Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #47141

An analysis on the effectiveness of randomized, auto-graded activities in introductory programming courses

Jamie Emily Loeber, zyBooks, A Wiley Brand

Jamie Loeber is an Assessment Specialist at zyBooks, a Wiley Brand. She earned her B.S. in Computer Science at the University of California, Irvine. She has taught programing and machine learning to students across the globe. Jamie is passionate about improving computer science education and creating better learning experiences in STEM.

Ms. Efthymia Kazakou, zyBooks, A Wiley Brand

Efthymia Kazakou is Sr. Assessments manager at zyBooks, a startup spun-off from UC Riverside and acquired by Wiley. zyBooks develops interactive, web-native learning materials for STEM courses. Efthymia oversees the development and maintenance of all zyBo

Dr. Yamuna Rajasekhar, zyBooks, A Wiley Brand

Yamuna Rajasekhar is Director of Content, Authoring, and Research at zyBooks, a Wiley Brand. She leads content development for the Computer Science and IT disciplines at zyBooks. She leads the authoring and pedagogy team at zyBooks, developing innovative learning solutions that drive measurable student success. She is also an author and contributor to various zyBooks titles. She was formerly an assistant professor of Electrical and Computer Engineering at Miami University. She received her M.S. and Ph.D. in Electrical and Computer Engineering from UNC Charlotte.

Dr. Annie Hui, zyBooks, A Wiley Brand

Annie Hui is a zyBooks assessment specialist. She has 15 years of experience teaching computer science, information technology, and data science courses, in both in-person and online modes. She has taught in Northern Virginia Community College and George Mason University. She specializes on course design to maximize student engagement and success.

Nicole Kehaulani Collins, zyBooks, A Wiley Brand

Nicole Collins is an Author Trainer and former Assessment Specialist at zyBooks, a Wiley Brand. She earned her B.S. in Computer Science and her M.Ed. in Learning, Design & Technology from UNC Charlotte. Her professional interests include computing education, online learning, educational technology, instructional design, curriculum development, and DEI in STEM. Nicole is passionate about creating engaging and effective learning experiences for students, leveraging her expertise in instructional design and technology to enhance educational outcomes for STEM disciplines.

An analysis on the effectiveness of randomized, auto-graded activities in introductory programming courses

Abstract

Introductory programming courses are often overwhelming to students who have no prior coding experience. The design of an introductory programming course (CS1) can influence a student's choice of pursuing computer science in their career. Research shows that breaking down programming concepts to simpler and smaller pieces decreases the cognitive load and struggle for students thereby increasing student interest and retention.

This paper analyzes data from four online, interactive introductory programming textbooks with activities that use a scaffolded approach to teach students programming concepts. The textbooks used are Programming in C++, in Java, in Python, and in C, respectively. The textbooks cover a variety of programming concepts from basic topics such as variables and branches to more advanced concepts such as recursion, across multiple languages. The textbooks have a series of randomized, auto-graded activities presented in a multi-level format, where each level is progressively harder than the previous. The activities are designed as code reading or code writing exercises to assess students' mastery to read and write code.

We analyzed over 726 such activities with 2179 levels attempted by 107,825 students across 572 universities. We study the average completion rate, the average time spent on an activity, and the average number of attempts per problem level. We further examined the study trends for how often students repeated completed activities for practice. The data validates the effectiveness of a scaffolding approach, and shows the effectiveness of randomized, auto-graded activities to teach programming concepts.

Introduction

The design of a CS1 course plays a crucial role in shaping a student's decision to pursue a career or degree in computer science. Research has shown that the way programming concepts are introduced and taught can significantly impact a student's interest and success in the field, and effective course design can help engage beginners and foster their interest in the subject [1]. In the following three sections, we discuss key factors from the literature that influence student motivation and course success. We also outline the motivation behind this study and define the concept of "effectiveness" as it will be used throughout the paper to validate our findings.

Scaffolding

An important characteristic of an effective learning activity includes an appropriate level of difficulty that builds upon prior knowledge through scaffolding. Scaffolding is an instructional approach that involves breaking down learning tasks into smaller, more manageable pieces and providing support at each step. In the context of introductory programming courses, scaffolding helps students build their skills incrementally by gradually increasing the complexity of programming tasks. Scaffolded activities present problems in a step-by-step manner, where each step builds upon the previous one. Research indicates that this method is highly effective in

designing homework assignments, as it helps students retain concepts more effectively [2], [3], [4]. By significantly reducing the mental effort required to process information (known as "cognitive load") [5], scaffolding increases student interest and learning potential. This approach not only helps students understand complex programming concepts but also boosts confidence and efficacy [6].

Feedback

In addition to scaffolding, an effective learning activity incorporates timely and constructive feedback that is both immediate and clear. Timely feedback allows students to address mistakes while the material is still fresh in their minds, maximizing the opportunity for learning. Constructive feedback provides step-by-step explanations of the expected answers, using clear language to guide students in identifying the specific areas of their mistakes. This approach encourages students to discover and correct errors on their own without revealing the solution outright. Furthermore, such feedback helps address student confusion, misconceptions, and recurring gaps by clarifying difficult concepts and providing targeted guidance where needed most [7], [8]. Together, these elements reinforce understanding, boost students' confidence, and ensure a deeper grasp of the material.

Accessibility

Developed by the Center for Universal Design, the general principles of universal design [9] emphasize creating environments that are accessible and inclusive for all. These principles serve as a foundation for ensuring that learning activities are effective and equitable. Accessibility is crucial for addressing the diverse needs of students, including varying levels of knowledge, abilities, and learning preferences. Simple and intuitive designs, interactive elements, hands-on exercises, and clear instructions should be employed to enable meaningful engagement for all learners [10]. Additionally, features such as alternative text for visuals, closed captions for audio content, and user-friendly interfaces promote equity by accommodating students with disabilities. By adhering to the vision of universal design, learning activities can foster an inclusive environment where every student has the opportunity to succeed [11].

Motivation

This paper addresses the challenge of making introductory programming courses more effective for beginners by focusing on the effectiveness of a scaffolding approach with randomized, auto-graded activities. The objective is to analyze the impact of this approach on student performance and engagement. By comparing data from both randomized and scaffolded activities as well as non-randomized and non-scaffolded activities across four programming books (C++, Java, Python, and C), we aim to provide insights into students' study habits and trends. Specifically, we will examine how often students repeat completed activities for practice, the average completion rate, the time spent on activities, and the number of attempts per problem level.

