] K. Mangaroska, K. Sharma, D. Gašević, and M. Giannakos, "Exploring students' cognitive and affective states during problem solving through multimodal data: Lessons learned from a programming activity," *Journal of Computer Assisted Learning*, vol. 38, no. 1, pp. 40–59, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.12590. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jcal.12590

[14] Y. Dong, S. Marwan, V. Catete, T. Price, and T. Barnes, "Defining Tinkering Behavior in Open-ended Block-based Programming Assignments," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, Feb. 2019, pp. 1204–1210. [Online]. Available: http://doi.org/10.1145/3287324.3287437

[15] M. Kong and L. Pollock, "Semi-Automatically Mining Students' Common Scratch Programming Behaviors," in *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. Koli Finland: ACM, Nov. 2020, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/10.1145/3428029.3428034

[16] G. Gao, S. Marwan, and T. W. Price, "Early Performance Prediction using Interpretable Patterns in Programming Process Data," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. Virtual Event USA: ACM, Mar. 2021, pp. 342–348. [Online]. Available: https://dl.acm.org/doi/10.1145/3408877.3432439

[17] J. Ramey, T. Boren, E. Cuddihy, J. Dumas, Z. Guan, M. J. van den Haak, and M. D. T. De Jong, "Does think aloud work? how do we know?" in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '06. New York, NY, USA: Association for Computing Machinery, Apr. 2006, pp. 45–48. [Online]. Available: https://doi.org/10.1145/1125451.1125464

[18] S. K. Kuttal, A. Sarma, and G. Rothermel, "On the benefits of providing versioning support for end users: An empirical study," *ACM Transactions on Computer-Human Interaction*, vol. 21, no. 2, pp. 9:1–9:43, Feb. 2014. [Online]. Available: https://doi.org/10.1145/2560016

[19] T. W. Price, Y. Dong, and D. Lipovac, "iSnap: Towards Intelligent Tutoring in Novice Programming Environments," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 483–488. [Online]. Available: https://doi.org/10.1145/3017680.3017762

APPENDIX

TABLE III: The Confusion Matrix

(a) Squiral

		Actual		
		Struggle	Progress	Mixed
Predicted	Struggle	9	2	1
	Progress	7	34	1
	Mixed	0	0	0

(b) Guessing Game

		Actual		
		Struggle	Progress	Mixed
Predicted	Struggle	12	4	4
	Progress	11	40	0
	Mixed	0	0	0