How, when, and why do novices struggle in programming? Exploring the experiences and perceptions of common programming moments in block-based environments

Heidi Reichert, Ally Limke, Benyamin T. Tabarsi, Thomas Price, Chris Martens, Tiffany Barnes

North Carolina State University

{hreiche, anlimke, btaghiz, twprice, crmarten, tmbarnes}@ncsu.edu

ABSTRACT

Positive student self-efficacy has been linked to undergraduate computer science students' improved retention rates and success in the major, with self-efficacy in programming being particularly important. To improve poor self-efficacy in programming, especially for novices, we must understand the moments that affect students' self-perceived programming ability. Although these moments have been identified in text-based programming environments, they have not been studied for relevance in block-based programming environments. This study, based on a modified replication of Gorson and O'Rourke's 2020 survey concerning the self-efficacy of introductory computer science students, sought to discover what negative and positive self-assessment moments CS students learning in a block-based environment encountered and how they perceived them. Over the course of two class sessions, students in an introductory, non-major CS course worked on a final project in a pair-programming environment. At the end of each class, students were prompted to take a brief survey about their programming experience that day - how satisfied they were with their progress, how they expected they would do, which struggle and success moments they experienced, and whether they felt these moments happened more frequently to them than to other students in the course. We hypothesized that students who felt that they had challenging moments more often than other students, or whose experiences did not meet their own expectations, would have more negative experiences. Struggle moments across the board were found to be relatively uncommon, with students often rating themselves as struggling and succeeding similarly to their peers. Results suggest that block-based programming environments may help mitigate some of the pitfalls of traditional introductory programming.

Keywords

CS0, block-based programming, novice programming, self-efficacy, computer science education

H. Reichert, A. Limke, B. T. Tabarsi, T. Price, C. Martens and T. Barnes. How, when, and why do novices struggle in programming? Exploring the experiences and perceptions of common programming moments in block-based environments. In B. Akram, T. Price, Y. Shi, P. Brusilovsky and S. Hsiao, editors, Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM) Workshop, pages 66–74, Durham, United Kingdom, July 2022.

1. INTRODUCTION

Computer science is a rapidly growing field; according to the U.S. Bureau of labor statistics, "[e]mployment in computer and information technology occupations is projected to grow 13 percent from 2020 to 2030, faster than the average for all occupations" [13]. Supply has not kept up with demand for CS positions, and there is currently a shortage of skilled workers to fulfill these positions [4]. To satisfy this future need for computer scientists and those with critical thinking skills in the discipline, we must ensure that computer science students at the undergraduate level are retained and supported throughout their educational experiences.

Although many factors, including gender and previous exposure to CS and mathematics, influence intent to join and remain in the major, one of note is a student's self-confidence and perception of their own ability [14]. Self-efficacy, as originally defined by Bandura, is "people's beliefs about their capabilities to produce designated levels of performance that exercise influence over events that affect their lives" [2]. Self-efficacy is influenced by a number of features, but, particularly in a programming context, an individual's perception of both positive and negative moments in relation to themselves is critical. Prior research has shown that self-assessment is connected to self-efficacy and overall motivation [10]. CS students frequently self-assess at lower levels than their actual abilities [5]. Students often interpret normal programming moments as indicative of poor performance; this can result in poor self-efficacy and may drive individuals out of the discipline [7]. This effect can be even stronger for historically marginalized students; for instance, women tend to self-assess at more negative rates than men [9]. Research has found that positive experiences in a student's first CS course are strongly correlated with intentions to remain in the field [3]; therefore, introductory CS is of critical importance in the continuation of the major.

Early intervention to improve student confidence and motivation may improve retention rates and performance. At minimum, improving a student's self-efficacy can improve their overall programming experience and their attitudes toward computer science as a discipline. Given the prevalence of anxiety and depression in CS students, the promotion of positive experiences may be of benefit to students and their

success in the field [11].

