Expansive Framing and Preparation for Future Learning in Middle-School Computer Science

Shuchi Grover and Roy Pea, Graduate School of Education, Stanford University, Stanford, CA
Stephen Cooper, Computer Science Department, Stanford University, Stanford, CA
Email: shuchig@stanford.edu, roypea@stanford.edu, coopers@stanford.edu

Abstract: Educators aspire to transfer of learning as a goal of their teaching. Expansive Framing and Preparation for Future Learning (PFL) are new perspectives on how to foster and assess transfer. As computing education makes its way into K-12 schools, efforts are underway to introduce children to programming in block-based environments like Scratch and Alice. This paper reports on a design-based research in progress that employs ideas of Expansive Framing and PFL to pedagogy and assessments in a middle school introductory CS curriculum that uses Scratch, and includes designed measures for evaluating how well it prepares students for success in future computing experiences with text-based programming.

Introduction

Computational Thinking (Wing, 2006), now widely recognized as a necessary skill for today's generation of learners, is increasingly being introduced in middle and early high school via programming in block-based introductory environments such as Scratch, Alice, and MIT App Inventor, among others (Grover & Pea, 2013). Ideally, educators would like these first experiences to be framed in such a way that learners can transfer their learning successfully to future computational experiences, which are likely to be in the context of higher-order, text-based programming languages. However, mediating transfer of learning from one context to another or even from one grade to the next is known to be difficult in STEM domains including programming (Kurland, Pea, Clement, & Mawby, 1986; Pea, 1987).

In designing and evaluating a 6-week introductory CS curriculum for middle school, we were guided by both work on "expansive framing" as a pedagogy to promote transfer and the *Preparation for Future Learning* (PFL) approach to assessing transfer. Expansive framing and PFL have yet to be investigated in the context of curricular interventions in CS Education. This paper describes our investigations around introductory CS instruction designed to nurture deep learning and expansive framing of CS constructs rather than focusing solely on the shallow surface features of the block-based programming environment, with the rationale that such an approach will prepare students for better success in future computational experiences, especially with text-based programming languages. The research effort includes the design and use of "*dynamic"* PFL assessments to assess how well students can apply their understanding of computing constructs learned in the context of Scratch to algorithmic solutions expressed in a more advanced text-based programming language.

Research Framework: Mediating Transfer and Preparation for Future Learning (PFL)

Transfer and PFL are embodied in notions of *deeper learning*, a topic at the center of a recent National Academy of Sciences synthesis report on developing transferable knowledge and skills for 21st century life and work (Pellegrino & Hilton, 2012). The need for classroom learning to be transferrable so it can be applied in practice and contexts outside of school is in keeping with the idea of 'learning and becoming in practice', the theme for the 11th International Conference of the Learning Sciences, 2014. The seminal literature on how people learn points to several critical features of teaching and learning that affect people's ability to transfer and suggests ways to facilitate transfer (Bransford, Brown & Cocking, 2000). While there is no single prescribed strategy for fostering learning for transfer, suggested instructional strategies aim to help students assemble new mental platforms for subsequent learning. As Engle et al. (2012) note, *expansive framing* fosters an expectation that students will continue to use what they learn later, create links between learning and transfer contexts so that prior learning is viewed as relevant during potential transfer contexts; and encourage learners to draw on their prior knowledge during learning, which may involve them transferring in additional examples and making generalizations. Educational psychologists also argue, "learners who compare cases will develop a more general problem-solving schema that primarily captures the common structure of the cases rather than the surface elements" (Gentner, Loewenstein, & Thompson, 2003). Consequently, in contrast to cases studied individually, analogous representations compared as part of a more expansive framing should be more easily retrieved when the learner encounters a new case with a similar structure.

The preponderance of studies in education literature however suggests that appropriate transfer comes with difficulty (Pea, 1987). Bransford and Schwartz (1999) also critique traditional tests of transfer for predominantly testing direct application of one's previous learning to a new setting or problem with no opportunities for learners to demonstrate their abilities *to learn* to solve new problems. To this end, they call for broadening previous conceptions of transfer by including an emphasis on people's "preparation for future

learning" (PFL) where the focus shifts to *assessments of people's abilities to learn from new resources*. The PFL perspective suggests that assessments of people's competencies can be improved by involving assessments that provide opportunities for new learning. Such "dynamic assessments" (Campione & Brown, 1990; Schwartz & Martin, 2004; Schwartz, Bransford & Sears, 2005) measure how well students "transfer in" skills to apply to their new learning rather than simply testing how well they "transfer out" of situations to solve problems.