We define *effectiveness* as the extent to which a learning activity achieves its intended purpose, aligns with classroom learning objectives, fosters student engagement, and promotes mastery of

concepts. This mastery leads to growth and enables students to apply gained skills in a variety of ways. We acknowledge that later chapters in the course are inherently more challenging, as they build upon previously introduced concepts and require students to synthesize and apply their accumulated knowledge in increasingly complex ways. Despite this natural progression in difficulty, our activities in these later chapters are designed to remain effective by reinforcing foundational knowledge, maintaining student engagement, and providing targeted support to ensure continued mastery of the material.

Challenge Activities

A Challenge Activity (CA) is a mastery-based assessment, and is our online textbook's version of a homework problem [12]. We define three types of CAs included in our online interactive programming books: *CodeOutput*, *CodeWriting*, and *CodeWarmup*. All CA types assess a student's mastery of programming by introducing code snippets that students have to either write code, or read code to identify the output. Further information for each activity type is provided in the following sections or in previous work [12], and a comparison of the CA types is shown in Table 1.

Table 1. Comparison of CA types.

CA Type	Minor error notification (whitespace, newline)	Explanation	Randomization	Scaffolded levels
CodeWriting	✔	✔	✔	✔
CodeOutput	✔	✔	✔	✔
CodeWarmup	✔	✔	✖	✖

CodeOutput

CodeOutput CAs (or "code analysis" CAs [12]) assess a student's mastery in programming by providing students with code that students must read and analyze to provide the code's text output as their solution.

CodeWriting CAs

CodeWriting CAs assess a student's mastery in programming by providing incomplete code that a student must complete given the activity's prompt. The prompt shows one or more examples of the expected output/result depending on the different cases examined so that the expected behavior and output of the code is clear to the student. CodeWriting CAs are highly randomized, meaning that the activities support three different forms of randomization: meaningful, cosmetic, and test case [12].

CodeWarmup CAs

Similar to CodeWriting CAs, CodeWarmup CAs assess a student's mastery in programming by providing incomplete code that a student must complete given the activity's prompt; however, CodeWarmup CAs only have one level and do not contain randomization. These CAs are designed to introduce students to simple problems initially, before progressing to more complex topics that require the mastery of multiple learning objectives.

Methods and Metrics

This study follows an observational design, as it evaluates student performance, engagement, and retry behaviors in CAs across four languages, C++, Java, Python, and C. The analyzed metrics include CA characteristics such as type, number of CAs, levels, and the chapter and section associated with each CA, as well as student behaviors including the number of students, scores, and reattempts before and after completion. The metrics were collected through Mode Analytics, which stores information from the platform hosting the CAs. Using SQL and Python, the data is extracted and organized to identify trends and patterns in student behaviors.

To evaluate the effectiveness of our learning activities, as defined by their ability to achieve intended purposes, align with classroom learning objectives, foster student engagement, and promote mastery of concepts, we analyzed a range of key metrics. Specifically, we examined students' performance and behavior, the scaffolding of CAs, and students' effort. These metrics were chosen to provide an assessment of how well our CAs supported student growth, facilitated mastery, and enabled the application of skills in progressively complex contexts, aligning with our definition of effectiveness.

Student performance

Student performance was categorized into four distinct groups based on performance exhibited by the students. The definitions for each group are as follows:

1. **Perfect score students**: Students who attempted at least 40 CAs and received a perfect score on all CAs attempted.
2. **Imperfect score students NOT struggling with later topics:** Students who attempted at least 40 CAs, did not receive a perfect score on all the CAs, and had a correlation of ≤ 0.05 between their decreasing scores and CA positions.
3. **Imperfect score students struggling more with later topics:** Students who attempted at least 40 CAs, did not receive a perfect score on all the CAs, and had a correlation of >0.05 between their decreasing scores and CA positions.
4. **Low participation students:** Students who attempted fewer than 40 CAs, and were excluded from further correlation analysis.

Student behavior

Student behavior was categorized into six distinct groups based on their behavior with CAs. The definitions for each group are as follows:

1. **Once-and-done:** Students who passed the CA on their first attempt and did not retry the CA after passing.
2. **Refiners:** Students who passed the CA on their first attempt but chose to retry the activity at least once to improve their understanding.
3. **Grade motivated:** Students who failed at least once but passed without retrying after their first success.
4. **Earnest learners:** Students who failed at least once but demonstrated persistence by retrying the activity after passing.
5. **Earnest strugglers:** Students who made at least three failed attempts but ultimately gave up without passing the activity.
6. **Lowly motivated:** Students who only made one failed attempt then gave up without further effort.

Scaffolding

Scaffolding was evaluated by looking into the number of CA levels for each CA type across all four programming languages. This was further analyzed by examining the correlation between the number of levels and the corresponding passing rates for each CA in order to assess how progressively layered concepts influenced student performance.

Student effort

Student effort was categorized into two primary groups based on their level of engagement:

1. **Highly proactive group:** Students who attempted at least 10 CAs of each type, thus attempting at least 30 CAs.
2. **Semi-inactive group**: Students who made at most 10 attempts on each type of CA, thus attempting at most 30 CAs.

Results

Demographics

A total number of 726 unique CAs across the four languages was analyzed, and 103,750 unique students attempted at least one CA over 572 unique institutions. The distribution of unique institutions categorized by the institution type (i.e. 4-year, 2-year, high school, and other) is shown in Table 2. To maintain data anonymity, further identifying details about the institutions and student population were not available for this study.

Table 2. Distribution of unique institution types.

Institution type	Four-year	Two-year	High school	Other	Total
Number of institutions	391	151	7	23	572

Data was collected and studied for all four languages. Our presentation focuses on C++ to highlight the patterns we observed when the results are homogeneous across languages. Only in cases where the data show language-specific differences are all four languages compared together.

The C++ book contains a total of 222 CAs available to instructors. The average number of CAs assigned by instructors to their students is 99. Students attempted an average of 77 CAs in a book they subscribed to. These numbers are comparable across all four languages. The numbers of student enrollment vary across different languages as shown in Table 3.

Table 3. Distribution of student enrollment per language.

Language	C++	Java	Python	C
Student enrollment	18,985	39,484	38,754	6,541

Student Performance

When analyzing student performance, we examined the correlation between imperfect student scores and the relative position of the corresponding CA. In all languages:

- **Imperfect score students NOT struggling with later topics** had a ≤0.05 correlation between students' decreasing scores and CA positions, suggesting that these students did not have a harder time with later CAs.
- **Imperfect score students struggling more with later topics** had a >0.05 correlation between students' decreasing scores and CA positions, suggesting that these students struggled more with later CAs.