While efforts have been made to study the moments that influence self-efficacy in computer science for novice programmers, to our knowledge, this has not been studied in block-based programming environments, through which many novice programmers are taught to program in high school and early college. These efforts have also studied exclusively CS-majors and more advanced learners (i.e. CS1, rather than CS0). Likewise, earlier studies have concerned hypothetical understandings of these self-assessment moments; to our understanding, none have focused on how students perceive their own programming efforts immediately after completing a programming session. Finally, to our knowledge, none of these studies have analyzed how students from historically marginalized communities (e.g. women, students of color) self-assess compared to their peers in block-based programming environments. Previous work has considered block-based programming and self-efficacy. Gunbatar and Karalar demonstrated in 2018 that programming in the block-based platform mBlock improved middle school student self-efficacy perceptions and attitudes toward programming [8]. Although not concerned with improving self-efficacy, Durak, Yilmaz and Yilmaz in 2019 found that, for middle school students who programmed robotic activities, there was a positive relationship between computational thinking, self-efficacy, and reflective thinking [6]. However, this work has not considered specific moments during programming that affect self-efficacy or self-assessment.

Of particular consideration to this paper is the 2020 work of Gorson and O'Rourke, who found 13 specific moments during which students negatively self-assess. These moments, listed in full in Table 2, include forgetting syntax, not understanding simple errors, and making typos while programming. These moments were assessed through surveys administered to three CS1 classes at three midwestern American universities. These surveys asked students to look back on previous experiences with moments rather than assessing students freshly immediately after programming; while these delayed recollections, which have the benefit of memory and reflection, are relevant and useful, they do leave us lacking an understanding of how students may perceive these moments in real-time. Likewise, these surveys were also overwhelmingly negative; most questions had binary answers related to whether a specific moment made somebody "bad at programming," and neither neutral nor positive moments were mentioned. These non-negative statements may be of use in more fully capturing students' senses of self-efficacy.

These moments also focused on text-based programming languages. In these languages, such errors as forgetting syntax or making typos are common; however, in block-based programming languages, these moments may not exist in that form. A student may forget syntax while programming in Python, for instance, but that would be impossible in Snap!; however, a student may forget where a specific block is in the programming environment, which offers a parallel to such a moment.

Thus, this work aimed to use the proposed Gorson and O'Rourke moments and assess their relevance and applicability in block-based programming environments. It is believed that the collection of data on these moments and the development and use of tools to catch when they happen can be used to introduce positive feedback in real-time to struggling students, thus potentially reducing negative self-efficacy by stopping it before it can occur.

2. STUDY DESIGN AND METHODS

This study aimed to answer the following research questions regarding novice (CS0) programmers:

RQ1: *How do a student's perceptions of their experiences, henceforth referred to as "experience", compared against the experiences of other students relate to how positively students perceived their experience?*

RQ2: *Are there differences in how students from historically marginalized communities self-assess their programming abilities compared to students from non-historically-marginalized communities in a block-based programming course?*

RQ3: *What programming events correlate with how students feel about their positive and negative programming experiences in a block-based programming environment?*

The answers to these questions were intended to improve novice student programming, particularly in block-based programming environments. Experience was considered to be an important measure, as positive experiences were considered to be more likely to encourage student retention and attitude toward the field. A survey was developed to answer these questions; unlike previous surveys, this survey was designed to be administered immediately post-programming so as to better capture students' subjective experiences and more realistically understand how these moments occur for students.

2.1 Survey Design

All surveys administered were explicitly based on the work of Gorson and O'Rourke; however, modifications were made to fit the block-based environment and class context.

The original study focused only on moments of struggle, resulting in an overall negatively-leaning survey experience. While negative self-assessment is a critical component of self-efficacy, positive self-assessment is also relevant to how students perceive their own programming abilities. To counterbalance this negative bias for our initial survey, as well as to more accurately and completely capture how students felt post-programming, two moments were included that were not present in the original survey; these two moments concerned how quickly a student finished a task and fixed an error in relation to their initial expectations.