Few studies in the realm of computing education have attended to transfer of learning from visual block-based environments to text-based ones. Dann et al. (2012) mediated transfer of learning from a special version of the Alice visual environment to the text-based Java environment. They contend that by using the exact same example in Alice and Java, their students succeeded with better learning results. More recently, Touretzky et al. (2013) used the idea of presenting contrasting cases and analogous representations of the use of the same computing constructs in a one-week summer camp that had 11 to 17-year-old children transition from Kodu to Alice to Robotics NXT-G in a structured way. By scaffolding instruction to help children see analogies between formalisms in each language, they sought to foster deeper conceptual understanding. For example, their strategies attempted to help children appreciate that "WHEN/DO in Kodu, If/Then in Alice, and SWITCH blocks in NXT-G all function as conditional expressions, even though they look different".

Building on this earlier research, we argue that in computing education, successfully mediating transfer will depend on expansively framed development of deeper conceptual understanding of computational thinking elements that children experience in their introductory computational learning. These include the ability to decompose problems and compose solutions, to understand fundamental notions of flow of algorithmic control that are broadly applicable, and to build practices with an academic vocabulary of the domain that will help students not only communicate computational ideas effectively but also aid in future programming experiences. Additionally, we contend that a concern for transfer of computational thinking experiences will be advanced using PFL assessments that measure readiness to work with more advanced programming environments. Successful PFL would require that students develop not only strong algorithmic thinking skills but also an *understanding of the underlying structures of programs beyond the syntax and surface features of the environment in which children are initially learning programming to more expansive frames in which similarities in deep structures across programming environments are anticipated, recognized and productively used*. Unlike earlier research that has attempted this by employing different programming languages to help students abstract deeper features of constructs, *our work is distinct in that we apply these ideas while using a single programming environment (Scratch), by employing the strategies described below.*

The remainder of the paper describes the features of an introductory CS "mini-course" inspired by the transfer of learning rationale above, and the empirical investigations for studying students' PFL as a result of this curricular intervention in a public middle school classroom.

Methods

This section describes the design-based research around a six-week middle school curriculum titled *'Foundations for Advancing Computational Thinking' (FACT)* designed to promote deeper engagement with foundational CT concepts and assess students through tests of direct application of skills as well as dynamic PFL assessments. The research question guiding this effort is: Does the FACT curriculum promote an understanding of algorithmic concepts that goes deeper than tool-related syntax details, as measured by Preparation for Future Learning (PFL) assessments?

Curriculum Design

Our curriculum focused on core CS concepts that would universally be identified as foundational to any computing experience for middle school. These include structured problem decomposition, and algorithmic notions of flow of control including conditional selection and repetition. The programming unit of the Exploring CS curriculum for high school (http://www.exploringcs.org/) inspired the curriculum and use of Scratch.

Instructional Approaches for Promoting Transfer and Deeper Learning

Our approach to teaching for transfer relies on using expansive framing and analogous representations of algorithmic solutions to help learners perceive these in forms more expansive than the constraints of a specific syntactical structure. We predict that guiding students to draw analogies between different formalisms will foster deep and abstract understanding of fundamental concepts of computational thinking.

To this end, the curriculum introduced new computational concepts through a "problem" example that required the use of the concept, demonstrations in Scratch, and an explanation of the concept using new computing vocabulary. Additional examples *using English & pseudo code were used to describe algorithms* involving the use of the concept so that students could see the concept being employed in algorithmic solutions that were represented in ways distinct from Scratch, thus setting them up for a more expansive framing of their learning than learning to program in Scratch alone. Short formative assessments included exercises and quizzes

involving pseudo code and short programming exercises in Scratch, and an end-of-unit activity involving use of the concept in the context of a more substantial programming task (in Scratch).

