Technology, Knowledge and Learning
https://doi.org/10.1007/s10758-022-09636-4

ORIGINAL RESEARCH

Scaffolding Computational Thinking Through Block Coding: A Learner Experience Design Study

Andrew A. Tawfik[1] · **Linda Payne**[1] · **Andrew M. Olney**[2]

Accepted: 23 November 2022

Abstract
Theorists and educators increasingly highlight the importance of computational thinking in STEM education. While various scaffolding strategies describe how to best support this skillset (i.e., paired programming, worked examples), less research has focused on the design and development of these digital tools. One way to support computational thinking and data science is through block coding and other ways that visualize the coding process. However, less is known about the learning experience design of these tools. Based on this gap, this work-in-progress study compared the learning experience design of novices and those with more advanced understanding of computational thinking. Results found differences emerge in the perceived dynamic interaction and scaffolding constructs of learning experience design. Implications for theory and practice are discussed.

Keywords Computational thinking · Scaffolding · STEM · Data science

1 Introduction

Over the past fifteen years, computational thinking (CT) has become popular in education circles, leading to many proposals for standards and curricular frameworks (Computer Science Teachers Association, 2017; K-12 Computer Science Framework Steering Committee, 2016). The current interest can largely be traced back to an influential opinion piece by the computer scientist Wing (2011), who defined CT as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that

✉ Andrew A. Tawfik
aatawfik@gmail.com

Linda Payne
lapayne2@memphis.edu

Andrew M. Olney
aolney@memphis.edu

[1] University of Memphis, 3798 Walker Ave, 38111 Memphis, TN, USA

[2] University of Memphis, 365 Innovation Dr, 38152 Memphis, TN, USA

Published online: 20 December 2022

can be effectively carried out by an information-processing agent" (Wing, 2011, p. 20). This definition is aligned with previous notions of algorithmic thinking and procedural thinking (Tedre & Denning, 2016), and has somewhat overshadowed an alternative perspective on CT arising from computational science, one that focuses on modeling processes and analyzing datasets. From the computational science view, CT is about using computers to think about phenomena rather than using computers to change how we think. It is evident in forecasts of the weather, visualizations of particle collisions, and discovery of disease markers in genetic databases. This science-first view of CT aligns with simulation and the focus of the present study, data science education. We are especially interested in early acquisition of CT concepts that are necessary for understanding and manipulating scientific datasets.

An important element of computational thinking and data science includes the interaction of the tools used to think about and support this STEM skillset. Whereas previous literature explored computational thinking learning outcomes (e.g., posttest scores) (Shute et al., 2017), theories and studies are increasingly exploring the designs CT learning environments and how to embed specific scaffolds, such as paired programming support, worked examples, and others (Hsu et al., 2018; Wang et al., 2021). The evolution of these supports thus highlights the importance of the human-computer interaction aspect of these learning tools, also known as learning experience design (LXD). Although competing definitions have recently emerged to better understand HCI within education (Chang & Kuwata, 2020; Gray, 2020), LXD can be broadly defined as interactions with technology that support human-centered learning (Tawfik et al., 2022). To date, the learning outcomes of interactions have been implicitly referenced through other constructs, such as extraneous cognitive load (Novak et al., 2018), effort expectancy (El-Masri & Tarhini, 2017), and others. However, theorists increasingly advocate for a more holistic view of learning experience design that considers both technical (e.g. usability, UX) and cognitive considerations embedded within interactions. Others have also expanded this definition to focus on goal-oriented behavior or broader socio-technical components of learning experience design (Gray, 2020; Jahnke et al., 2020).

Although there is growing discourse about CT and data science, there are few empirical studies that specifically detail the unique interaction patterns for complex learning processes. In many cases, this aspect was often briefly considered as part of the summative evaluation phase during the design and development of learning technologies (Lu et al., 2022). Given the lack of dedicated research focus, Novak et al. (2018) commented that "despite a growing body of research in the area of digital learning and information processing, the literature on how people process and interact with information on electronic devices and computers is still very scarce" (p. 151). This is important as CT expands beyond practitioners and becomes more embedded in various curricula (e.g., pre-service teaching; data science students) that cater to diverse skill sets and their unique user-needs. It is further important to understand the interaction patterns to ensure CT and data science are accessible across a broader array of learners. This specific work-in-progress study attempts to address this gap, especially as it explores LXD patterns as different types of learners (novices, advanced) interact with CT tools. The article first outlines the literature describing computational thinking, especially differences that emerge in experts and novices. We also describe the emergence of LXD as a field of study, including the varying definitions and empirical research around the phenomenon. Finally, we present a study that looks at the

LXD of novices and advanced CT learners using a tool designed to scaffold specific aspects of computational thinking and data science education.

2 Literature Review

2.1 Emergence of Computational Thinking

Although the current interest in CT was largely sparked by Wing (2011), the history of CT can be traced back some 50 years prior (Tedre & Denning, 2016). Despite this long history, or perhaps because of it, there is no consensus definition of CT, and NRC workshops on CT have shown a variety of viewpoints (National Research Council, 2010, 2011). Perhaps the most prominent is Wing's definition, which can be summarized as "think like a computer programmer." As discussed by Tedre & Denning (2016), Wing's definition is consistent with previous perspectives on procedural and algorithmic thinking that attempted to capture what made computer science distinct from mathematics. This perspective on CT takes a micro view; that is, one that focuses on key characteristics of computational thinking rather than the goals or products of computational thinking. Example characteristics include abstraction, decomposition, parallel processing, and debugging. However, a close inspection of the perspectives raised in the NRC workshops on CT suggests that most other definitions take a macro view that focuses on the goal and products of CT. The precise goals and products tend to be discipline-specific and aligned with the use of computers in that discipline. Engineering-based disciplines primarily view CT in terms of problem solving, since it is common to program computers to find custom solutions in these fields. Fields that do not program computers instead focus on using existing software in creative ways to achieve goals, such as scientific fields focus on building models and analyzing data. Notably these models are runnable process models, like simulations, rather than traditional box and arrow models best thought of as diagrams. Rather than solving a specific problem, these models and analyses are aimed at providing an understanding or explanation of the phenomena in question.

While these micro and macro views may seem irreconcilably different, they in fact share a deep commonality that can be traced back to Papert (1980), who appears to have been the first to use the term "computational thinking" in the modern sense. Papert's orientation matched Wing's in the sense that he viewed CT as a kind of thinking developed from programming computers. However, Papert's view is also deeply rooted in microworlds, computer-based simulations of the world. Papert's Logo learning environment included a robot named "turtle" and a graphical replica of it, referred to as a "display turtle" (Papert, 1980; Solomon et al., 2020), and a quintessential activity in Papert's instruction would be for a child to program the turtle to do something in a microworld. For example, a child would give the turtle simple commands to move it a certain number of spaces and in a particular direction on the screen [e.g., turtle.forward (1)] and then in different directions [e.g., turtle, right (12) to draw a shape] (Li et al., 2020; MacConville et al., 2022; Papert 1980). This activity can be viewed as solving a problem (to make the turtle do something), but it is also in reference to a model of the world.

Papert also explored microworlds with alternative physics as a means of teaching Newtonian mechanics. Thus Papert's early work on CT blended the problem solving view of CT and the modeling and simulation view of CT. We likewise argue that these views are not

in conflict, but rather that problem solving in CT always references a model. Sometimes this model is not as obvious as Papert's microworlds because it is an abstract information processing model, or even the model of computation represented by the physical computer. Scientific analysis-driven CT is an example of CT that references an abstract model, both in terms of the data (a measurement model) and in terms of the decisions made at each stage of the analysis (an analytic argument model). Because these models are more abstract than Papert's microworlds, they pose a significant challenge to learners in this area of CT.

