Modality Matters: Comparing Block-based, Text-based, and Hybrid Programming Environments for Novice Learners in High School CS Courses

CHRISTINE GKREKA, National and Kapodistrian University of Athens
KOSTAS GAVRILIS, National and Kapodistrian University of Athens

This study investigates the impact of different programming modalities on novice programmers in introductory computer science courses. We conducted a quasi-experimental study comparing block-based, text-based, and hybrid programming environments in high school classrooms over a 5-week period. Using a mixed-methods approach, we analyzed students' conceptual understanding, programming practices, and attitudes across the three modalities. Our findings indicate that both block-based and hybrid approaches led to greater conceptual gains and more positive attitudes compared to text-based programming. However, text-based programming was perceived as more authentic. The hybrid approach showed promise in combining the benefits of both block and text modalities. This study contributes to our understanding of how programming modalities shape novices' learning experiences and has implications for curriculum design and tool selection in introductory CS education.

Keywords: Programming education, Block-based programming, Hybrid programming environments, Novice programmers, Computer science pedagogy

Reference Format:
Christine Gkreka and Kostas Gavrilis. 2024. Modality Matters: Comparing Block-based, Text-based, and Hybrid Programming Environments for Novice Learners in High School CS Courses. *Education and Technology*, Vol. 1, Article 5 (September 2024), 10 pages.

Correspondence: Christine Gkreka, christinegkreka@eds.uoa.gr, National and Kapodistrian University of Athens; Kostas Gavrilis, National and Kapodistrian University of Athens.

1 INTRODUCTION

In recent years, the landscape of introductory computer science education has undergone significant changes, particularly in the realm of programming instruction. The emergence of block-based programming environments, such as Scratch [18] and Alice [5], has introduced new modalities for teaching programming concepts to novices. These visual environments, which allow users to construct programs by dragging and connecting graphical blocks, have gained popularity due to their perceived accessibility and engagement potential [12]. Simultaneously, traditional text-based languages continue to be widely used, while hybrid approaches that blend features of both block and text modalities are beginning to emerge [23].

This evolution in programming instruction tools raises critical questions about the impact of different programming modalities on novice learners. Understanding how these modalities shape students' conceptual understanding, programming practices, and attitudes towards computer science is crucial for developing effective instructional strategies and environments. This study aims to address this need by investigating the research question: How do different programming modalities (block-based, text-based, and hybrid) impact novice programmers' conceptual understanding, practices, and attitudes in introductory CS courses?

The importance of this question is underscored by the growing emphasis on computer science education at the K-12 level and the need to broaden participation in computing [8]. Choosing appropriate programming environments and instructional approaches for novices can significantly influence their learning trajectories and long-term engagement with computer science.

Prior studies have contributed valuable insights into the potential benefits of block-based programming for novices. Research has shown that block-based environments can support the development of computational thinking skills [7] and increase student engagement [12]. Comparative studies have suggested that block-based approaches may lead to faster initial learning of programming concepts compared to text-based languages [22].

However, existing research has several limitations. Many studies have focused on either block-based or text-based environments in isolation, with fewer investigations into hybrid approaches. Additionally, much of the research has been conducted in informal learning contexts or with younger students, leaving gaps in our understanding of how different modalities function in formal high school computer science classrooms. Furthermore, there is a need for more comprehensive studies that examine not only conceptual understanding but also the development of programming practices and attitudes across different modalities.

Our study addresses these gaps by conducting a quasi-experimental comparison of block-based, text-based, and hybrid programming environments in high school introductory CS courses. By employing a mixed-methods approach that combines quantitative assessments of learning outcomes with qualitative analyses of student practices and attitudes, we aim to provide a more holistic understanding of the impact of programming modalities on novice learners.

This research makes several key contributions to the field of computer science education. First, it offers empirical evidence on the relative effectiveness of different programming modalities in supporting conceptual understanding, fostering productive programming practices, and shaping student attitudes. Second, it provides insights into the potential of hybrid approaches, which have been less extensively studied but may offer a promising middle ground between block-based and text-based programming. Finally, our findings have practical implications for curriculum design and tool selection in introductory computer science courses, potentially informing strategies to improve learning outcomes and increase student engagement in computing.

By examining these issues, we aim to contribute to the ongoing dialogue about how best to introduce programming to novices and support their development as computational thinkers and problem solvers in an increasingly digital world.