The distribution of students in each category of performance is similar across all languages. Figure 1 provides a visualization of the data for C++.

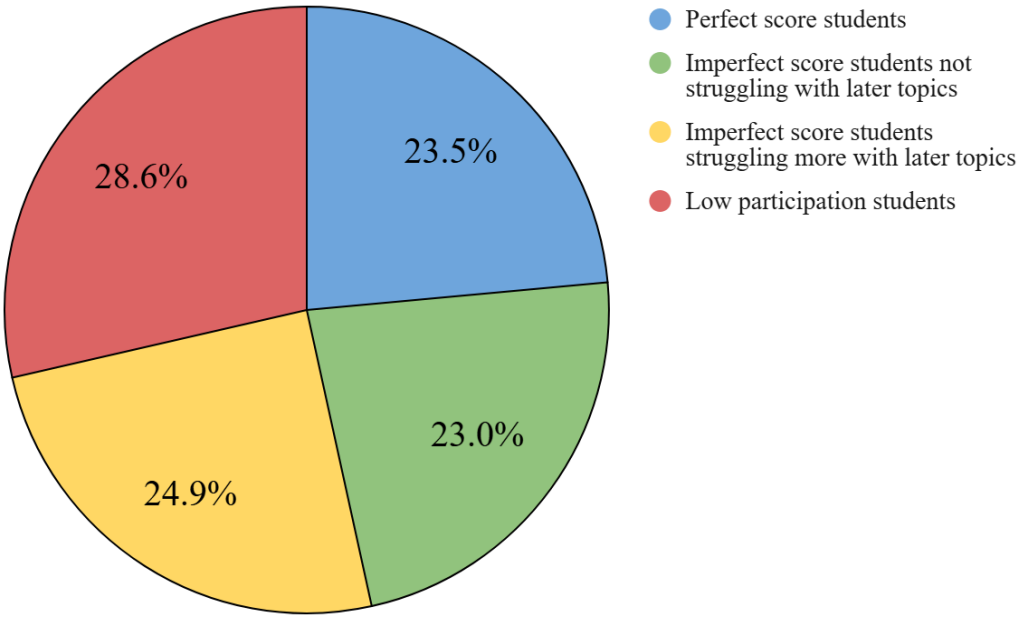

Figure 1: Student performance distribution for C++.

Low participation students did not provide enough data points to draw meaningful conclusions, leading to unreliable results. Thus, the low participation students were excluded from further analysis.

Imperfect score students include those who struggle more with later topics and those who do not. Figure 2 visualizes the correlation between the imperfect student scores and CA position in the C++ book. As students progress through the book, the difficulty of CAs increases. The mean correlation is 0.071 for students of the C++ book. The correlations being significantly less than 1 suggests that struggling students generally do not find later topics, which require more prior knowledge, significantly harder to learn, assuming they complete the CAs in chronological order.

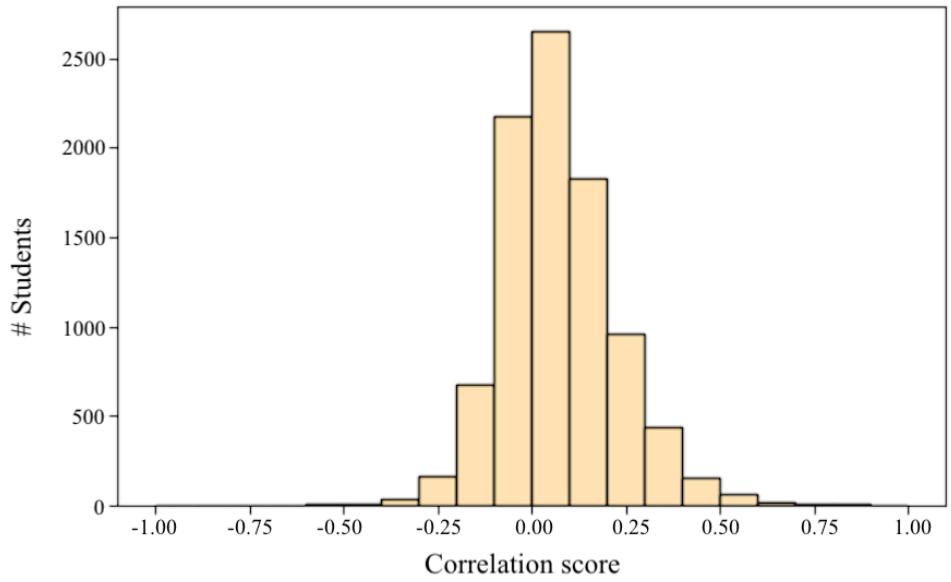

Figure 2. Distribution of correlation between student's decreased passing rate and CA position for C++.

This correlation, 0.071, while positive, is within expected ranges, indicating that the increase in difficulty is gradual. It is reasonable to assume that students will naturally have a slightly harder time with later chapters. Thus, this finding further confirms the observation that later chapters are more challenging to students without suggesting any unreasonable difficulty.

Student behavior

Students were categorized into six distinct behavioral groups based on their interactions with CAs: *Once-and-done, Refiners, Grade motivated, Earnest learners, Earnest strugglers,* and *Lowly motivated.* These categories, defined in section Methods and Metrics, reflect variations of engagement and performance. Data collected across all languages show very similar distributions of these behavioral groups. Figure 3 illustrates the distribution of these behavioral groups for C++, and Table 4 shows the total number of attempts per CA type in C++.

Once-and-Done Student Attempts

CodeOutput accounted for the majority of once-and-done attempts, with 61.93% of all attempts being completed on the first and only try as shown in Figure 3. This suggests that CodeOutput is generally the easiest type of CA to complete in a single attempt.