2.2 Computational Thinking Tools

Tools for CT reflect the micro and macro views of CT, and an examination of these tools reveals a continuum of CT across disciplines. On the micro end of the continuum, programming languages are the principal tool in which they represent an idealized model of a computer (Du Boulay et al., 1999). Programmers must learn this idealized model and construct mechanisms by supplying the computer with instructions written in the language (i.e., programs). To do this correctly, programmers must be able to mentally simulate the mechanism in order to understand the program's control flow and how values change as the program is executed, which is extremely difficult when runtime behaviors of mechanisms are not visible and have limited feedback (Du Boulay et al., 1999; Soloway, 1986). Modern programming environments provide various accessory tools to support the programming task (desRivieres & Wiegand, 2004). Further along the continuum are tools like Simulink (Chaturvedi, 2017) that are used to create simulations using a higher-level language (e.g., block diagrams). Simulink/MATLAB is widely used in engineering disciplines for electronic and mechanical design. At the macro end of the continuum are pre-built simulators that have abstracted away from programming entirely. Such simulators are often found in computational science domains, like weather forecasting. For example, the Weather Research and Forecasting Model (Powers et al., 2017) allows scientists to vary parameters and initial conditions to study specific weather processes.

It must be stressed that the progression of these tools reflects the maturity of the area under study, as well as the CT involved. For example, a weather simulator is not created overnight but rather begins with programming. It is argued that the tools naturally rise to the level of abstraction appropriate to the CT of the discipline. In many computational sciences, including those that analyze datasets, it is common to use computational notebooks as a tool for CT (Kaggle, 2017; Shen, 2014). Computational notebooks combine text, mathematical equations, code, and graphs into a single document that can be shared with other scientists. Unlike a traditional report, where the analysis is done elsewhere, computational notebooks contain the code for the analysis and can be run to create the corresponding results and graphs. CT with computational notebooks therefore involves programming (e.g., in programming languages like Python or R) and makes use of scientific software libraries for standard analysis tasks like loading data or inferential statistics. Such scientific software libraries represent an idealized model of solving problems in a scientific domain in much the same way that programming languages represent an idealized model of a computer.

Against this backdrop, a variety of approaches and tools have emerged to make computational thinking more engaging and relevant to students, while also trying to make programming more approachable. Perhaps the largest amount of work has focused on games/simulations and general multimedia platforms (Armoni et al., 2015; Deng et al., 2020;

Kalelioğlu, 2015), and results have broadly shown greater motivation, self-efficacy, and enrollment in future CS courses. Virtually all of these approaches over the last 20 years have used some alternative to traditional text-based programming. A widely used alternative is blocks-based programming, where LEGO-like blocks representing code are combined together to create programs (Bau et al., 2017; Resnick et al., 2009). Blocks-based languages, such as Scratch or Scratch, Jr. are typically graphical interfaces on top of popular general programming languages, and so can be used to learn coding (Zhang & Nouri, 2019). Blocks-based programming simplifies traditional programming in several respects. First, blocks are provided graphically on a palette and can be dragged and dropped onto a workspace to create a program. As a result, students do not need to memorize programming language constructs and recall them; instead, they can search through the list of available blocks until they recognize the block they need. Second, blocks are shaped to fit together only in allowed ways. This property, together with the reduction in typing associated with blocks, greatly reduces the chance of syntax errors that are common when learning to program. Thus blocks-based programming has the potential to enable CT with the same expressivity as traditional programming, but with reduced extrinsic cognitive load (Olney & Fleming, 2019). It is for these reasons that research shows block-based approaches reduce the initial complexity of the coding process to learn, which makes CT more accessible to a wide array of learners (Fagerlund et al., 2021; Moreno-León & Robles, 2016; Popat & Starkey, 2019).

2.3 Learning Experience Design

As emphasis on CT education has grown, the field has developed a number of learning environments to support the development of coding skills. A critical component of how learners engage in computational thinking relates to the design with tools that scaffold learning (Angeli & Giannakos, 2020). This also places a greater emphasis on the human-computer interaction aspect of learning technologies. Recently, theorists describe this as *learning experience design (LXD)*, which details how to consider the learning environment for interaction that supports meaningful learning. Related literature focuses on the importance of interaction and design on engagement in gaming (Annetta et al., 2009; Vann & Tawfik, 2020), immersive realities (Oprean & Balakrishnan, 2020), and other learning environments (Wijekumar, 2021). Others have explored interaction from traditional UX or usability factors, such as course structure in design. Beyond just affective outcomes such as engagement, studies show that poor interaction and UX impacts how learners understand the content as they traverse the learning environment and its perceived usefulness (Carey & Stefaniak, 2018; Wei et al., 2015). Additional studies describe the impact of interaction and interface design on learning outcomes in terms of retention (Novak et al., 2018), self-directed learning, (Kim et al., 2021), and collaborative learning (Lemay et al., 2019; Lewin et al., 2018). Collectively, the empirical studies on LXD conclude that learning through technology extends beyond content to include elements of human-computer interaction.

One of the challenges of the current state of 'learning experience design' is that UX is embedded within existing methods and theories. For example, the UX portion is often considered as a final, summative component of evaluation during the construction of learning technologies (Lu et al., 2022). In terms of theory, users' interaction is considered as a secondary construct within other theories, such as cognitive load theory (Janssen & Kirschner, 2020), unified theory of acceptance and use of technology (UTAUT) (Lemay et al., 2019),

or pedagogical usability (Bakki et al., 2020). As the importance of interaction within learning technologies has grown within the literature, various theorists have attempted to formalize aspects of LXD. For example, Chang & Kuwata (2020) defined LXD as "the practice of designing learning as a human-centered experience leading to a desired goal" (p. 141). Gray (2020) details constructs that especially consider the socio-technical perspective, such as aesthetics, knowledge gains in authentic contexts, empowering learners' unique qualities, and supporting inquiry and action. Toward a more holistic understanding, Jahnke et al., (2020) describe the dimensions of the social (designing to support collaborative interactions), technological (designing for user-friendliness and usability), and pedagogical (designing for learning goals, content organization, activities, and assessment). While the prior researchers are largely theoretical, Tawfik et al. (2022) performed a grounded theory study and described LXD as a confluence of interaction with the learning environment (customization, expectation of content placement, functionality of component parts, interface terms aligned with existing mental models, and navigation) and interaction with the learning space (engagement with the modality of content, dynamic interaction, perceived value of technology feature to support learning, and scaffolding). To better complement the theoretical basis, additional empirical studies are needed to better understand the learning experiences that support learning outcomes across different domains.

3 Research Question

The aforementioned studies suggest that the LXD of digital tools is an important element during the learning process. This may be especially true in STEM domains such as computational thinking where there are multiple elements of an interface that individuals must manipulate during the coding process. While there is a growing call for STEM education, very few studies have looked at the LXD as individuals use tools that support computational thinking. Indeed, in a recent systematic review of the CT literature, Zhang and Nouri (2019) conclude that "human–computer interaction should be considered as being a computational perspective" (p. 15). Although LXD studies are often focused on novice learners, additional research is thus needed to understand how different learners interact with the design affordances and CT supports within these learning environments. Therefore, the purpose of this LXD study was to compare advanced and novice LXD interactions when users were provided a learning tool specifically designed to scaffold their CT. Specifically, the research question to address this gap is as follows:

1. To what degree do more advanced or novice learners perceive their LXD when engaging with tools designed to scaffold their computational thinking?