2 LITERATURE REVIEW

2.1 Block-based vs Text-based Programming for Novices

Block-based programming environments have gained significant popularity as introductory tools for teaching novice programmers in recent years. These environments use a visual, drag-and-drop interface where code is represented as interlocking blocks, aiming to reduce syntax errors and cognitive load for beginners [3]. In contrast, traditional text-based environments require users to type code directly, offering more flexibility but also introducing potential for syntax errors.

Several studies have compared learning outcomes between block-based and text-based approaches for novices. Weintrop and Wilensky [22] found that students performed better on concept assessments when questions were presented in a block-based format compared to text, suggesting block-based representations may aid conceptual understanding. However, Price and Barnes [16] observed no significant difference in learning outcomes between block-based and text-based versions of the same language, though block-based users completed tasks more quickly. These mixed results indicate that while block-based tools may offer some advantages, their benefits are not universal across all learning contexts and measures.

Long-term impacts of starting with block-based programming remain unclear. Armoni et al. [1] found that students with prior Scratch experience performed better initially in a text-based course, but differences diminished over time. This suggests block-based experience may provide an early advantage that fades as students transition to text. More research is needed on how early modality choices influence learners' programming trajectories.

2.2 Emerging Hybrid and Dual-modality Approaches

Recognizing potential benefits and drawbacks of both block-based and text-based approaches, researchers have begun exploring hybrid and dual-modality programming environments. These tools aim to combine advantageous aspects of both paradigms, supporting learners as they progress from novice to more advanced programming practices.

Dual-modality environments like Pencil Code allow users to seamlessly switch between block and text representations of the same code [2]. Studies of such environments have shown that learners leverage both modalities, often using blocks for exploration and text for efficiency as they gain proficiency [20]. This suggests value in providing multiple representational options to suit different tasks and stages of learning.

Frame-based editing represents another hybrid approach, blending block-like structure with keyboard-driven text editing [10]. Early evaluations indicate frame-based editing may reduce errors while maintaining the expressive power of text [17]. However, more extensive studies are needed to assess long-term learning impacts of these novel modalities compared to traditional approaches.

2.3 Programming Practices and Modality

While much research has focused on learning outcomes, fewer studies have examined how different programming modalities shape novices' actual coding practices and behaviors. Understanding these effects is crucial for designing environments that cultivate productive programming habits.

Block-based environments have been observed to encourage exploratory, "bricolage" approaches to coding, where learners experiment by assembling pieces [11]. This aligns with constructionist learning theories but may lead to less-structured programs. Text environments, in contrast, may

promote more planned, top-down approaches [15]. How these early practice patterns influence long-term development of programming skills remains an open question.

Some evidence suggests that block-based programmers may develop problematic habits that hinder later text-based learning, such as creating overly fine-grained procedures [15]. However, other studies have found that block-based experience supports developing key computational thinking practices that transfer to text contexts [7]. More research comparing programming practices across modalities could help clarify these conflicting findings.

2.4 Affective Impacts of Programming Modalities

The choice of programming modality may influence not only learners' performance and practices, but also their attitudes, motivation, and sense of self-efficacy regarding computer science. These affective factors can significantly impact persistence and long-term engagement with programming.

Multiple studies have found that block-based environments tend to generate high levels of enjoyment and engagement among novices [12, 26]. The visual nature and immediate feedback of these tools appear to create a low-stress entry point to coding. However, some research suggests older learners may perceive block-based tools as less authentic or "toy-like" compared to text environments [21].

Regarding self-efficacy, findings have been mixed. While some studies report increased programming confidence after block-based experiences [6], others have found no significant difference compared to text-based learning [24]. The impact on long-term interest in computer science also remains unclear, with some evidence suggesting block-based tools may boost initial enthusiasm but not necessarily translate to sustained engagement [14]. Given the potential for programming experiences to shape learners' computational identities, especially for underrepresented groups in computer science, further research on the affective impacts of different modalities across diverse populations is crucial.

2.5 Gaps in Current Research

While existing literature provides valuable insights into the effects of different programming modalities, several important gaps remain. First, most comparative studies have focused on short-term interventions, leaving questions about long-term impacts on learning trajectories. Longitudinal studies tracking students from initial exposure through more advanced programming could clarify how early modality choices influence later outcomes.