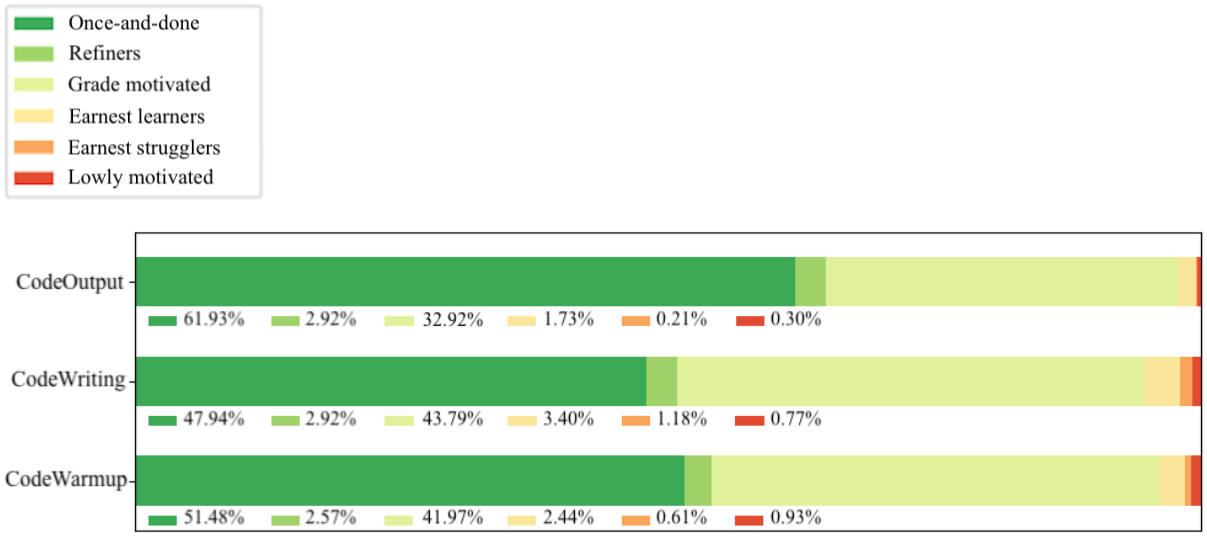

Figure 3. Study behavior group distribution.

Refiner Students

Across all CA types, Refiner students represent a small percentage of students (2.57-2.92%) who strive for mastery beyond passing (Figure 3). In CodeWriting CAs, Refiners average the highest number of after-passing attempts at 1.93, which is a significantly higher average than other CA types (Figure 4), suggesting that these students see value in revisiting these exercises to perfect their understanding. This underscores the capabilities of CodeWriting CAs to be utilized multiple times, providing students with new variations for continued practice. CodeWarmup CAs show the lowest at 1.10 average attempts, indicating limited after-passing engagement which is expected, as these CAs lack any form of randomization.

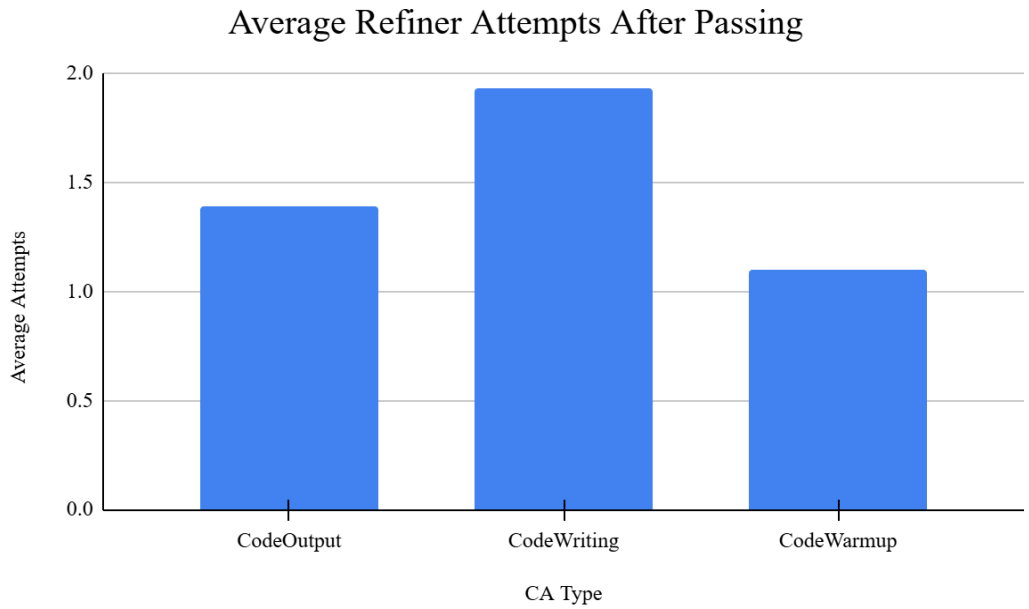

Figure 4. Average number of attempts made after the refiner students pass.

Grade-Motivated Students

Grade motivated students make a strong effort, suggesting that grades drive significant engagement. The average number of attempts for these students peaked for CodeWriting CAs with 3.13 attempts before students passed the problem (Figure 5). CodeOutput has the lowest average attempts (1.78), reinforcing its low level of difficulty.

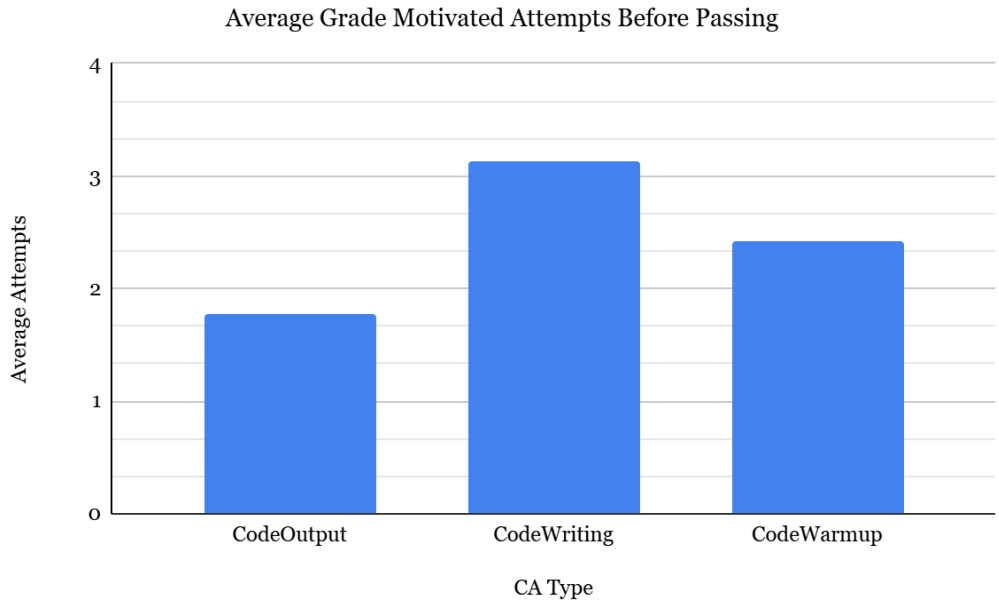

Figure 5. Average number of attempts made before the grade motivated students pass.