4 Methodology

4.1 Design-Based Research

To address this research question, the research team performed in-lab LXD testing of a block-based visual programming tool called Blockly, which has been incorporated into a computational notebook called Jupyter Notebook (Olney & Fleming, 2021). As detailed below, the study tasks users with a series of basic programming tasks. The following sections describe the processes used to recruit study participants, perform tests, and analyze data outputs to questions regarding more advanced and novice learners' perceptions about their interactions within the learning space. Specifically, the below describes a design-based research (DBR) study, the goal of which is to iterate through cycles based on specific design goals to move student thinking in the direction of mastery (Anderson & Shattuck, 2012) The application of DBR in this context involves research that is *in situ* or in "naturalistic contexts" (Barab & Squire, 2004, p. 11), which aligns with the use of think-aloud protocols during usability testing.

The need for this study arose from feedback from instructors regarding basic obstacles they witnessed students encountering with the data science learning space deployed during the DataWhys summer internship program Memphis. Using LXD testing, the specific design goal of this study was to uncover what and how aspects of the learning space may need improvement from the perspective of both more advanced and novice learners. The focus of experts versus novice users relates back to a main objective of the DataWhys learning program, which is to teach data science to learners who do not necessarily have much experience with computational thinking practices. Identification of interaction patterns between these two user groups will help determine future design iterations of the learning space, including decisions around the degree of scaffolding and other potential LXD modifications to support learning.

4.2 Participants

Ten college students were recruited from a large research university located in the mid-south portion of the United States. Five students, later referred to as participants P1-P5 in the Results section, were majoring in education and considered novices in using computer science tools, including computational notebooks and other programming tools. The other five students, later referred to as participants P6-P10 in the Results section, had taken college courses in computer programming and had varying degrees of experience using computational notebooks. Given the research gap and questions, no additional participant demographic information was taken for this study.

While researchers have debated the optimal number of participants to utilize in qualitative studies, a number of researchers (Budiu, 2021; Nielsen, 2000; Whitenton, 2019) posit that for qualitative usability studies, adding more users to the same test often does not result in considerable new findings and UX issues. Nielsen (2000) explains that researchers learn approximately one-third of what there is to know about the usability of a design from the first test user. After the fifth user, he argues that very little new knowledge is learned about the user's interaction patterns (Nielsen, 2000). Coinciding with this research, the sample size for this study was 10 participants (5 novice learners and 5 advanced learners).

4.3 Procedure

Tests were held in-person using a concurrent think-aloud protocol in which participants say aloud their perceptions as they interact with the software to perform assigned tasks. As opposed to post-hoc reflection, the goal is for learners to verbally articulate their response as to more accurately capture their real-time interaction perceptions. In doing so, this methodological approach is able to identify user perceptions, along with potential usability issues that may be an issue for effective use of technology (Fan et al., 2020). Concurrent think-aloud protocol is therefore commonly used in usability testing to gain real-time feedback from users on an interface (Boren & Ramey, 2000; Fan et al., 2019; Reinhart et al., 2022).

After receiving approval from the Internal Review Board, the facilitator presented a series of on-screen slides. The first few slides welcomed the participants and provided a conceptual introduction to the software and its intended uses. Next, the facilitator explained the think-aloud protocol, encouraged participants to ask questions as needed, and reminded them that they and their computer actions would be recorded on the computer using Zoom. As per the methodology, they were encouraged to think aloud as they read, clicked, hovered, dragged, etc., and became acquainted with the interface. After the initial introduction, the remaining think-aloud tasks were as follows:

1. Print “hello” 5 times.
2. Create a block and convert it to code, thinking aloud as you perform each step.
3. Edit your code in the Jupyter notebook without using blocks.
4. Explore two ways to copy and paste into different cells.

Upon completion of the user interface tasks, participants were asked post-hoc interview questions related to their experience. This included questions about how the Blockly and Jupyter tools worked together, if elements and labeling were clear, and if participants had any suggestions for opportunities to improve the learning environment.

5 Materials

5.1 Learning Environment

The software used in the study was the computational notebook JupyterLab with a Blockly extension for blocks-based programming. JupyterLab is an open-source computational notebook that supports dozens of languages; however, only the Python programming language was used in this study. JupyterLab runs in a web browser and presents a computational notebook as a list of *cells* (see Fig. 1, right-hand side). Each cell may contain code or text in a simplified markup language called MarkDown (cf. HTML). As cells are executed, they display the results of running the code in the cell output area immediately below the cell. Cells can be cut/copy/pasted, rearranged, and otherwise manipulated using the normal conventions of a text editor. The Blockly extension (Olney & Fleming, 2021) integrates Google’s Blockly library for blocks-based programming into JupyterLab (see Fig. 1, left-hand side), with the additional feature of *Intelliblocks*, which are blocks that are automatically generated whenever a software library is loaded. Individual blocks are nested under

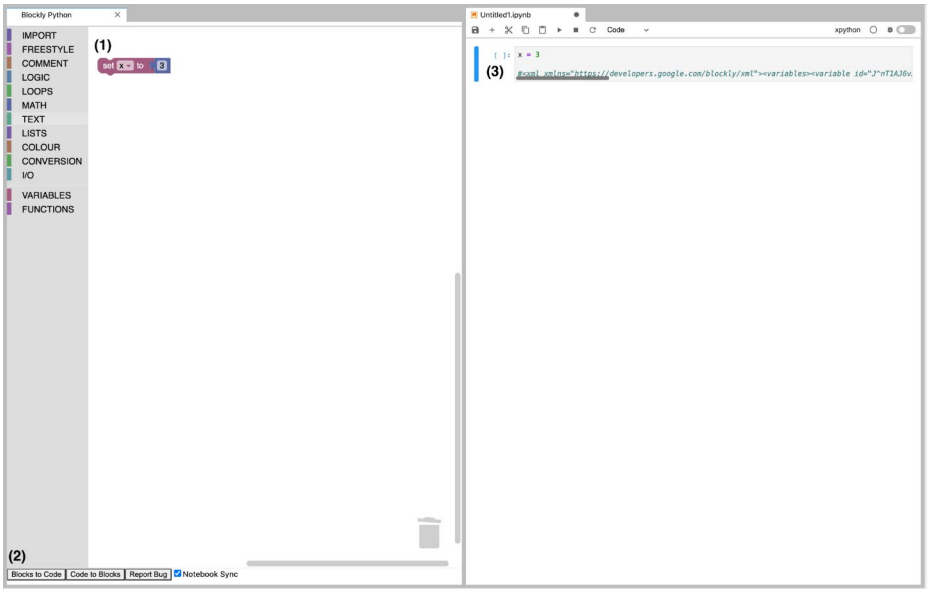

Fig. 1 Jupyter Notebook with Blockly Extension. *Note.* (1) Student combines blocks using Blockly. (2) Student presses "Blocks to Code." (3) Python code corresponding to the blocks appears in the selected Jupyter cell

categories (e.g., IMPORT) and appear in a pop-out menu when the category label is clicked. From there, they may be selected and dropped onto the workspace where they may be combined with other blocks. Notably, the FREESTYLE block category contains blank blocks directly into which Python code can be written. The multiple forms of CT representations and interactions may thus serve as a way to fade scaffolding within the learning environment (see Fig. 1).

In the study, the Blockly extension and corresponding notebook were presented side by side within the JupyterLab interface. As shown in Fig. 1, blocks that were created on the Blockly side were converted into corresponding Python code in the active cell on the notebook side when users pressed the "Blocks to Code" button on the lower left-hand side. Python code generated from blocks always contained an XML comment at the bottom of the code cell (i.e., starting with #). This XML data was used to reconstruct the blocks in the workspace if the user navigated to another code cell. In this way, both the Blockly and notebook sides remained in sync with each other.