Additionally, much research has concentrated on either block-based or text-based approaches in isolation, with fewer studies examining hybrid or transitional models. As the field moves towards more flexible, multi-modal environments, more work is needed to understand how learners navigate and benefit from these complex tools.

Finally, there is a need for more holistic studies that simultaneously examine conceptual understanding, programming practices, and affective outcomes. Most existing research has prioritized one or two of these dimensions, but a comprehensive view of how modality shapes the overall learning experience could provide crucial insights for curriculum and tool design. By addressing these gaps, future research can contribute to a more nuanced understanding of how programming modalities influence novice learners, ultimately informing the development of more effective and inclusive computer science education approaches.

3 RESEARCH DESIGN

This study was designed to address the following research question – *How do different programming modalities (block-based, text-based, and hybrid) impact novice programmers' conceptual understanding,*

practices, and attitudes in introductory CS courses? To investigate this question, we conducted a quasi-experimental study comparing three different programming modalities across multiple dimensions. The study took place over a 5-week period in introductory high school computer science classes.

3.1 Participants and Setting

Participants were 90 students enrolled in an introductory programming course at a large urban public high school. The students were divided into three classes of 30 students each. Each class was assigned to one of three conditions:

- Block-based programming using Scratch
- Text-based programming using Python
- Hybrid block/text programming using a custom Pencil.cc environment

The same teacher taught all three classes, following an identical 5-week curriculum covering fundamental programming concepts like variables, loops, conditionals, and functions. The only difference between conditions was the programming environment used.

3.2 Data Collection

We collected multiple forms of data to assess the impact of programming modality:

3.2.1 Pre/Post Assessments. Students completed identical pre- and post-assessments at the beginning and end of the 5-week period. These assessed conceptual understanding of programming concepts through multiple choice and short answer questions.

3.2.2 Programming Logs. The Pencil.cc environment automatically logged all student programming activity, including code written, edits made, and program runs. This provided detailed data on students' programming practices and behaviors.

3.2.3 Attitudinal Surveys. Students completed surveys at the beginning and end of the study period measuring attitudes toward programming, self-efficacy, and interest in future CS courses.

3.2.4 Semi-Structured Interviews. We conducted 20-30 minute interviews with a subset of students from each condition at the end of the study. These probed students' experiences, perceptions, and approaches to programming.

3.2.5 Classroom Observations. Researchers observed and took field notes during class sessions to capture qualitative data on student engagement and behaviors.

3.3 Data Analysis

We used a mixed methods approach to analyze the multi-faceted dataset:

3.3.1 Quantitative Analysis. Pre/post assessment scores were compared across conditions using ANOVA to assess differences in conceptual learning gains. Programming log data was analyzed to compare metrics like frequency of code compilation, time spent programming, and program length across conditions. Survey responses were analyzed using t-tests to evaluate changes in attitudes.

3.3.2 Qualitative Analysis. Interview transcripts and classroom observation notes were analyzed using thematic coding to identify key themes related to students' experiences and practices in each modality. Representative vignettes were selected to illustrate typical student approaches.

3.3.3 Integrated Analysis. Quantitative and qualitative findings were triangulated to develop a holistic understanding of how each programming modality shaped students' learning. We looked for areas of convergence and divergence across data sources.

This mixed-methods design allowed us to assess the impact of programming modality across multiple dimensions - conceptual understanding, programming practices, and student attitudes and experiences. By combining controlled comparisons with rich qualitative data, we aimed to develop a nuanced picture of how different modalities shape novice programmers' learning trajectories.

4 RESULTS

Our analysis revealed several key findings regarding the impact of different programming modalities on novice programmers. We present these results organized by our main areas of investigation: conceptual understanding, programming practices, and student attitudes.

4.1 Conceptual Understanding

To assess differences in conceptual learning across modalities, we compared pre- and post-assessment scores.

Table 1 shows the mean pre- and post-test scores for each condition.

Condition	Pre-test Mean (SD)	Post-test Mean (SD)	Mean Gain
Block-based	54.3% (12.2%)	66.6% (13.4%)	12.3%
Text-based	51.6% (14.5%)	58.8% (14.6%)	7.2%
Hybrid	53.1% (13.8%)	64.7% (14.0%)	11.6%

Table 1. Pre- and post-test scores by condition

An ANOVA revealed a significant main effect of condition on learning gains ($F(2,87) = 3.72$, $p < .05$). Post-hoc Tukey tests showed that both the block-based and hybrid conditions had significantly higher gains than the text-based condition ($p < .05$), but did not differ significantly from each other.