Earnest learners

The earnest learners group demonstrates a growth mindset, engaging with the material even after passing the CA successfully. They are a small percentage (1.73-3.40%), but show notable persistence (Figure 3). CodeWarmup CA attempts have the biggest difference after passing, suggesting that this group of students found less value retrying these CAs after passing.

CodeWriting CAs had the highest number of attempts before and after passing, at 3.91 attempts before and 2.75 after (Figure 6 and Figure 7). This suggests that CodeWriting CAs are more challenging than other types, but students still benefit from retries, likely due to the randomization, which enhances practice and mastery.

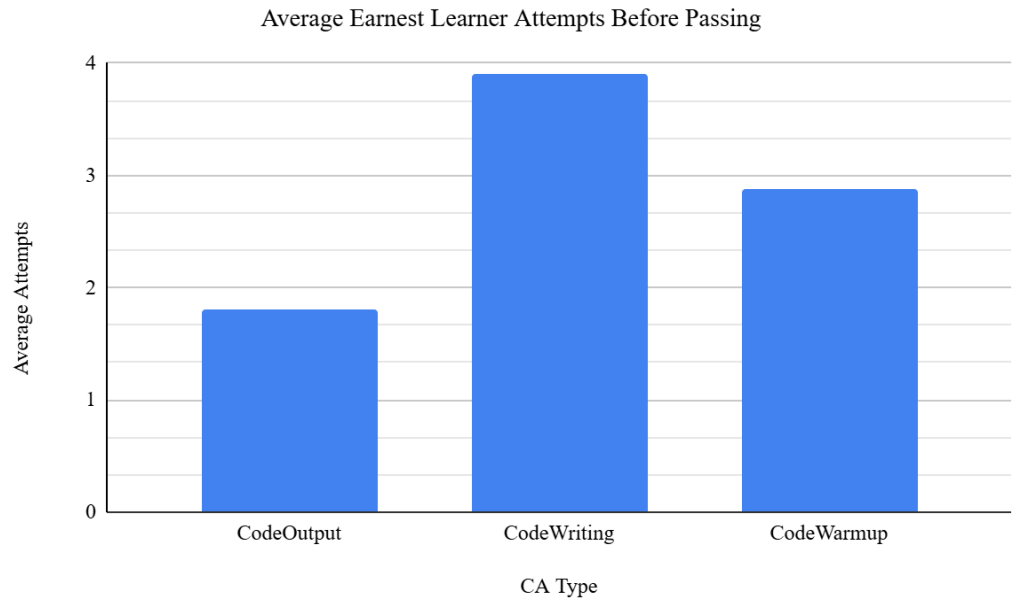

Figure 6. Average number of attempts made before the earnest learner students pass.

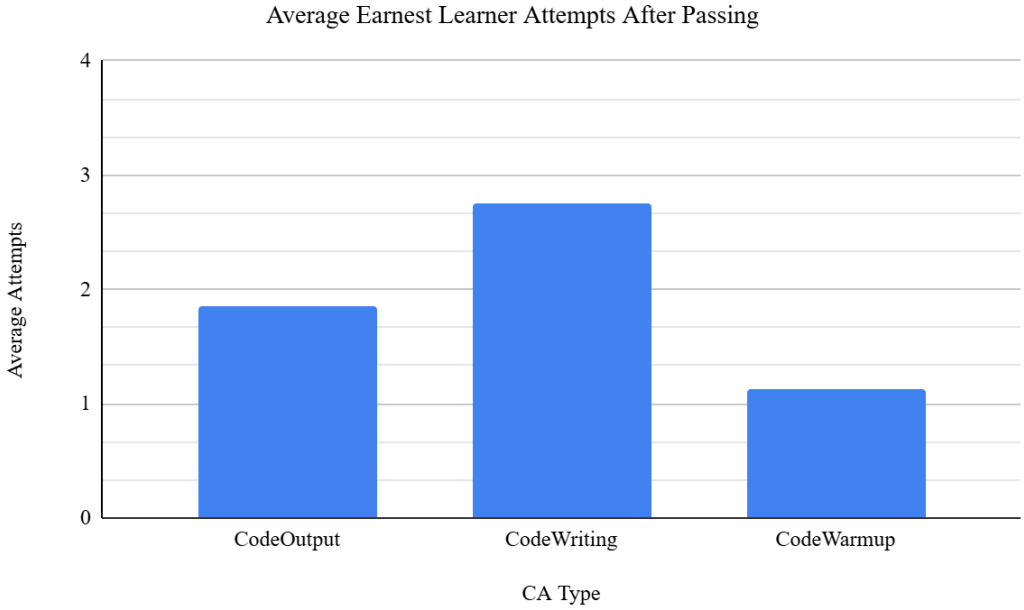

Figure 7. Average number of attempts made after the earnest learner students pass.

Earnest Strugglers

The earnest strugglers group demonstrate the importance of support for struggling students. They represent the smallest group (0.21-1.18%) (Figure 3) of students but attempt the most before passing.

For this group, CodeWriting CAs required the highest effort, with 8.36 average attempts, aligning with its higher difficulty (Figure 8). CodeWarmup CAs prove to be the next difficult CA

type, with CodeOutput CAs once again being the type with the least amount of overall average attempts, further indicating its low difficulty.

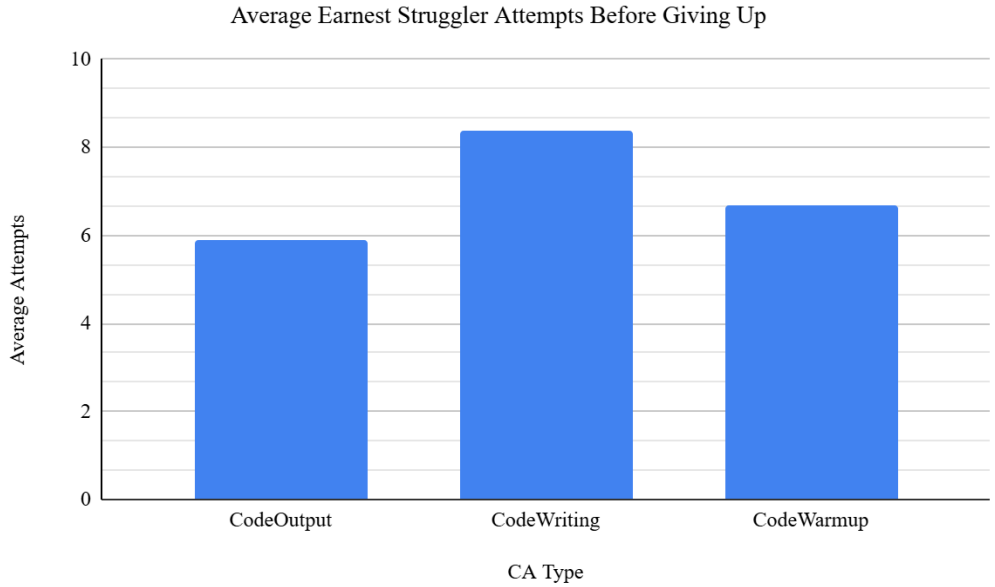

Figure 8. Average number of attempts made before the earnest strugglers give up.