The original survey was also thirteen pages. This allowed for a more in-depth survey, but it was repetitive, with several questions being almost identically-phrased. The longer form was considered too time-consuming for a short, post-programming survey. We chose to first reduce redundancies in questions, as many questions were near-identical repeats of other questions (presumably for the purposes of validation). We also removed vignette-style questions; these were hypothetical scenarios in which students determined whether hypothetical programmers experiencing the listed

Original self-assessment moment	Modified self-assessment moment
Getting a simple error	Getting a simple error, like something not working as desired
Starting over	Starting over or erasing a significant portion of code to try again
Not understanding an error message	The code not working as expected
Stopping programming to plan	Stopping programming to plan
Getting help from others	Getting help from others (e.g. partner or TA)
Spending a long time on a problem	Spending a long time working on a feature
Not knowing how to start	Feeling unsure where/how to start programming
Using resources to look up syntax	Looking up how to do something
Spending time planning at the beginning	Spending time planning before starting to program
Spending a long time looking for a simple error	Spending a long time looking for a simple error or mistake
Struggling to fix errors	Struggling to fix errors and get the program to work
Not able to finish in time expected	Taking longer than expected to finish a feature
Does not understand the problem statement	Feeling unsure about what to work on
(No equivalent)	Finishing a feature quicker than expected
(No equivalent)	Fixing an error faster than expected

Table 1: Self-assessment moments: original and adjusted

programming moments were bad programmers. These cuts reduced the survey-taking experience from approximately 10 minutes (as found by the researchers taking the test) to approximately 3 minutes, which was more acceptable to the course instructor.

The original Gorson study focused on three CS1 classes using text-based programming languages, with a specific emphasis on Python. Three of the 13 struggle moments directly represented problems that were solely present in text-based programming environments; for instance, syntax-based errors and typo-based errors would not usually occur in a block-based environment. These moments were transformed for our study with the goal of capturing the original frustration moment while also being reasonable to find in a block-based environment. Typographical issues were generally replaced, as well as error-specific moments (i.e., receiving an error message); respectively, these were transformed into questions on being unable to find specific blocks and not understanding why a program did not run. Certain moments, such as "Does not understand the problem statement," were changed to better reflect the context of the open-ended assignment (here, this moment was changed to "Feeling unsure about what to work on," as there was no problem statement in our study). The extra context was also provided as appropriate to moments; while the original Gorson and O'Rourke study provided vignettes that offered additional context to what each moment meant, we wanted to reduce bulk while maintaining clarity for participating students. This resulted in the assessed moments having extra context in their names; for instance, this can be seen in the change from "Getting help from others" to "Getting help from others (e.g. partner or TA)." Table 1 lists all of the modifications that were made for each moment.

Students were asked how their overall programming experience/progress went on a Likert scale, in which 1 represented "Very dissatisfied" and 6 represented "Very satisfied." Students were then specifically asked to think about their programming experience from the day and choose from four options assessing each of the 15 moments; the options were "Didn't happen," "I felt bad / frustrated," "It was normal and I felt OK," and "It was good / useful or helped me learn." These options were included to give students more than a binary choice of whether a moment has or has not happened, as well as to give students the opportunity to demonstrate more than simply negative emotions toward a moment; for instance, a student could indicate pride at having overcome an obstacle, or neutrality toward a moment. Likewise, students were asked to note for each self-assessment moment whether they believed other students in their class experienced that moment "Less than me," "The same as me," or "More than me" – in other words, students were asked to compare themselves to their peers.

Demographic Category		Count	Percentage
Gender	Female	7	35%
	Male	11	55%
	Prefer not to say	2	10%
Race	Asian	5	25%
	Black or African American	2	10%
	Mixed	2	10%
	White	9	45%
	Prefer not say	2	10%

Table 2: Demographic information of participants

Students were finally asked for their demographic information. This was done to determine whether the existence of specific programming moments affected a student's perception of programming overall, as well as to determine whether students from historically marginalized communities in computer science experienced moments significantly differently from their non-marginalized peers.

2.2 Survey Administration

The population space was composed of 45 undergraduate students enrolled in an introductory computer science course for non-computer science majors at an American public university. This population was chosen due to its being an introductory class with novice programmers; as students were not computer science majors, most students had little computer science knowledge as the class occurred in the second

(Spring) semester of the 2021-2022 school year. The course utilized a hybrid model, in which students could attend class in-person or online.

We have an approved IRB protocol to survey students about their experience with the course, and only consenting students were included in the study. The initial surveys were administered over the course of two class periods. During each 75-minute class, students worked on their projects in group sizes ranging from one to three. Each project was open-ended and intended as a capstone project to demonstrate knowledge of block-based programming in Snap! After completing this program, students ceased to work in Snap! and began to work in another programming environment.