We used pseudo-code not only to describe and deliberately lay out the sequence involved in organizing the algorithmic steps to accomplish a goal (which has its own benefits), but also to introduce students to *analogical terms and representations of algorithmic solutions distinct from the Scratch environment*. Our reasoning was that this design would bolster familiarity with textual representations of programs, and analogous terms and description of loops and conditional structures that were different from Scratch. For instance, Scratch has only "REPEAT" and "REPEAT UNTIL" blocks for bounded and unbounded iteration. However, using terms like "WHILE" or "FOR" in pseudo-code aim to help students recognize that different terms can be used to describe the idea of repetition of steps (even though there are subtle differences in the ways in which these constructs operate in different programming languages). This approach was taken throughout the course accompanied by suggesting relevance of these terms and ideas in programming experiences in text-based languages such as Java and Python, e.g. "*Even though a loop in other languages like Java or Python will be expressed with terms like While or For, they help to accomplish the same things in an algorithmic process like the Repeat Until loop does in your Scratch program that finds the average test score for a class.*" At various points in each unit, students were also given an opportunity to examine the same algorithm put together in Java or some other language, so even though they were not being taught Java, the analogical instances helped students see the deeper structure of the program, for example, the rather simple "SAY" command in Scratch accomplishes the same goal as the more convoluted "system.out.print". Space constraints preclude inclusion of figures to demonstrate these analogous snippets of code shown to the students.

Dynamic Assessments for Assessing PFL

The curriculum design included the design of dynamic assessments for assessing PFL that attempt to measure how well students "transferred in" their conceptual understanding of computing constructs to learn from a new resource and apply it to understand code presented in a text-based language. The questions are described along with their goals and student results in the Results & Discussion section below.

The problems were thus preceded by "new learning" in the form of descriptions of how the syntax for the fictitious (Pascal/Java-like) text-based language worked. Two different types of syntax were explained, followed by questions each involving programs coded using the new syntax. For example, the following explanation formed part of the new (Pascal-like) syntax description that preceded Questions 1 and 2.

'<--' (left arrow) is used to assign values to variables. *For example:* n <-- 5 assigns the value 5 to the variable n
If there are blocks of compound statements (or steps), then the **BEGIN**..**END** construct is used to delimit (or hold together) those statement blocks (like the yellow blocks for REPEAT and IF blocks in Scratch).
FOR and **WHILE** are loop constructs like REPEAT & REPEAT UNTIL

WHILE (*some condition is true*)
BEGIN
... (Execute some commands) .....
END

Figure 1. Sample new syntax specification preceding questions in "dynamic" PFL assessment

In order for students to tackle Questions #3, #4 and #5, they had to use a Java-like syntax preceding the questions that were explained in a similar fashion to the prior Questions 1 and 2.

Study and Data Measures

The curriculum was taught for six weeks in April-May, 2013 in a public school classroom of 25 children from 7th and 8th grade (20 boys and 5 girls, mean age: ~13 years) enrolled in a semester-long *Computers* elective. As an elective class, students were self-selected. This accounted for the low numbers of female students. The class met for 55 minutes each day four times per week. The lead researcher on this effort was also the curriculum developer and teacher for the pilot 6-week FACT course. The regular *Computers* teacher was present in the class at all times.

Data Measures

Beyond a survey to assess prior programming experience, the following data were gathered for assessing students' learning through the FACT curriculum:

- *Measures of computational learning*: Students were given pre-post tests that measured their understanding of computational concepts especially in the context of Scratch. Questions were borrowed from prior work involving middle school kids and Scratch (Ericson & McKlin, 2012; Zur Bargury, Pârv & Lanzberg, 2013).
- *Preparation for Future Learning Test*: The post-test included a section pertaining to testing for PFL using the five questions as described in the table below. A researcher unrelated to the project graded the test.

Results and Discussion

The results on two out of the five PFL questions are described in the table below that describes the grading procedure as well. In Question #1, 8 out of the 25 respondents misunderstood the question and explained the code instead. Question #2 has been excluded from the table. It used a FOR loop, which has been found to be problematic for novice programmers (Robins, Rountree & Rountree, 2003). Not surprisingly, less than 50% of the students could tackle that question correctly. Question #3 met with the most success. 84% of the respondents got it completely correct or close to correct. Questions #4 & #5 were based on the same snippet of code that was very similar to #3 but with the added complexity of checking for divisibility by 2 and then incrementing one or the other counter variable. The answers were coded with one of 4 scores (3, 2, 1 or 0) as in Question #3. The wording of Question #5 also seemed to have caused some confusion, with some students giving the number of variables that the code used rather than how many numbers are processed by the loop. In both Questions #4 and #5, roughly 65% of the students got the answer correct or mostly correct.