Breaking down performance by specific programming concepts revealed some interesting patterns (Figure 1).

For loops and conditionals, the block-based condition showed the highest gains, followed by hybrid, then text. For variables and functions, the differences were less pronounced, with all conditions showing similar improvements.

4.2 Programming Practices

Analysis of the programming log data revealed several differences in how students approached programming tasks across modalities. Students in the hybrid condition compiled their code significantly more often than those in the other conditions ($F(2,87) = 8.71$, $p < .001$). On average, hybrid students compiled 1,073 times over the 5 weeks, compared to 733 for block-based and 743 for text-based. There were no significant differences in total time spent programming across conditions. However, the distribution of time differed:

- Block-based students spent more time in the initial planning stages
- Text-based students spent more time debugging syntax errors
- Hybrid students alternated frequently between blocks and text modes

Programs written in the block-based condition were significantly longer on average than those in the other conditions ($p < .05$). However, text-based programs tended to use more advanced constructs like nested loops and complex boolean logic.

4.3 Student Attitudes

Survey and interview data provided insight into students' experiences and perceptions across modalities. All conditions showed increases in programming self-efficacy, but the increase was

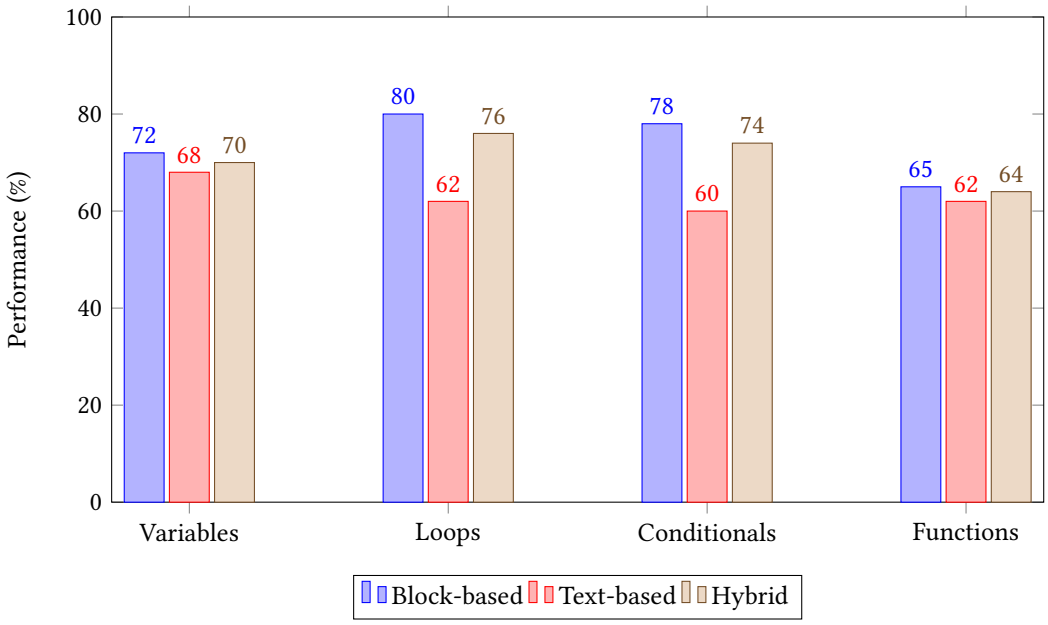

Fig. 1. Performance by concept across programming modalities

largest for the block-based condition (mean increase of 1.2 on a 5-point scale, compared to 0.8 for text and 0.9 for hybrid).

Block-based and hybrid students reported significantly higher enjoyment of programming activities compared to text-based students ($p < .01$). Classroom observations also noted higher visible engagement in these conditions. Text-based students rated their programming experience as more "authentic" and similar to "real programming" compared to block-based students ($p < .05$). Hybrid students' ratings fell in between. Block-based and hybrid students expressed significantly higher interest in taking future CS courses compared to text-based students ($p < .05$).