Ten minutes before each class period ended, the students received a popup within their programming environments with a request to complete the survey. Upon clicking the popup, students were directed to the survey in Qualtrics. In total, 20 consenting students participated over the course of both surveys; of these students, 11 participated in only the first round of surveys, and 9 participated in both rounds of surveys. Surveys from the same student collected over multiple days were used as separate surveys; in total, this resulted in 29 usable surveys to analyze. Demographic information from the 20 students is available in Table 2.

3. ANALYSIS AND RESULTS

3.1 Analysis

The primary dependent variable analyzed during this study was student programming satisfaction; this was, as mentioned under study design, a number from one to six that reflected how a student felt about their programming progress and experience of the day, ranging from very dissatisfied to very satisfied.

Due to the nature of the questions asked, the survey questions were analyzed using different methods. As stated earlier, each student was asked to rank 15 moments on whether each of the moments had happened and, if so, how they felt about each one; students were also asked to share how often they believed each moment happened to their classmates in comparison to themselves. Because of this, two different statistical methods were predominantly used. The experience of moments were split along two lines: whether the moments did or did not happen and, if the moment did happen, whether the moment was negative, neutral, or positive. All moments were designated as either having happened (1) or not happened (0); logistic regression was conducted with moment occurrence as the independent variable and programming satisfaction as the dependent variable. Moments that did happen had their perceived emotions translated to a number (with 1 being negative, 2 being neutral, and 3 being positive), and linear regression was conducted with moment-related emotion as the independent variable and programming satisfaction as the dependent variable.

The occurrence of self-assessment moments, along with the perception of these moments, were correlated against student programming satisfaction. Gender and race also were correlated against student programming satisfaction and each moment's occurrence. Finally, the responses of pairs of individuals, when available, were correlated against each other, along with responses from individuals who completed the survey twice (i.e. on each of the two days in which surveys were administered). Spearman's rank-order correlation was chosen due to our analysis of ranked variables. These correlation coefficients, in conjunction with their p-values, were also used to assess statistical significance.

For all analyses, the typical p-value of 0.05 was used to assess for statistical significance. Corrections were not done for multiple tests.

3.2 Results

3.2.1 RQ1

We had hypothesized that students who perceived themselves as encountering errors *more frequently* than their peers would also have a more negative perception of their programming experience. When assessing student comparison data in relation to experience, no moments of statistical significance were detected. In other words, no meaningful correlation was found between how students perceive other students struggling or succeeding and their feelings regarding their own programming experiences of the day. While initially surprising, there are a few potential explanations for these findings. Most students, as seen in Figures 1 and 2, stated that they believed other students overwhelmingly perceived the same moments as they had; closer inspection of survey results revealed that in nine surveys students exclusively chose "Same as me" to this question for all 15 moments. It is potentially the case that students truly do feel that their peers are performing equivalently to themselves. It is more likely, however, that students were attempting to quickly complete the survey so as to leave class earlier.

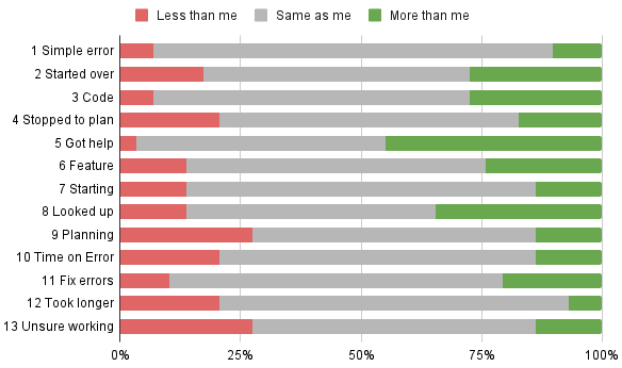

Figure 1: Histogram of self-assessment/comparison of negative moments, and whether students felt others encountered the same moment less than, equal to, or more than them; percentage out of 29 surveys.