5.2 Data Coding

The goal of this study was to explore the ease of use of the Blockly interface when combined with a Jupyter notebook and to gather data on user interactions and perceptions of the software environment as a tool for learning how to code. After all tests were completed, the facilitator and primary research assistant transcribed the audio files from the recorded sessions using Otter, a transcription software. Next, the primary research assistant entered participants' transcribed audio into a spreadsheet and created line items based on the most

Table 1 Interaction with the Learning Space Codes and Definitions

Interaction with the Learning Space	Definitions
Engagement with modality of content	Thoughts on learning design format, aesthetics, and learner's desire to engage with elements used on the learning interface.
Dynamic interaction	Interaction that engenders learners' desire to continue or discontinue in their self-directed learning
Perceived value of technology feature to support learning	How learners view the utility of a technology feature to aide in instruction and complete a learning task
Scaffolding	Cues, hints, etc. that expand upon learners' prior knowledge

detailed level of each step within the corresponding task. Upon completion of the interviews, the researchers conducted a thematic analysis of line items with the Tawfik et al. (2022) framework serving as a priori codes. The authors contend that LXD falls within two broad constructs: interaction with the learning *environment* and interaction with the learning *space*. Given the work-in-progress nature of the study, the researchers especially wanted to focus on the latter, which explored the specific learning activities given the available features of the CT tool. The specific codes used to determine interaction with the learning space consisted of the following: engagement with modality of content, dynamic interaction, perceived value of technology feature to support learning, and scaffolding (see Table 1 for definitions).

Two research assistants individually coded each line item in a spreadsheet, with neither research assistant having visibility into the other person's codes. Once each research assistant completed his/her individual coding, they met to compare their codes, discussed justification for assigned codes, and came to an agreement regarding the most accurate codes. The two research assistants completed a second round of analysis based on the four 'Interaction with the Learning Space' codes and reached an agreement on 55% of the line items. After negotiating the discrepancies, the inter-rater reliability increased to 98%. A senior member of the research team reviewed the remaining 2% and determined a code to best fit each based on the team's definitions.

6 Results

6.1 Interaction Within the Learning Space

6.1.1 Code 1: Engagement with Modality of Content

In general, both novices (i.e., P1-P5) and more advanced users (i.e., P6-P10) described a positive experience with the construct of engagement with modality of content. For example, P2 noted *"I have not really done a lot with coding, maybe just the scratch here and there. And I felt very comfortable*." Similarly, P3 answered, "*They're pretty easy; at first I was a little confused, but they were pretty straight-forward."* When asked why the learning environment supported their computational thinking, novices shared that format and other design features helped their engagement with the learning materials. For example, P3

mentioned that *"It was a little bit more creative and I like how there were different colors"*, while P4 expressed that: *"It was fun. It was really cool how you can do different types of blocks to create different types of codes. I've never coded before. It was really fun*." One novice (P4) especially highlighted the following unique visual component of the modality: "*It's like a visual aid and then like the written down code, so it probably helps people [to] like double check their work or focus more."* In this case, the modality of CT content was supported through the use of colors and blocks as visual aids.

In addition to the colors and blocks as visual aids, novices also described how the blocks were especially beneficial to understand computational thinking. P3, a novice learner, elaborated that:

> *"I think the overall layout is good. I guess just because I'm not really familiar with how to code and stuff, I was kind of confused, but I think it's really good. I really like how you put the blocks together and convert it to the cell, and I understand how to run the cell".*

Because novices expressed engagement with how the components were designed within the interface, they underscored how the learning space encouraged them to engage in and explore other elements of computational thinking. For example, P1 said,

> *"The only thing I had confusion about was going from the basic starting out to getting from…which blocks did it put in there to figuring out what to do on all the tasks and everything else. And then going from, at firststarting to run the code whenever you actually put it in there."*

As another example, a novice user (P3) mentioned how:

> *"Trying to figure out what this does on the right. I know there's like a URL. (Referring to the code that automatically appears in the code cell and is used to convert code back to blocks.) 'Import some library as variable name' from some input variable name. So you could type A, like so freestyle is pretty much what you could type so you can put your own codes and math has like sin, square roots. I guess you can add your own numbers in it=text, color, color with red 100, green 50, blue zero. We could change the colors"*

The above data from the think-aloud responses describes how participants had a desire to engage with the interface and largely expressed positive feelings about the learning format, especially the role of color coding during CT.

The more advanced users - those with experience in computer science - also expressed favorable perceptions about the modality of the data science content. When discussing its potential use, P6 mentioned that:

> *"I thought it was really a really cool way to show how the different aspects of code are linked together. And I thought it was also a good way to introduce the concept because everyone's used to playing with blocks and puzzle pieces. And it's a good way to conceptualize programming."*

Another advanced user, P8, commented positively in saying:

> *"I was really impressed with it. Yes, I've seen it used, but I haven't used it myself. And I was just kind of skeptical about it, but now actually going through it…it does actually seem really intuitive and clear".*

As in the case of the novice learners, the quotes suggest that the visual representation of the blocks was perceived positively when doing their computational thinking task for this group of learners. When asked about what specifically made the interface engaging for learning, P8 referenced the modality about seeing the code in multiple formats rather than just different colors:

> *"I like being able to see that code, like right away, being able to copy and paste the code once it's made with the blocks. And also even when they click on the code, you can see it in the blockly form. Like I thought it was excellent. It's really great".*

Similarly, another advanced user, P10, said,

> *"Honestly, I feel like it's user friendly, at least to me. You know, I don't know what age people will be working with this and things like that, but I felt like everything was a smooth interface. It made sense what I needed to be doing. I like the layout where you can have the two screens, you know, right next to each other with still clear resolution. [It was] an easy interface between the two tabs, you know, all of that felt very smooth to me".*

In line with novices, the above data suggests learners in the more advanced users referenced the visual representation of the blocks, and also commented as to how the dual nature of the visualization supported the CT tasks.

6.1.2 Code 2: Dynamic Interaction

Dynamic interaction describes the reciprocal relationship between the learner and technology, with a special emphasis on how the interface progresses the learner towards new tasks. Although novices were initially clear on how to interpret the interface, they often expressed confusion as to the next step in their computational thinking. For example, when trying to perform an action on the print block, P4 noted that

> *"Okay, I clicked the print block. I see that the print is in red and that the loop of action is in green. I also see that the five is in parenthesis and the word 'hello' is in parenthesis, so maybe that indicates that only those things? Like how many times and then what word is being said?"*

.In this instance, the user alludes to a disconnect between the blocks and what actions would be performed by the code, which impedes their perception of the subsequent task. When learners did encounter challenges, others highlighted confusion about the outcomes

and ambiguity as to next interactions. When interacting with the blocks, P6 described the following:

> *"I wonder if I could pull it over. Nope, that didn't work. I wonder if I could copy here and paste it over there. Nope. I see something that says 'code to blocks'. I'm going to select this one, then select 'code to blocks' and then click 'play.' Let's try another one and do the same. I clicked 'code to blocks' and had that one selected. That doesn't work. So I have to be inside of this cell. I clicked 'code to blocks' and clicked 'play' and that did not work."*

As the participant explained, they tested out various elements, but the system response and next steps remained unclear, which served as a disruption to their dynamic interaction.