4.4 Qualitative Insights

Thematic analysis of interview data revealed several key themes:

- Block-based students appreciated the visual nature and ease of use, but some felt it was "childish"
- Text-based students felt challenged but valued learning a "real" language
- Hybrid students liked the flexibility to switch between modes based on the task
- All groups struggled with certain concepts (e.g., variables, functions) regardless of modality

These qualitative insights help contextualize the quantitative findings and provide a richer understanding of students' experiences across modalities.

5 DISCUSSION

Our study provides insights into how different programming modalities shape novice programmers' learning experiences in introductory CS courses. We discuss our findings in relation to conceptual understanding, programming practices, and student attitudes, considering their implications for CS education.

5.1 Impact on Conceptual Understanding

Our results indicate that both block-based and hybrid modalities led to greater conceptual gains compared to the text-based condition. This aligns with previous research suggesting that visual programming environments can support novices in grasping fundamental programming concepts [7, 22]. The similar performance of block-based and hybrid conditions is particularly noteworthy, suggesting that the hybrid approach may offer the conceptual benefits of blocks while potentially easing the transition to text-based programming.

The concept-specific findings provide additional nuance. The advantage of block-based and hybrid approaches was most pronounced for loops and conditionals, concepts that are often challenging for novices [19]. This may be due to the visual representation of these constructs making their structure more apparent. However, the similar performance across conditions for variables and functions suggests that some concepts may be equally challenging regardless of modality.

These findings highlight the potential of block-based and hybrid approaches for introducing key programming concepts. However, they also underscore the need for targeted instructional strategies to address persistently challenging concepts across modalities.

5.2 Shaping of Programming Practices

The differences in programming practices across modalities reveal how interface design can influence novices' approaches to coding tasks. The higher compilation frequency in the hybrid condition suggests that students were leveraging the ability to quickly switch between blocks and text to iteratively develop and test their code. This aligns with the idea of "tinkering" as a valuable learning strategy in programming [4].

The longer programs in the block-based condition, contrasted with the more advanced constructs in text-based programs, illustrate how modality can shape the nature of students' code. Block-based environments may encourage more expansive, step-by-step solutions, while text-based environments might push students towards more concise, abstracted approaches. This trade-off between accessibility and sophistication is a key consideration in designing introductory programming experiences [9].

The time distribution findings highlight how different modalities present distinct challenges and affordances. Block-based environments appear to support initial planning and ideation, while text-based environments demand more time for syntax-related problem-solving. The hybrid approach seems to offer a balance, allowing students to leverage the strengths of each modality as needed.

These practice-related findings underscore the importance of considering not just what students learn, but how they learn when designing programming environments. The choice of modality can significantly shape students' problem-solving approaches and coding strategies.

5.3 Influence on Student Attitudes

The attitudinal results reveal both the benefits and potential drawbacks of different programming modalities. The higher self-efficacy and enjoyment in block-based and hybrid conditions align with previous research on the motivational benefits of visual programming environments [12]. These positive attitudes could be crucial for encouraging continued engagement with CS, particularly for underrepresented groups [13].

However, the perception of text-based programming as more "authentic" highlights a potential tension. While block-based approaches may be more engaging, there's a risk that students might view them as less relevant to real-world programming. This echoes concerns raised by Meerbaum-Salant et al. [15] about potential drawbacks of block-based languages.

The hybrid approach appears promising in potentially bridging this gap, offering both the engagement of blocks and a clearer connection to text-based coding. This aligns with recent work on dual-modality environments that aim to ease the transition between blocks and text [25].

The higher interest in future CS courses among block-based and hybrid students is particularly encouraging. It suggests that these approaches may be effective in not just teaching concepts, but also in fostering long-term engagement with computer science. This could have significant implications for broadening participation in CS [8].

5.4 Implications for CS Education

Our findings have several important implications for CS education:

(1) The effectiveness of block-based and hybrid approaches in supporting conceptual understanding suggests that these modalities should be seriously considered for introductory programming courses, particularly at the high school level.
(2) The distinct programming practices fostered by different modalities highlight the need for educators to carefully consider how their choice of environment might shape students' problem-solving approaches.
(3) The attitudinal benefits of block-based and hybrid approaches, combined with their conceptual advantages, make a strong case for their use in introductory contexts. However, educators should be mindful of the need to explicitly connect these experiences to text-based programming.
(4) The potential of hybrid approaches to combine the benefits of blocks and text warrants further exploration. These environments could offer a promising path for smoothing the transition from visual to traditional programming languages.
(5) Given the persistent challenges with certain concepts across modalities, there's a need for targeted instructional strategies and tools to support learning of particularly difficult ideas, regardless of the programming environment used.