3.2.2 RQ2

Gender had some influence on how students self-assessed; non-male students (i.e., students who identified as female or did not provide a gender) were **less likely to feel certain about where/how to start programming** (p-value 0.03 via chi squared test); however, as stated earlier, results were not corrected for, and multiple comparisons were made, and are

Self-assessment moment	Correlation coefficient	p-value
Feeling unsure where/how to start programming	0.4576	0.01256
Spending a long time looking for a simple error or mistake	0.4922	0.00668
Feeling unsure about what to work on	0.4238	0.02194
Finishing a feature quicker than expected	-0.5765	0.00106
Fixing an error faster than expected	-0.5394	0.00253

Table 3: Significant correlations between self-assessment moments and negative programming experiences

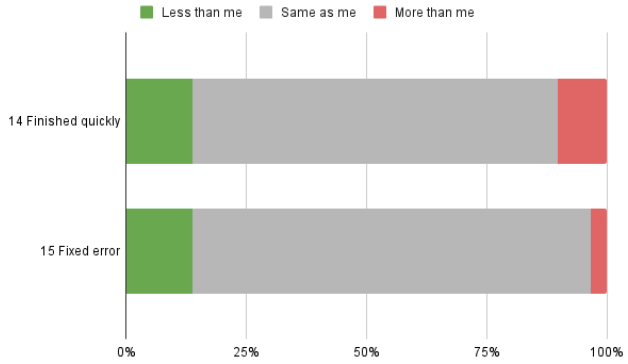

Figure 2: Histogram of self-assessment/comparison of positive moments, and whether students felt others encountered the same moment less than, equal to, or more than them; percentage out of 29 surveys.

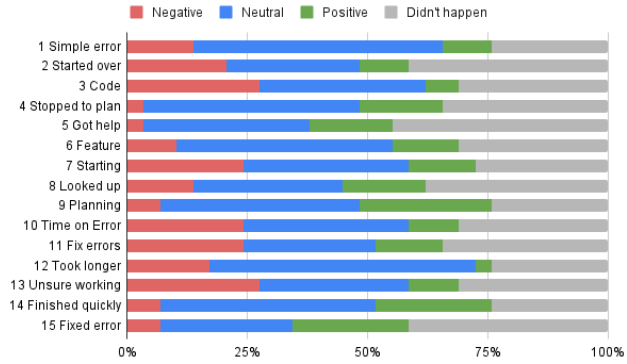

Figure 3: Histogram of each combined moment and students' feelings about them; percentage out of 29 surveys.

consequently most useful for generating hypotheses for evaluation in future work. No statistically significant correlations were found regarding race in any capacity; in other words, a student's reported race had no effect on how students compared themselves to their peers. This is likely due to small sample sizes.

3.2.3 RQ3

Students broadly felt that most moments that they encountered felt normal or okay. The majority of students had neutral or positive programming experiences, as seen in Figure 3. No statistically significant results were found through either the conducted logistic or linear regressions, as seen in tables 3 and 4 located in the appendix. Statistically significant results regarding self-assessment moments were found only during our correlation analysis, and specifically moments that were split between did or did not happen, rather than split among negative, neutral, and positive perceptions. This is likely due to the fact that, while all moments either did or did not happen, only moments that happened could provide meaningful data for linear regression. Correlation coefficients and p-values are located in Table 3.

When students felt unsure of where/how to start programming, they were more likely to have a negative programming experience; this is also the case with spending a long time looking for a simple error or mistake, as well as for feeling unsure about what to work on. When students had finished a feature quicker than expected, as well as when they fixed an error faster than expected, they were more likely to have a more positive programming experience. Two of the negative moments, feeling unsure where/how to start programming and feeling unsure about what to work on, concern uncertainty on the student's end about the task at hand; this may be due to the open-ended nature of the particular programming activity we studied, but may also indicate general uncertainty around beginning a new project. Two of the positive moments offer insights into student expectations; here, students finished a feature quicker and fixed an error faster than expected. Students who came in with neutral expectations and found success then were more likely to feel positive about their programming experiences.

3.2.4 Other Findings

Surveys were also analyzed for correlations between pair-programming partners as well as individual students who had taken the survey twice. These correlations found no statistically meaningful correlations between the responses of pairs of programmers, and only some minor correlations between a single individual's survey responses (i.e., the same individual took the survey twice over the two class sessions). These results emphasize how different students may experience and perceive their programming differently even in a pair-programming dynamic. Likewise, the same student may have a very different experience based on the day. Although two students may have the same project, both are not guaranteed to encounter the same issues. This also suggests a need for more personalized attention to students, rather than having attention be based on projects alone.