In contrast to novices that struggled with this construct, more advanced learners expressed a desire to test the system when the system did not respond as expected. P6 said the following:

> *"I/O is input/output I think. So, can I print the colors? I don't really know what to do with the colors. Random color. What if we run that? What does that do? Ah, we get an error 'Color blend is not defined'. Okay, but it also got rid of all my blocks in here, which is a little annoying".*

Rather than stop short in their interaction, P6 went on to say,

> *"I really want to figure out this color stuff because it's weird. Okay, let's put that in there, and let's pull up a freestyle thing here. And what if we do 'Print X' and we do variables 'Create variable x' and we do set x to 123? So if we do 'blocks to code' now and then we do 'run', is there a print block?"*

Despite no explicit indication of the next CT task, the user tested out different elements; that is, they were able to generate and describe multiple potential interactions that would advance the task. In another example, P6 described how he was able to understand the loop patterns over time:

> *"So we 'Control C,' this loop. And we put it in this cell that it's already in. 'Control V.' We still get the 'print Paul' from when I copied it earlier. Can I do this? I wonder what if I 'Control C' that and I tried to paste it in here. Did that work? If I copy this, paste it? Okay. It seems like a 'Ctrl C' 'Ctrl. V' 'copy paste' is determined by whichever window I'm clicked in. Yeah. So I can 'Control V' here all day long. And I will just keep duplicating this loop. And I can 'Control V' here all day long".*

In doing so, a key differentiator between novices and more advanced users is that the latter engaged in iterative testing rather than waiting for the system to prompt the next interaction.

6.1.3 Code 3: Perceived Value of Technology Features to Support Learning

Because of the design of the system, it was important that users understand how specific embedded technology features supported learning as they engaged in computational thinking. Novice user P3 commented that coding was initially a challenge, whereas the blocks were seen as beneficial for their learning: "*I feel like it [CT] would be a lot more difficult, so this [the blocks] makes it easier*." P1 commented that they were able to understand how the features helped them code faster, commenting as follows:

> "*That way, you know what's going on. You could just go ahead and do it and be done. But if you actually want to have a little help, that way you don't have to type too much. The blocks are probably good."*

In another instance, P1 remarked that,

> "*I know it's kind of fun to have it be a puzzle piece. Well, it was good, because you know, once you put it in one side and you want to do something else, you can just click a button like 'blocks to code' and it puts it in there for you."*

Collectively, the results of the study showed that novice users perceived the Blockly-Jupyter interface to be a helpful tool in learning to code. Some novice users preferred to use the visual blocks to perform the tasks within the software, whereas others voiced their preference for writing code directly into the Jupyter notebook cells. However, all novice users perceived that the Blockly-Jupyter interface was a valuable tool that aided the computational tasks.

More experienced users expressed a somewhat different sentiment when describing interaction with the available tools within the learning environment. Not only did users comment as to the systems' ability to support their initial learning, but also how the interface supported more advanced computational thinking tasks. For example, when comparing blocks to standard coding, P9 said the following:

> *"And all of the different functions, as well as the sort of shapes of the blocks and things, like it makes it very intuitive where you would do sort of an indented code or something like for a for loop".*

When reflecting on the overall process, P7 mentioned,

> "*It's more efficient when you're not in the programming domain, so they can just easily do a lot of things within a short time. Sometimes they're having issues of how to write the code or how to import, so they can easily import from the Blocky palette."*

Another advanced user had a similar response to using blocks to create code compared to not using blocks. P8 remarked that the features would be especially helpful with novel CT that he was not familiar with:

> "*I liked them. I would use them for doing things that like I'm not familiar with. Also, just to kind of create like these larger, more universal codes that I could repeat using. But on some level, I also just felt like I could type faster. But I don't think so for, like, more complicated code. I think the blocks help for trying to figure out what to do next. But then sometimes just dealing with, like, my specific variable names and everything. I think just typing it out might be easier."*

In this instance, they were able to parse out when they would use the features and for specific CT purposes, especially for novel tasks and simpler tasks. Another example of this was when P9 commented,

> *"Like I said, I make quite a few errors when it comes to punctuation and spaces and indentations and stuff. So I think I prefer blocks for those basic kinds of things. Maybe in a more complicated problem, I might prefer code, but….as far as the basics, I think the blocks work a little nicer. The code, just writing it out feels a little more familiar, fluent".*

6.1.4 Code 4: Scaffolding

Although the learners highlighted some of the benefits in terms of the representation, they underscored some of the remaining challenges in terms of learning computational thinking skills and the need for additional support. P5 pointed that out by saying, "*I think the instructions were fine. They were quite clear, and I appreciated when I did get stuck. That there were some helpful hints, so there's some scaffolding along the way*." Similarly, when asked if the elements of the pages they interacted with were intuitive, if the text labeling was clear, and if the blocks were clear, P3 replied, *"Yeah, I think they were pretty clear. I understood most of it where the math was and the text. But then again, there were others that I was kind of confused about."* In this instance, the learners especially appreciated scaffolding for initial CT related actions, although they indicated the need for support in other areas of computational thinking.

As the data collection progressed, novice users expressed a desire for more scaffolding as to how to perform certain tasks. In their think-aloud, P2 questioned, *"Okay, so I have 'set to' and I need to connect it to a 3? I don't know if that's right. That seems right."* In this case, the user was unclear if s/he had completed the action successfully and desired scaffolds in the form of immediate feedback that would have supported their learner. In another example, one student gave up on putting two blocks together because the shape of the blocks appeared as though they would not connect. This resulted in the student thinking he did not know which blocks to use, when he actually had performed the correct task. P3 similarly shared their frustration with the CT task as follows:

> *"Still trying to figure out how these work together. Okay, blank color one. Blend, blend red with purple ratio. Okay, I'm gonna start connecting. Make a big one and put them all together. Okay, so if I can find something that could go into this one. I'm just looking for something that could go on the inside of this green one. We see the shape of it. Okay, so this could fit in it? Oh, no, it does not."*

There were additional indicators that the software was not always intuitive for scaffolding beginner users. When asked the same question, P5 had a more pointed response when she said:

> *"No. So I would say that nothing was where I expected it to be, except for the menu. So the menu was helpful, maybe even if I hover over the menu, [but] it would be great to have some type of text. For example, 'you choose this to add text to your page'. Something that kind of helps as an additional feature to help me understand what I need to do. That made it very difficult just for me to understand...just coming in as a person who has not used code before or just as a person who hasn't used the system. I guess it's just difficult not knowing. When I first get on there, it's a blank page."*

More advanced users described differing perspectives of their scaffolding experiences. In many cases, they especially identified how colors clearly aligned with specific interactions. For example, one user indicated that, while the Blockly palette itself was fairly clear, there may be a need for more scaffolding when copying from Blockly to Jupyter. P6 described the following:

> "*I explored the Blockly palette space pretty thoroughly. I believe, at least half of it, I went pretty in depth with the individual blocks that I had. So I wasn't really confused on what to do or where to go to access things."*

In terms of how the interface elements specifically served to scaffold a computational thinking task, P6 similarly expressed that:

> "*I thought that any explanation of the block that was given….I could highlight over them. I didn't right click and then click on help for any of them. I probably should have. But the explanations that were given when I highlighted over a few of them were pretty easily understandable. And they aligned with what I already thought the blocks meant. So that was good."*

In contrast with the novice experience, the more advanced users did not need more explicit scaffolds and feedback; rather, they were able to reference the support structure on an as-needed basis. When asked about how the Blockly and Jupyter Notebook tools worked together, P9 suggested that:

> *"I think that's a good way to sort of be able to focus on what matters with programming. Like being able to see that this is how it produces, say, that loop without worrying about there's some little quirks, like with using a range for a loop or something. There's some of these little quirks that you don't really have to worry about as much because he could use the block to get the basic idea of that, and then you get the code next to it to actually see how it's performed. So I think that worked really well together.*

Whereas novices needing prompting and immediate feedback during coding, the more advanced users indicated that the support structures were sufficient through colors and blocks, which allowed them to reference the scaffolds on an as-needed basis.