Lowly Motivated

The presence of the lowly motivated group suggests they may need different motivational goals or interests. They represent less than 1% for all CA types (Figure 3), and average the lowest number of average attempts before giving up at 1.26 (Figure 9).

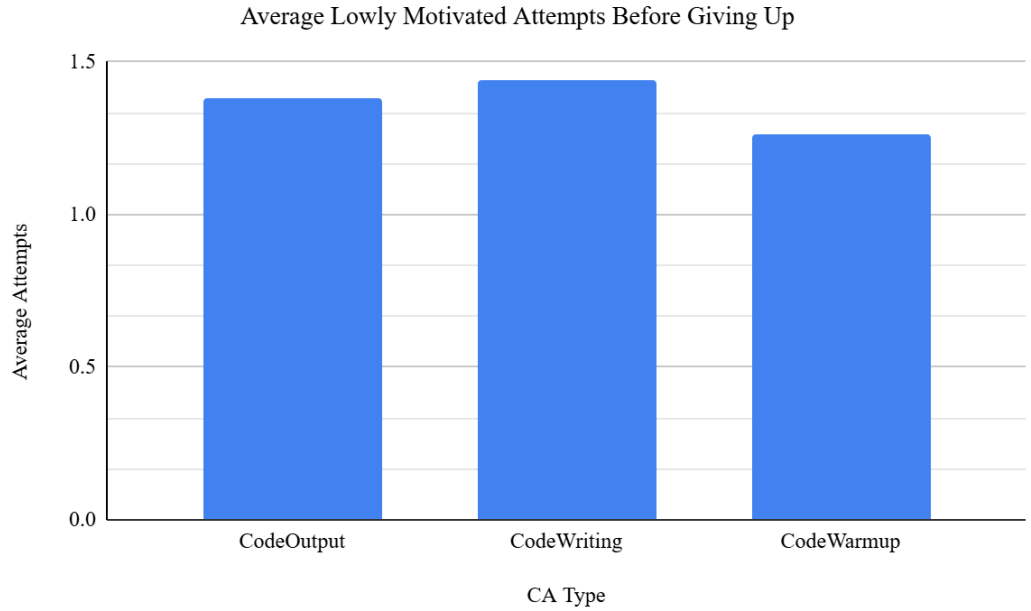

Figure 9. Average number of attempts made before the lowly motivated students give up.

Overall

The differences in attempts before and after passing a CA reflect the varying levels of difficulty and engagement for each CA type. CodeOutput CAs appear to be the best for reinforcing basic skills and addressing simple concepts, while CodeWriting CAs consistently present the greatest challenge for students. Notably, students working on CodeWriting CAs tend to have the highest number of attempts and retries across all CA types.

CodeWriting and CodeWarmup CAs demonstrate the most consistent and balanced patterns in the number of attempts and average retries (Figure 6) and (Figure 7). CodeWriting CAs generally require slightly more attempts before passing compared to CodeWarmup CAs, indicating CodeWarmup CAs may be easier than CodeWriting CAs. CodeWriting appears particularly appealing to earnest learners aiming for mastery, while CodeOutput CAs offer less of a challenge but better for those struggling as indicated by the behavior of earnest strugglers (Figure 3). These patterns are observed across all languages.

Scaffolding

To assess whether the number of levels in CAs correlates to increased effort or impacts student success, CA performance data was analyzed, focusing on passing rates, student attempts, and the role of scaffolding across multiple levels. Data collected across various languages show consistent results. So, only the results from C++ are highlighted here. The distribution of C++ CAs by level of scaffolding is presented in Table 4.

Table 4. Number of CAs with various levels of scaffolding in C++.

Scaffolding	**1-level**	**2-level**	**3-level**	**4-level**	**5-level**	**6-level**	**7-level**
Number of CAs	39	68	55	31	21	5	1

Passing rates across CAs with varying levels is illustrated in Figure 10. Single level CAs and multi-level CAs (2 to 7 levels) maintain consistently high passing rates, ranging from 91.20% to 97.56%. These differences are not statistically significant and may reflect variations in CA difficulty unrelated to the number of levels. While CAs with multiple levels naturally require more effort, the gradual progression in difficulty aligned with effective scaffolding, helps maintain student engagement and does not serve as a significant deterrent.

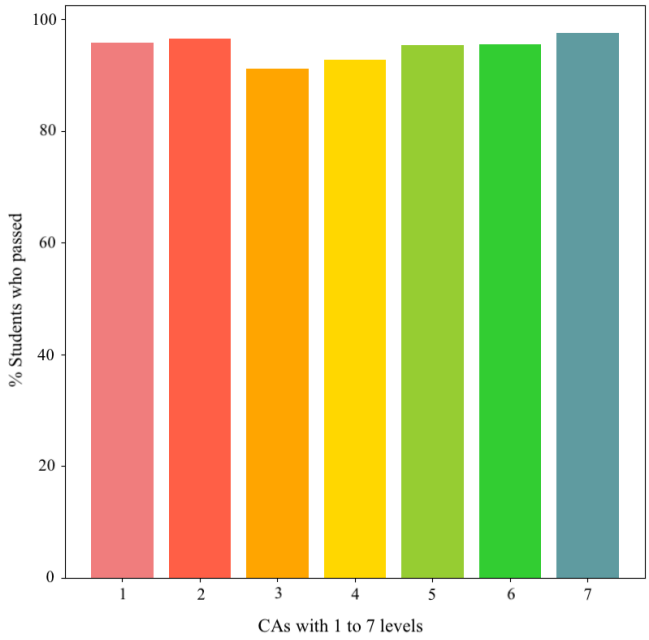

Figure 10. Success rates on CAs with multiple levels of scaffolding in C++.

Student Effort

Given the large number of activities available to students, we wanted to understand the types of CAs that the students put in most of their time and effort. The number of CAs varies by type for each language. Table 5 summarizes the CA count by type for each language. Two groups of students were identified and studied based on the time and effort they put into the CAs: The highly proactive group and the semi-inactive group.