Finally, survey moments were correlated against each other. These findings are presented in the appendix in tables 4 and 5; all moments with a correlation coefficient of above 0.4 (p-value > 0.05) are presented, as these indicate some level of statistical significance. These correlations, while certainly not causal, do suggest a level of relationship and perhaps a

"clustering" of moments; the mitigation of one moment may have a significant effect on other moments as well, suggesting that minor changes that affect the programming experience can have far-reaching effects.

4. DISCUSSION

Our results suggest that students in block-based programming environments do experience programming self-assessment moments differently than students in text-based programming environments; while developing our survey, we found that many of the original moments proposed by Gorson and O'Rourke required modifications to be appropriate for the block-based context. Thus, a block-based environment mitigates several of the common moments of struggle students encounter while programming. However, regardless of the environment, we found that students did experience moments of struggle during programming, often relating to confusion on what is meant to be done and what should be done next.

Certain answers were unexpected at first glance; for instance, at least one student rated experiencing receiving a simple error as a positive experience. Although receiving a simple error may be ostensibly an irritating experience, it is potentially the case that a student who has encountered this moment soon overcame the obstacle, consequently feeling satisfaction and associating the moment with positivity rather than negativity. Similarly, several students rated the experience of spending a long time planning as positive or neutral; it may be that a prolonged planning period better prepared students for their programming sessions rather than providing an irritating experience, or that they enjoyed the planning and storyboarding process. It is thus likely the case that these supposed negative self-assessment moments are, in some cases, actually jumping-off points for students to experience positive self-efficacy after overcoming them.

Our analysis suggests that students' programming experiences, at least in this limited setting, are not significantly tied to their perceptions of how other students perform. This may be due to an overwhelming number of students answering that others were performing similarly to them; further study would be required to confirm this hypothesis. However, it may also be that novice programming students do not have a strong basis for comparison; given the class utilized a hybrid approach, it is possible that students were not in positions to consider how their peers would have responded to scenarios. Likewise, students' programming experiences, in general, did not seem to be tied to their identity statuses, with the exception of male students typically feeling more certain of where/how to begin programming. This is positive news, as it indicates that students from historically marginalized backgrounds, at least in this class context, did not generally have worse programming experiences or encounter more moments of struggle than their peers.

Programming moments often occur together or concurrently; changing certain moments may therefore have a direct response on the overall programming experience, both from a positive and negative perspective. For instance, reducing how many students experience code not working as expected would also likely affect how many students experience spending a long time working on a feature. Although not all of these moments are necessarily negative (a student may spend a long time working on a feature but feel immense pride upon completing a task, as suggested earlier), it is generally the case that these moments were associated with more negative or neutral feelings. Alternatively, these moments may be noted to students, with students then gaining the understanding that these moments are common and can be overcome.

Five particular programming moments seem to be especially related to students' overall impressions of their programming experiences; these include three negative ("Feeling unsure where/how to start programming," "Spending a long time looking for a simple error or mistake," and "Feeling unsure about what to work on") and two positive ("Finishing a feature quicker than expected" and "Fixing an error faster than expected") moments, with opportunities for students to minimize discomfort and maximize feelings of success upon navigating these experiences.

5. LIMITATIONS AND FUTURE WORK

This work was severely limited regarding the number of participants. As only 29 survey responses were available for analysis, of which several were from the same individuals on different days, statistical findings are limited. Slight deviations in data may have resulted in large swings in results that may not be present in future analyses. Likewise, for comparison-focused survey questions, most students chose that other students experienced moments exactly the same as them. This may have resulted from survey fatigue, with too many questions having previously been administered; consequently, results that deviated from "Same as me" potentially had undue sway in final analysis results. A larger number of participants may allow for more interesting data mining analysis, which would be a worthwhile future research topic; other work has related to the automatic detection of these self-assessment moments through trace log data [12], and consequent work could use more survey results in conjunction with more programming data to draw more conclusions.