7 Discussion

Computational thinking is increasing in the learning domain as technology becomes ubiquitous in society. Although educators cite a desire to implement these skills, the research shows that this is often difficult to do for various reasons. To date, research has been especially focused on the challenges from a cognitive processing perspective and how to best engage in learning strategies related to paired programming (Umapathy & Ritzhaupt, 2017), worked examples (Qian & Lehman, 2017), and others. While these have added towards the advancement of CT competencies, the data of this work-in-progress study above argues that another critical factor includes the design of these systems and the interaction that supports computational thinking tasks. Indeed, a recent systematic literature review on the UX aspect of CT found only 22 studies and concluded that "the existing literature on the usability and effectiveness of learning tools for programming is truly scarce" and "there is a significant gap both when it comes to designing and developing tools that encourage computational thinking and are suited to different educational levels" (Rijo-García et al., 2022, p. 8). Although CT research is often focused on gateways to CT (Perera et al., 2021a; Price & Barnes, 2015), this research extends understanding of LXD for CT as it explores different interaction patterns among both novices and more advanced learners. Based on comments by both novice and advanced users, learners appreciated the intuitive nature of the interface to varying degrees, including the categorical color coding and labeling of the blocks. Both learner types reacted positively to how the two sides of the screen, Blockly and Jupyter, functioned in tandem with the exception of the Code to Blocks functionality, which led to confusion among both groups of learners. In that sense, it seems as though the tool is beneficial for the engagement with the 'modality of content' construct of LXD. That is, the individuals noted how the modality that included multiple forms of visualization was beneficial to their computational thinking.

Empirical literature has explored the user of visual and block-based programming and found various benefits, such as a perceived ability to compose (Weintrop & Wilensky, 2015), more efficient time on task (Price & Barnes, 2015), and motivation (Sáez-López et al., 2020). This study builds on prior research by exploring interaction patterns of different user groups as part of their LXD. While both groups highlighted the benefits of the modality for the CT tasks, differences emerged in their subsequent interactions and expectations when using the learning environment. Novice learners mostly looked to the visual blocks as a tool for learning to code, but more advanced users examined the blocks from a technical and programming lens, comparing them to how they typically approach coding already, which is via the text-based approach in the computational notebooks (i.e., Jupyter notebooks) . More advanced and novices differed in their dynamic interaction with the interface, especially when reaching an impasse during the task. Those with CT experience showed a tendency to test and use the design elements of the tool, along with previous knowledge of coding and familiarity with other coding tools, to continue in self-directed learning and explore various possible solutions paths. In that sense, they were able to complete the task even if the learning environment was not immediately intuitive. On the other hand, novices struggled to identify alternative CT approaches and had a propensity to discontinue self-directed learning when they were unable to work through a problem after their first or second attempt. Rather than allowing learners full autonomy during interaction, one LXD implication for this work-in-progress study is how to make the interactions of these systems transparent

in terms of (a) the next learning task that align's with Wing's (2011) phases of CT and (b) overcoming unexpected outcomes that occurred during the learning task.

The data additionally revealed specific features that supported the scaffolding construct of LXD. In many ways, the study aligns with prior literature that notes the benefits of a block-based approach and its ability to make aspects of CT more salient to novices (Weintrop & Wilensky, 2015; Xu et al., 2019). Both groups valued the hover text that accompanied the blocks, although in different ways. The more advanced users naturally viewed the help text as a connection or different way of visualizing that which aligned with their prior knowledge. Alternatively, the novices lacked this knowledge base, so more time was spent trying to understand the functionality of the blocks and the meaning of the associated text. That said, most found the labeling for the basic block functionality (e.g., text, print) initially clear. For learners who explored the graphical icons within the tool, most benefited from the associated hover text, with experts more often referencing previous knowledge of the tool affordances and novices sometimes expressing confusion of specific features. For example, this may stem from experts having previous knowledge of using the 'play' button to run or execute code, whereas novices did not have this connection to make when asked to "run the code." As this design-based research project builds on this work-in-progress study, it is important to determine LXD strategies that allow learners to move forward in their learning interactions, especially when during an impasse.

8 Limitations and Future Research

While this qualitative research study leveraged the use of concurrent think-aloud protocols and retrospective follow-up questions to understand learners' LXD, additional methods of data sourcing could be employed in future studies. For example, gathering participant eye movements through eye tracking software or interaction analytics within the software would produce valuable data, especially for the constructs of dynamic interaction and scaffolding. In terms of the former, it would allow researchers to determine learners' gaze as they look to progress in their CT tasks. In terms of scaffolding, this type of data would help determine the 'when' and 'where' of learners' help-seeking behavior, which would allow the team to further iterate the learning envrinment. Triangulating this data with participant think-aloud transcripts would thus provide even more validity to the data presented.

Another study could explore expanding beyond the participants presented in this study. For this research, participants were gathered primarily from two programs - computer science (advanced learners) and education (novices) majors - within the institution. A future approach to further analyze the learning experience within the Jupyter-Blockly tool would be to expand the research to different schools, college majors, or even contexts outside of a university setting, such as an industry setting. Other research shows that CT is increasingly emphasized in primary and secondary school (Zhang & Nouri, 2019), so future studies could also explore LXD across different age groups. A better understaning of LXD for these learners may may help educators and researchers develop learning systems that align with their respective stages of cognitive development.

This study was performed in a one-on-one setting in a large office room on campus using synchonours conferencing technology. In addition, the study could be extended to include other types of user experience studies (e.g., in-person classroom observations/stud-

ies and in-person focus groups). While some research (Budiu, 2021; Nielsen, 2000; Whitenton, 2019) suggests that five participants is enough for usability testing, other researchers (Alroobaea & Mayhew, 2014; Faulkner, 2003) argue that more participants may be needed for studies depending on the scale of the study. It is possible that additional participants from each user group may have elucidated additional interaction patterns that were important for their computational thinking and LXD.

The project team captured participants' interpretation of a novel learning environment. However, in a "real-world" environment, instructional designers would provide didactic lessons, along with supportive materials (e.g., video demonstrations, note cards showing graphical block commands, etc.) to scaffold learners through coding projects. Furthermore, directions assigned during think-aloud study sessions were fairly basic, entry-level CT tasks. It would be interesting to see both novice and expert learners' perceptions of the learning environment when assigned more advanced tasks and how their perceptions might change over time as interactions become increasingly complex.

9 Conclusion

The think-aloud study described in this manuscript was conducted to begin to fill a research gap pertaining to the study of design and development of computational thinking tools. The researchers involved in this work-in-progress study tested participant perceptions of the LXD of a block-based coding tool, which employs CT methods and was created to aid in teaching data science to adult learners. Transcription data gathered from student participants – half who had experience with CT programming tools and half who did not – highlighted the differences in perceptions and design interactions between the two groups. In terms of agreement, both learner types seemed to generally agree on *engagement with modality of content* and *perceived value of technology feature to support learning.* In doing so, this builds on prior literature which suggests that block-based coding tools can be an effective way to support the development of CT skills (Perera et al., 2021b). Alternatively, the most significant differences among novices and advanced users were in the constructs of *dynamic interaction* and *scaffolding*.