Table 5. CA Type and Language Distribution.

Number of CAs by type	**C++**	**Java**	**Python**	**C**
CodeOutput	68	58	51	35
CodeWriting	69	65	63	49
CodeWarmup	38	53	35	62

Across all languages, the highly proactive students did an average of 103 total CAs, and the average scores on these CAs were at least 90%, indicating a strong level of commitment. For C++, Java, and Python, a majority of those students preferred CodeWriting, possibly due to its challenge or perceived learning benefits. In C, those students preferred CodeWarmup. A significantly fewer number of students in this group preferred CodeOutput. This group achieved the highest mean scores compared to the semi-inactive group, reflecting mastery and persistence (Figure 11).

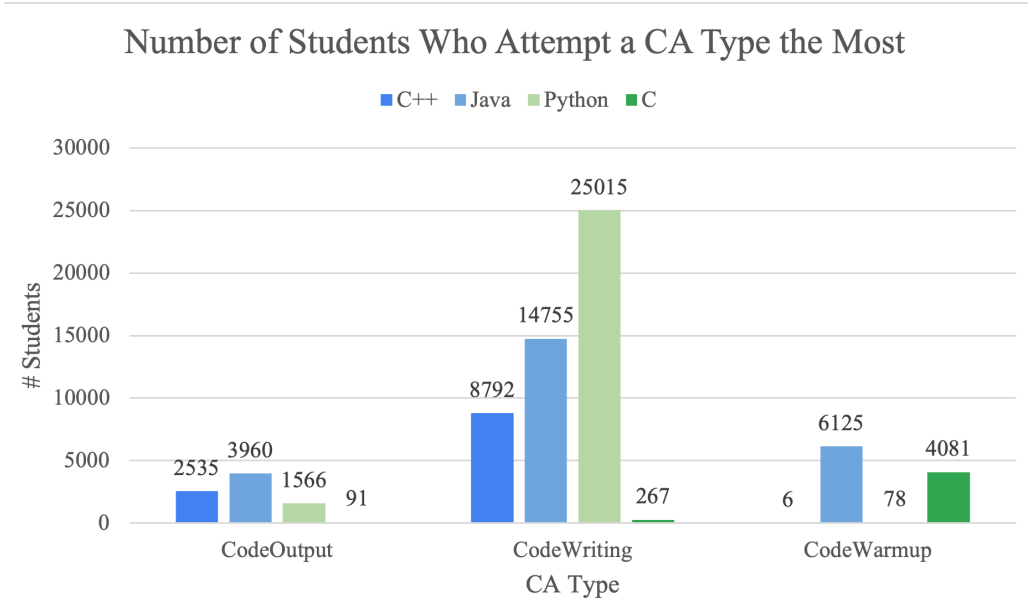

Figure 11. Highly proactive students CA type engagement.

The semi-inactive group averaged only 11 CAs, indicating minimal interaction with the CAs. For C++ and Java, those students had a preference for CodeOutput CAs, likely due to this CA type's simplicity and ease of completion. For Python, those students preferred CodeWarmup. For C, those students preferred CodeWriting. Overall, this group had lower mean scores, particularly with CodeWriting, with the lowest mean score of 70.37% in Python, suggesting this group struggles with more complex tasks (Figure 12).

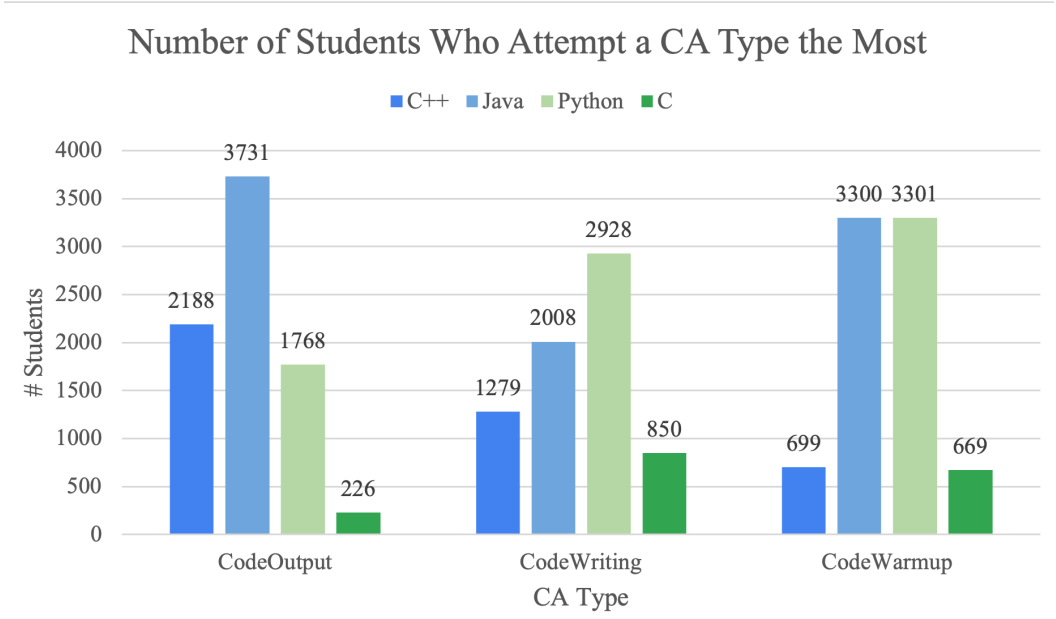

Figure 12. Semi-inactive students CA type engagement.

This trend aligns with broader metrics of student engagement. In books with at least 40 available CAs, the median score on attempted CAs was 98.04%, reflecting students' strong performance on tasks they chose to engage with. However, the median score across all available CAs–considering both completed and unattempted tasks–was 85.47%. This highlights the potential impact of effort on overall achievement: proactive students not only attempted more CAs but performed significantly better across all languages and CA types. Semi-inactive students focused on simpler tasks requiring minimal effort, which limited their overall engagement and achievement.

Discussion and future work

Our findings indicate that students generally do not find later topics, which require greater prior knowledge, significantly harder to learn when they complete CAs in the intended chronological order. Among the different types of CAs, CodeOutput CAs are particularly effective for reinforcing basic skills, making them well-suited for foundational concepts. In contrast, CodeWarmup CAs pose greater challenges that foster deeper understanding but may require additional support for struggling students to maximize their effectiveness. Finally, CodeWriting CAs strike a balance by appealing to grade-motivated students while also fostering opportunities for mastery.