The students in this course were not computer science majors; consequently, no specific results can be drawn regarding retention in the field. A longitudinal study focused on computer science students from their freshmen year on could be more revelatory in that respect.

Not all students worked in similar conditions; while some students were in groups of two, others worked individually or were in groups of three. Consequently, unlike with a homework assignment, work may not have been evenly distributed, and a student's programming experience could have, at least in part, been influenced by the work of their peer programmer(s) as well. The replication of this study on a different assignment would be necessary to test this.

The analysis in this work would be strongly benefited by the introduction of corrections; for instance, the use of Bonferroni correction would help to more thoroughly assess the significance of the results we have concluded.

Although not directly related to this work, it is hypothesized by the researchers that a frank discussion with novice pro-

gramming students regarding these moments would be beneficial for their learning. The work of Gorson and O'Rourke found that many students considered negative self-assessment moments to be indicative of their performance as programmers; consequently, the naming of these moments as both common and normal among all students (and professional programmers) could be highly beneficial to students as they learn to assess their own programming abilities. Future research could more quantitatively assess the value of these discussions.

6. CONCLUSIONS

This work is a preliminary investigation into how students in a block-based programming environment differ in their struggles and potential successes from students in text-based programming environments; some of the first work focusing on students' comparisons against each other in relation to how they affect perceived self-efficacy; and, finally, an exploration into how 15 critical moments during student programming affect both programming experience as well as the experience of other moments for different demographics of students. This study also served to explore the relevance and replicability of the work of Gorson and O'Rourke in a new context; the students in our study were non-majors in CS0, assessed at the end of their learning of Snap! rather than in the middle of their learning of Python. Although replication is highly important to the scientific method, particularly within computer science, it is often neglected [1]; while this work is not a perfect replication, it does serve to help verify previous results for new contexts, which is a contribution.

Overall, we found that the current paradigm for moments of negative self-assessments does not necessarily fit the experiences of block-based programming novices in our studied class and assignment context. We propose the addition of positive moments, as well as the adjustment of moments to better reflect more general aspects of programming experiences. We did find that students often felt these moments, and that they revealed them immediately post-programming. We finally suggest that researchers studying moments of struggle and success for students in block-based programming environments consider developing or adapting their own moments to measure, as there is a clear divide between block-based and text-based environments, and there may be other differences with other languages or programming paradigms.

7. REFERENCES

[1] A. Ahadi, A. Hellas, P. Ihantola, A. Korhonen, and A. Petersen. Replication in computing education research: researcher attitudes and experiences. In *Proceedings of the 16th Koli calling international conference on computing education research*, pages 2–11, 2016.

[2] A. Bandura and S. Wessels. *Self-efficacy*, volume 4. na, 1994.

[3] S. Beyer. Why are women underrepresented in computer science? gender differences in stereotypes, self-efficacy, values, and interests and predictors of future cs course-taking and grades. *Computer Science Education*, 24(2-3):153–192, 2014.

[4] D. D. Bowman. Declining talent in computer related careers. *Journal of Academic Administration in Higher Education*, 14(1):1–4, 2018.

[5] M. A. Collura and S. B. Daniels. How accurate is students' self assessment of computer skills? 2008.

[6] H. Y. Durak, F. G. K. Yilmaz, and R. Yilmaz. Computational thinking, programming self-efficacy, problem solving and experiences in the programming process conducted with robotic activities. *Contemporary Educational Technology*, 10(2):173–197, 2019.

[7] J. Gorson and E. O'Rourke. Why do cs1 students think they're bad at programming? investigating self-efficacy and self-assessments at three universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 170–181, 2020.

[8] M. S. Gunbatar and H. Karalar. Gender differences in middle school students' attitudes and self-efficacy perceptions towards mblock programming. *European Journal of Educational Research*, 7(4):925–933, 2018.

[9] I. T. Miura. The relationship of computer self-efficacy expectations to computer interest and course enrollment in college. *Sex roles*, 16(5):303–311, 1987.

[10] E. Panadero, A. Jonsson, and J. Botella. Effects of self-assessment on self-regulated learning and self-efficacy: Four meta-analyses. *Educational Research Review*, 22:74–98, 2017.