The interaction patterns among the different learner profiles may elucidate important insights as educators employ learning environments that facilitate CT. In terms of *dynamic interaction*, novices struggled to progress to the next learning task, indicating a possible need for more support. Indeed, CT skills include varying skill sets that are dependent on one another. At any given time, learners may need to engage in problem reformulation, recursion, problem decomposition, abstraction, or systematic testing (Wing, 2011). To better support dynamic interaction, it may be that learning environments not only outline the specific coding actions, but explicitly progress the learner through these specific stages of CT. On the other hand, more advanced users showed engagement in self-directed learning, as they tended to troubleshoot issues within the system until uncovering a solution. In the area of scaffolding, novices expressed the need for a more task-based approach to coding and commented on the immediacy of the feedback. As future learning environments build on this work-in-progress study, the data suggests design features that provide additional support as to how to complete the technical aspects of CT and assessment as to whether they were able complete the task successfully.

An inference made from this study is the need for transparency in regard to the interactions within the learning system, including (a) what the next task in the learning process is and (b) how to overcome unexpected outcomes that occur during learning tasks. While this study employed a 5-test users (per group) approach, future studies may benefit from a higher number of participants. Other ideas to further this research include testing in a more "real-world" environment that includes (a) additional supports (i.e., didactic lessons, video demonstrations, note cards, etc.), (b) assigning participants more complex CT tasks, and (c) using eye-tracking software during testing.

Funding This material is based upon work supported by the National Science Foundation under Grant No. 1918751. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Statements and Declarations

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

References

Alroobaea, R., & Mayhew, P. J. (2014). How many participants are really enough for usability studies? *2014 Science and Information Conference*, 48–56. https://doi.org/10.1109/SAI.2014.6918171

Anderson, T., & Shattuck, J. (2012). Design-based research a decade of progress in education research? *Educational Researcher*, *41*(1), 16–25. http://edr.sagepub.com/content/41/1/16.short.

Angeli, C., & Giannakos, M. (2020). Computational thinking education: issues and challenges. *Computers in Human Behavior*, *105*, 106185. https://doi.org/10.1016/j.chb.2019.106185.

Annetta, L. A., Minogue, J., Holmes, S. Y., & Cheng, M. T. (2009). Investigating the impact of video games on high school students' engagement and learning about genetics. *Computers & Education*, *53*(1), 74–85. https://doi.org/10.1016/j.compedu.2008.12.020.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions in Computing Education*, *14*(4), 1–15. https://doi.org/10.1145/2677087.

Bakki, A., Oubahssi, L., George, S., & Cherkaoui, C. (2020). A model and tool to support pedagogical scenario building for connectivist MOOC. *Technology Knowledge and Learning*, *25*, 899–927. https://doi.org/10.1007/s10758-020-09444-8.

Barab, S., & Squire, K. (2004). Design-based research: putting a stake in the ground. *The Journal of the Learning Sciences*, *13*(1), 1–14. http://www.jstor.org/stable/1466930.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the ACM*, *60*(6), 72–80. https://doi.org/10.1145/3015455.

Boren, T., & Ramey, J. (2000). Thinking aloud: reconciling theory and practice. *IEEE Transactions on Professional Communication*, *43*(3), 261–278. https://doi.org/10.1109/47.867942.

Budiu, R. (2021, July 11). *Why 5 participants are okay in a qualitative study, but not in a quantitative one*. Nielsen Norman Group. https://www.nngroup.com/articles/5-test-users-qual-quant/

Carey, K. L., & Stefaniak, J. E. (2018). An exploration of the utility of digital badging in higher education settings. *Educational Technology Research and Development*, *66*(5), 1211–1229. https://doi.org/10.1007/s11423-018-9602-1.

Chang, Y. K., & Kuwata, J. (2020). Learning experience design: Challenges for novice designers. In M. Schmidt, A. A. Tawfik, I. Jahnke, & Y. Earnshaw (Eds.), *Learner and user experience research: An introduction for the field of learning design & technology*. EdTechBooks. https://edtechbooks.org/ux/LXD_challenges

Chaturvedi, D. K. (2017). *Modeling and simulation of systems using MATLAB® and Simulink®*. CRC press.

Computer Science Teachers Association. (2017). *CSTA K-12 Computer Science*. Computer Science Teachers Association. http://www.csteachers.org/standards.

Deng, W., Pi, Z., Lei, W., Zhou, Q., & Zhang, W. (2020). Pencil Code improves learners' computational thinking and computer learning attitude. *Computer Applications in Engineering Education*, *28*(1), 90–104. https://doi.org/10.1002/cae.22177.

desRivieres, J., & Wiegand, J. (2004). Eclipse: a platform for integrating development tools. *IBM Systems Journal*, *43*(2), 371–383. https://doi.org/10.1147/sj.432.0371.

Du Boulay, B., O'shea, T., & Monk, J. (1999). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Human-Computer Studies*, *51*(2), 265–277. https://doi.org/10.1006/ijhc.1981.0309.

El-Masri, M., & Tarhini, A. (2017). Factors affecting the adoption of e-learning systems in Qatar and USA: extending the Unified Theory of Acceptance and Use of Technology 2 (UTAUT2). *Educational Technology Research and Development*, *65*(3), 743–763. https://doi.org/10.1007/s11423-016-9508-8.

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with scratch in primary schools: a systematic review. *Computer Applications in Engineering Education*, *29*(1), 12–28. https://doi.org/10.1002/cae.22255.

Fan, M., Lin, J., Chung, C., & Truong, K. N. (2019). Concurrent think-aloud verbalizations and usability oroblems. *ACM Transactions on Computer-Human Interaction*, *26*(5), 1–35. https://doi.org/10.1145/3325281.

Fan, M., Shi, S., & Truong, K. N. (2020). Practices and challenges of using think-aloud protocols in industry: an international survey. *Journal of Usability Studies*, *15*(2), 85–102. http://uxpajournal.org/practices-challenges-think-aloud-protocols-survey/.

Faulkner, L. (2003). Beyond the five-user assumption: benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers: A Journal of the Psychonomic Society, Inc*, *35*(3), 379–383. https://doi.org/10.3758/bf03195514

Gray, C. (2020). Paradigms of knowledge production in human-computer interaction: Towards a framing for learner experience (lx) design. In M. Schmidt, A. A. Tawfik, I. Jahnke, & Y. Earnshaw (Eds.), *Learner and user experience research: An introduction for the field of learning design & technology*. EdTechBooks. https://edtechbooks.org/ux/paradigms_in_hci

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: suggestions based on a review of the literature. *Computers & Education*, *126*, 296–310.

Jahnke, I., Schmidt, M., Pham, M., & Singh, K. (2020). Sociotechnical-pedagogical usability for designing and evaluating learner experience in technology-enhanced environments. In M. Schmidt, A. A. Tawfik, I. Jahnke, & Y. Earnshaw (Eds.), *Learner and user experience research*. EdTechBooks. https://edtechbooks.org/ux/sociotechnical_pedagogical_usability

Janssen, J., & Kirschner, P. A. (2020). Applying collaborative cognitive load theory to computer-supported collaborative learning: towards a research agenda. *Educational Technology Research and Development*, *68*(2), 783–805. https://doi.org/10.1007/s11423-019-09729-5.

K-12 Computer Science Framework Steering Committee (2016). *K-12 Computer Science Framework*. K-12 Computer Science Framework. http://www.k12cs.org

Kaggle (2017). *The state of ML and data science 2017*. Kaggle. https://www.kaggle.com/surveys/2017

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, *52*, 200–210. https://doi.org/10.1016/j.chb.2015.05.047.

Kim, D., Jung, E., Yoon, M., Chang, Y., Park, S., Kim, D., & Demir, F. (2021). Exploring the structural relationships between course design factors, learner commitment, self-directed learning, and intentions for further learning in a self-paced MOOC. *Computers & Education*, *166*, 104171. https://doi.org/10.1016/j.compedu.2021.104171.