While CAs with multiple levels demand more effort, their gradual progression in difficulty—consistent with scaffolding principles—helps maintain student engagement and does not act as a deterrent. Additionally, our analysis highlights the significant role of student proactivity: proactive students consistently engage more with CAs and achieve higher performance across all CA types, underscoring the strong correlation between effort and achievement. In contrast, less active students tend to favor simpler tasks that require minimal effort, limiting their potential for deeper learning.

Future work could explore the development of additional CAs that span learning objectives across multiple sections or chapters, promoting broader concept integration. For example, combining code writing with Parsons puzzles [13] can help students develop the hierarchical mental structure [14], [15] of how to solve a problem at multiple resolutions.

In light of the findings, we see great potential for CAs to be used to measure students' learning outcomes in assessment tools such as exams. Typically, a CS1 exam may include both long and short questions. Long questions assess students' skills on integrating multiple concepts into a full scale solution. Short questions cover conceptual understanding, logical deduction, and small-scale problem solving with a few lines of code. Randomized CodeOutput CAs ask students to interpret how code works by producing the expected outcomes. Successful completion of such CAs indicate students' ability to apply logical deduction based on their understanding of the code, thus reaching the application level of competence in the Bloom taxonomy [16]. Randomized CodeWriting CA, on the other hand, can assess students' ability to create a solution for a small-scale problem, thus reaching the synthesis level of competence in the same taxonomy. Our exams platform supports a variety of question types, including the highly randomized CodeOutput and CodeWriting CAs. With the high degree of randomization, these assessment tools can measure students' performance based on genuine critical thinking skills rather than

memorized answers. Additionally, comparing exam outcomes when a CodeWriting CA is assigned as homework versus when it is not, may provide useful insights into how prior exposure influences mastery and retention.

We aim to explore how scaffolded, auto-graded activities can better support students with varying levels of engagement and preparedness. While our current approach effectively aids proactive learners, we recognize the need to address the challenges faced by students who may struggle to engage independently. One potential improvement is to incorporate more explicit feedback mechanisms that encourage students to seek help when needed, reinforcing that reaching out for clarification is part of the learning process. Additionally, we plan to examine how continuous revisions to conceptual assessments can enhance accessibility and learning outcomes for all students, particularly those who may require additional guidance.

Further work could design activities that span broader objectives and explore their integration into assessments to evaluate long-term mastery. Additionally, a long-term study on CS student success may involve surveying students on their overall course performance and their preparedness for advanced CS courses upon completion of CS1.

Conclusion

This study demonstrates the effectiveness of a scaffolding approach in CS1 courses for teaching programming concepts across four languages (C++, Java, Python, and C). By analyzing 726 activities with 2,179 levels attempted by over 107,825 students, we validated the role of randomized, auto-graded activities in fostering engagement, and mastery. Gradual progression in difficulty supports learning and engagement, with some activities reinforcing foundational skills and others fostering deeper understanding. Proactive engagement strongly correlates with improved performance.

References

[1] M. D. Gurer, I. Cetin, and E. Top, “Factors affecting students’ attitudes toward computer programming” *Informatics in Education*, vol. 18, no. 2, pp. 281–296, 2019, doi: 10.15388/infedu.2019.13.

[2] R. E. Mayer and R. Clark,“Applying the segmenting and pretraining principles” in *e-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning*, 4th ed. Hoboken, NJ, USA: Wiley, 2016, pp. 203–208.

[3] B. R. Belland, A. E. Walker, N. J. Kim, and M. Lefler, “Synthesizing results from empirical research on computer-based scaffolding in STEM education: A meta-analysis.” *Review of Educational Research,* vol. 87, no. 2, pp. 309–344, 2017.

[4] B. R. Belland. *Instructional scaffolding in STEM education: Strategies and efficacy evidence*. Springer Open, 2017.

[5] I. Horvath, "Reducing cognitive load through scaffolding," *eLearning Industry*, December 18, 2023. [Online]. Available:

https://elearningindustry.com/reducing-cognitive-load-through-scaffolding [Accessed March 17, 2025].

[6] Grand Canyon University, "What Is Scaffolding in Education and How Is It Applied?," *GCU Blog: Teaching and School Administration*, September 19, 2023. [Online]. Available: https://www.gcu.edu/blog/teaching-school-administration/what-scaffolding-in-education-how-applied. [Accessed March 17, 2025].

[7] R. E. Mayer and R. Clark “Does practice make perfect?” in *e-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning*, 4th Ed. Hoboken, NJ, USA: Wiley, 2016, pp. 275-278.

[8] M. C. Lovett, M. W. Bridges, M. DiPietro, S. A. Ambrose, and M. K. Norman, "What Kinds of Practice and Feedback Enhance Learning?" in *How learning works: Eight research-based principles for smart teaching*, 2nd Ed. San Francisco, CA, USA: Jossey-Bass, 2023, pp. 130-161.

[9] Center for Excellence in Universal Design. (n.d.). *The 7 principles of universal design*. [Online] Available: https://universaldesign.ie/about-universal-design/the-7-principles [Accessed March 17, 2025]

[10] *Accessibility at Wiley*. [Online] Available: https://vendors.wiley.com/accessibility/ [Accessed March 17, 2025]

[11] N. Coombs, *Making online teaching accessible: Inclusive Course Design for Students with Disabilities*. Wiley Professional Development (P&T), 2010.

[12] E. Kazakou, A. D. Edgcomb, Y. Rajasekhar, R. Lysecky, and F. Vahid, "Randomized, structured, auto-graded homework: Design philosophy and engineering examples," in *ASEE Annual Conference and Exposition, Conference Proceedings*, *2021 ASEE Virtual Annual Conference*, *July 26-29, 2021*. [Online].

[13] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education, ACE 2006, Hobart, Australia*, *January 16 - 19, 2006,* D. Tolhurst & S. Mann Eds. NSW, Australian Computer Society, vol. 52. pp 157-163

[14] D. Hanson, Instructor's guide to process-oriented guided-inquiry learning. Stony Brook, NY: Stony Brook University, 2006. [E-book] Available: Pacific Crest

[15] C. Wieman, "Why Not Try a Scientific Approach to Science Education?" *Change: The Magazine of Higher Learning*, vol. 39, no. 5, pp 9-15, 2007. https://doi.org/10.3200/CHNG.39.5.9-15

[16] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, *Taxonomy of educational objectives: The classification of educational goals. Vol. Handbook I: Cognitive domain.* New York: David McKay Company, 1956.