These implications suggest a nuanced approach to choosing and implementing programming environments in CS education, one that considers both the immediate learning outcomes and the longer-term trajectory of students' CS engagement.

6 CONCLUSION

This study provides evidence that the choice of programming modality significantly impacts novice programmers' learning experiences in introductory CS courses. Our findings suggest that block-based and hybrid approaches offer advantages in terms of conceptual understanding and fostering positive attitudes towards programming, while text-based environments may better prepare students for perceived real-world programming tasks. The hybrid approach shows particular promise in bridging the gap between the accessibility of blocks and the authenticity of text. These results highlight the need for carefully considered choices in introductory programming environments, balancing immediate learning outcomes with long-term goals of CS education. Future research should focus on refining hybrid approaches and developing pedagogical strategies that leverage the strengths of different modalities to support diverse learners in their journey into computer science.

REFERENCES

[1] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 1–15.
[2] David Bau, D Anthony Bau, Mathew Dawson, and C Sydney Pickens. 2015. Pencil code: block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children*. 445–448.

[3] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. *Commun. ACM* 60, 6 (2017), 72–80.

[4] Matthew Berland, Taylor Martin, Tom Benton, Carmen Petrick Smith, and Don Davis. 2013. Learning, understanding, and computational algorithmic thinking: Conceptual learning in high school students' beginning programming courses. *Journal of the Learning Sciences* 22, 4 (2013), 564–599.

[5] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges* 15, 5 (2000), 107–116.

[6] Diana Franklin, Gabrielle Skifstad, Reiny Rolock, Isha Mehrotra, Valerie Ding, Alexandria Hansen, David Weintrop, and Danielle Harlow. 2017. Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 231–236.

[7] Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education* 25, 2 (2015), 199–237.

[8] Mark Guzdial. 2015. *Learner-centered design of computing education: Research on computing for everyone.* Morgan & Claypool Publishers.

[9] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 83–137.

[10] Michael Kölling, Neil CC Brown, and Amjad Altadmri. 2017. Frame-based editing: Combining the best of blocks and text programming. *2017 IEEE Blocks and Beyond Workshop (B&B)* (2017), 1–4.

[11] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[12] John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin* 40, 1 (2008), 367–371.

[13] Jane Margolis and Allan Fisher. 2003. *Unlocking the clubhouse: Women in computing.* MIT press.

[14] Lauren E Margulieux, Richard Catrambone, and Mark Guzdial. 2012. Engaging learners in computer science education through computational remixing with EarSketch. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 85–90.

[15] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in scratch. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (2011), 168–172.

[16] Thomas W Price and Tiffany Barnes. 2015. Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. 91–99.

[17] Thomas W Price, Neil CC Brown, Dragan Lipovac, Tiffany Barnes, and Michael Kölling. 2016. Evaluation of a frame-based programming editor. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. 33–42.

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: Programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[19] Elliot Soloway and Kate Ehrlich. 1983. Novice mistakes: Are the folk wisdoms correct? *Commun. ACM* 26, 11 (1983), 853–865.

[20] David Weintrop and Nathan Holbert. 2017. From blocks to text and back: Programming patterns in a dual-modality environment. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 633–638.

[21] David Weintrop and Uri Wilensky. 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*. 199–208.

[22] David Weintrop and Uri Wilensky. 2015. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. 101–110.

[23] David Weintrop and Uri Wilensky. 2017. Between a block and a typeface: Designing and evaluating hybrid programming environments. In *Proceedings of the 2017 Conference on Interaction Design and Children*. 183–192.

[24] David Weintrop and Uri Wilensky. 2017. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–25.

[25] David Weintrop and Uri Wilensky. 2018. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction* 17 (2018), 83–92.

[26] Amanda Wilson and David C Moffat. 2010. Evaluating scratch to introduce younger schoolchildren to programming. In *Proceedings of the 22nd Annual Psychology of Programming Interest Group Workshop*.