Lemay, D. J., Doleck, T., & Bazelais, P. (2019). Context and technology use: Opportunities and challenges of the situated perspective in technology acceptance research: Context and technology use. *British Journal of Educational Technology*, *50*(5), 2450–2465. https://doi.org/10.1111/bjet.12859.

Lewin, C., Cranmer, S., & McNicol, S. (2018). Developing digital pedagogy through learning design: an activity theory perspective. *British Journal of Educational Technology*, *49*(6), 1131–1144. https://doi.org/10.1111/bjet.12705.

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM education. *Journal for STEM Educ Res 3*, 147–166. https://doi.org/10.1007/s41979-020-00044-w.

Lu, J., Schmidt, M., Lee, M., & Huang, R. (2022). Usability research in educational technology: a state-of-the-art systematic review. *Educational Technology Research and Development: ETR & D*. https://doi.org/10.1007/s11423-022-10152-6

Moreno-León, J., & Robles, G. (2016). Code to learn with Scratch? A systematic literature review. *2016 IEEE Global Engineering Education Conference (EDUCON)*, 150–156. https://doi.org/10.1109/EDUCON.2016.7474546

National Research Council. (2010). Report of a workshop on the scope and nature of computational thinking. *National Research Council*. https://doi.org/10.17226/12840.

National Research Council. (2011). Report of a workshop on the pedagogical aspects of computational thinking. *National Research Council*. https://doi.org/10.17226/13170.

Nielsen, J. (2000, March 18). *Why you only need to test with 5 users*. Nielsen Norman Group. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Novak, E., Daday, J., & McDaniel, K. (2018). Assessing intrinsic and extraneous cognitive complexity of e-textbook learning. *Interacting with Computers*, *30*(2), 150–161. https://doi.org/10.1093/iwc/iwy001.

Olney, A. M., & Fleming, S. D. (2019). A Cognitive Load Perspective on the Design of Blocks Languages for Data Science. 2019 IEEE Blocks and Beyond Workshop, 95–97. https://doi.org/10.1109/BB48857.2019.8941224

Olney, A. M., & Fleming, S. D. (2021). JupyterLab Extensions for Blocks Programming, Self-Explanations, and HTML Injection. In T. W. Price & S. San Pedro, Joint Proceedings of the Workshops at the 14th International Conference on Educational Data Mining, Vol. 3051, CSEDM–8. CEUR-WS.org.

Oprean, D., & Balakrishnan, B. (2020). From engagement to user experience: A theoretical perspective towards immersive learning.Learner and User Experience Research. https://edtechbooks.org/ux/10_from_engagement_t

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. Basic Books.

Perera, P., Tennakoon, G., Ahangama, S., Panditharathna, R., & Chathuranga, B. (2021a). A systematic mapping of Introductory Programming Languages for Novice Learners. *Ieee Access : Practical Innovations, Open Solutions*, *9*, 88121–88136. https://doi.org/10.1109/ACCESS.2021.3089560.

Perera, P., Tennakoon, G., Ahangama, S., Panditharathna, R., & Chathuranga, B. (2021b). A Systematic Review of Introductory Programming Languages for Novice Learners.IEEE Access. https://ieeexplore.ieee.org/abstract/document/9455382/

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365–376. https://doi.org/10.1016/j.compedu.2018.10.005.

Powers, J. G., Klemp, J. B., Skamarock, W. C., Davis, C. A., Dudhia, J., Gill, D. O., Coen, J. L., Gochis, D. J., Ahmadov, R., Peckham, S. E., Grell, G. A., Michalakes, J., Trahan, S., Benjamin, S. G., Alexander, C. R., Dimego, G. J., Wang, W., Schwartz, C. S., Romine, G. S., & Duda, M. G. (2017). The weather research and forecasting model: overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, *98*(8), 1717–1737. https://doi.org/10.1175/BAMS-D-15-00308.1.

Price, T. W., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, 91–99. https://doi.org/10.1145/2787622.2787712

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming. *ACM Transactions on Computing Education*, *18*(1), 1–24. https://doi.org/10.1145/3077618.

Reinhart, A., Evans, C., Luby, A., Orellana, J., Meyer, M., Wieczorek, J., Elliott, P., Burckhardt, P., & Nugent, R. (2022). Think-aloud interviews: a tool for exploring student statistical reasoning. *Journal of Statistics and Data Science Education*, 1–35. https://doi.org/10.1080/26939169.2022.2063209.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779.

Rijo-García, S., Segredo, E., & León, C. (2022). Computational thinking and user interfaces: A systematic review. *IEEE Transactions on Education*, 1–10. https://doi.org/10.1109/TE.2022.3159765

Sáez-López, J. M., del Olmo-Muñoz, J., González-Calero, J. A., & Cózar-Gutiérrez, R. (2020). Exploring the effect of training in visual block programming for preservice teachers. *Multimodal Technologies and Interaction*, *4*(3), 65. https://doi.org/10.3390/mti4030065.

Shen, H. (2014). Interactive notebooks: sharing the code. *Nature*, *515*(7525), 151–152. https://doi.org/10.1038/515151a.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 850–858. https://doi.org/10.1145/6592.6594.

Tawfik, A. A., Gatewood, J., Gish-Lieberman, J., & Hampton, A. (2022). Toward a definition of learning experience design. *Technology Knowledge & Learning*, *27*(1), 309–334. https://doi.org/10.1007/s10758-020-09482-2.

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 120–129. https://doi.org/10.1145/2999541.2999542

Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: implications for educational practice. *ACM Trans Comput Educ*, *17*(4), 1–13. https://doi.org/10.1145/2996201.

Vann, S., & Tawfik, A. A. (2020). Flow theory and learning experience design in gamified learning environments. In M. Schmidt, A. A. Tawfik, I. Jahnke, & Y. Earnshaw (Eds.), *Learner and user experience research*. EdTechBooks. https://edtechbooks.org/ux/flow_theory_and_lxd

Wang, C., Shen, J., & Chao, J. (2021). Integrating computational thinking in stem education: A literature review. International Journal of Science and Mathematics Education, 1–24. https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/s10763021-10227-5&casa_token=jdU2Pd8G40MAAAAA:oTTAX9vCNPBngGf-xS7X1d9TRE07TEBsjPDYwEvsd02Q2mNtmikfFikEv7e7N7zTs-55hKqL4Jv4k2b_

Wei, H. C., Peng, H., & Chou, C. (2015). Can more interactivity improve learning achievement in an online course? Effects of college students' perception and actual use of a course-management system on their learning achievement. *Computers & Education*, *83*, 10–21. https://doi.org/10.1016/j.compedu.2014.12.013.

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children*, 199–208. https://doi.org/10.1145/2771839.2771860

Whitenton, K. (2019, February 24). *How to respond to skepticism of testing small groups of users*. Nielsen Norman Group. https://www.nngroup.com/articles/responding-skepticism-small-usability-tests/

Wijekumar, K. (2021). Influence of emotions on digital learning. *Educational Technology Research and Development*, *69*, 55–57. https://doi.org/10.1007/s11423-021-09957-8.

Wing, J. (2011). Research notebook: Computational thinking–what and why? *The Link: The Magazine of Carnegie Mellon University's School of Computer Science*, 20–23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study. *Computer Science Education*, *29*(2–3), 177–204. https://doi.org/10.1080/08993408.2019.1565233.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through scratch in K-9. *Computers & Education*, *141*, 1–25. https://doi.org/10.1016/j.compedu.2019.103607.