[11] L. M. Soares Passos, C. Murphy, R. Zhen Chen, M. Gonçalves de Santana, and G. Soares Passos. *The Prevalence of Anxiety and Depression Symptoms among Brazilian Computer Science Students*, page 316–322. Association for Computing Machinery, New York, NY, USA, 2020.

[12] B. T. Tabarsi, A. Limke, H. Reichert, R. Qualls, T. Price, and T. Barnes. How to catch novice programmers' struggles: Detecting moments of struggle in open-ended block-based programming projects using trace log data. *Joint Proceedings of the Workshops at the International Conference on Educational Data Mining*.

[13] U.S. Bureau of Labor Statistics. Computer and information technology occupations, Apr 2022.

[14] B. C. Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. *SIGCSE Bull.*, 33(1):184–188, feb 2001.

APPENDIX

	Got a simple error like something not working as desired
The code was not working as expected	0.66669372
Spent a long time working on a feature	0.49251248
Struggled to fix errors and get the program to work	0.43845407
Took longer than expected to finish a feature	0.62337662
	Started over or erased a significant portion of code to try again
Got help from others, e.g. partner or TA	0.50973276
Spent a long time working on a feature	0.4957609
Struggled to fix errors and get the program to work	0.56888717
Took longer than expected to finish a feature	0.50777003
	The code was not working as expected
Got a simple error, like something not working as desired	0.66669372
Started over or erased a significant portion of code to try again	0.4957609
Spent a long time working on a feature	0.83888889
Felt unsure where/how to start programming	0.41978988
Spent a long time looking for a simple error or mistake	0.67777778
Struggled to fix errors and get the program to work	0.76784806
Took longer than expected to finish a feature	0.66669372
Spent time planning before starting to program	0.60798964
Felt unsure about what to work on	0.45421998
	Got help from others, e.g. partner or TA
Started over or erased a significant portion of code to try again	0.509732762
	Spent a long time working on a feature
Got a simple error, like something not working as desired	0.49251248
Started over or erased a significant portion of code to try again	0.4957609
The code was not working as expected	0.83888889
Felt unsure where/how to start programming	0.58655573
Spent a long time looking for a simple error or mistake	0.67777778
Struggled to fix errors and get the program to work	0.76784806
Took longer than expected to finish a feature	0.66669372
The code was not working as expected	0.41978988
Spent a long time working on a feature	0.58655573
Spent a long time looking for a simple error or mistake	0.58655573
Struggled to fix errors and get the program to work	0.52613405
Felt unsure about what to work on	0.75332157
	Spent time planning before starting to program
Stopped programming to plan	0.60798964
Finished a feature quicker than expected	0.43506494

Table 4: Correlation coefficients of moments, p-value > 0.05

	Spent a long time looking for a simple error or mistake
The code was not working as expected	0.67777778
Spent a long time working on a feature	0.67777778
Felt unsure where/how to start programming	0.58655573
Struggled to fix errors and get the program to work	0.76784806
Took longer than expected to finish a feature	0.66669372
Felt unsure about what to work on	0.51666667
	Struggled to fix errors and get the program to work
Got a simple error like something not working as desired	0.43845407
Started over or erased a significant portion of code to try again	0.56888717
The code was not working as expected	0.76784806
Spent a long time working on a feature	0.76784806
Felt unsure where how to start programming	0.52613405
Spent a long time looking for a simple error or mistake	0.76784806
Took longer than expected to finish a feature	0.60798964
	Took longer than expected to finish a feature
Got a simple error, like something not working as desired	0.62337662
Started over or erased a significant portion of code to try again	0.50777003
The code was not working as expected	0.66669372
Spent a long time working on a feature	0.66669372
Spent a long time looking for a simple error or mistake	0.66669372
Struggled to fix errors and get the program to work	0.60798964
	Felt unsure about what to work on
Stopped programming to plan	0.454219979
Felt unsure where/how to start programming	0.753321569
Spent a long time looking for a simple error or mistake	0.516666667
	Finished a feature quicker than expected
Spent time planning before starting to program	0.43506494
Fixed an error faster than expected	0.67138482
	Fixed an error faster than expected
Finished a feature quicker than expected	0.67138482

Table 5: Correlation coefficients of moments, p-value > 0.05, cont.