Table 1: Sample PFL questions, their goals, and results describing student responses

<table>
<tr><th>PFL Assessment Question</th><th colspan="2">Goal of Question & Results</th></tr>
<tr><td>#1: When the code below is executed, what is displayed on the computer screen?
PRINT("before loop starts");
num <-- 0;
WHILE (num < 6) DO
BEGIN
num <-- num + 1;
PRINT("Loop counter number ", num);
END
PRINT("after loop ends");</td><td colspan="2">Goal: To test whether students were able to transfer in ideas of (a) sequence and what comes before and after the loop in addition to the things that happen within the loop, (b) looping (using a WHILE loop here), (c) how variable values change with each iteration of the loop, and (d) understanding the loop terminating condition.
Results: 32% of respondents misunderstood the question (and explained the code instead).
71% of those who followed the required format of the response (i.e., 68% of the total) got it correct while 19% were off by 1 (num went up to 5 rather than 6).
71% of the total number of students paid attention to the Print commands before and after the loop.</td></tr>
<tr><td rowspan="4">#3: Describe in plain English what this piece of code is doing. What are FirstCounter & SecondCounter keeping track of?
int TotalCount = 0; int FirstCounter = 0; int SecondCounter = 0;

while (TotalCount < 100) {
int num;
num = InputFromUser();
if (num == 0)
{
FirstCounter++;
}
else
{
SecondCounter++;
}
TotalCount++;
}</td><td colspan="2">Goal: To test whether students could make sense of (a) a FOR loop and (b) how the variables num and i changed with each loop iteration. The results are shown below (the column to the right indicates number of students)</td></tr>
<tr><td>Correct (3 points)
<u>Example</u>:" It is asking for 100 user inputs. If the input is 0, then it changes FirstCounter by 1. If it is something else, then it changes SecondCounter by 1."</td><td>19</td></tr>
<tr><td>Mostly Correct (2 points)
"it keeps track of 0s for 100 responses."</td><td>2</td></tr>
<tr><td>Mostly Wrong (1 point)
<u>Example</u>: variable total set to 0 / variable First set to 0 / variable second set to 0 / repeat total till total <100 / / input variable num / if num is equal to 0 / set first counter else second"
Wrong (0)
<u>Example</u>:"The code is asking for numbers that are less than 100. The two numbers."</td><td>1
3</td></tr>
</table>

We found the results to be largely encouraging. It is worth noting that: (1) In most cases, students were able to correctly get a sense for the program flow and at a fundamental level understood the concept of looping or conditional execution in the code, even though their responses were not always completely accurate. (2) The students whose responses were consistently right or consistently (completely) wrong mapped closely to the best and worst performers on the Scratch test. This is consistent with earlier literature that contends that skills mastery in the original context is essential for transfer (Kurland, Pea, Clement, & Mawby, 1986). (3) The nature of some of the errors committed in the PFL test were similar to those committed on the Scratch test, suggesting weak initial learning of some concepts. (4) Problems such as the "off-by-1" looping error (Question #5) or issues with the FOR construct (Question #2) are common among older novice programmers at the undergraduate level as well. (5) Most of the PFL question involved loops with variable manipulation, a topic that students found challenging and performed poorly on in the Scratch test as well.

Conclusion and Future Work

As a second iteration of this design-based research, the same curriculum was taught in the same classroom with a new cohort of students comprising 20 boys and 8 girls (mean age ~12.35 years). The assessments were largely unchanged from this study. The wording of Questions #1 and #5 was rectified to improve clarity of what was

being asked. We also added additional PFL assessment questions, and survey questions that elicit student beliefs of future applicability of their learning from this course. Data from this study are currently being analyzed.

Overall, the results of the PFL test were promising, and in answer to the research questions that guided this study, we found that students were able to transfer many algorithmic ideas from the 6-week Scratch curriculum to their new learning, and broadly interpret programs in a text-based language although the mechanics of some constructs were difficult to grasp. Most students evidenced an inherent understanding of the algorithmic flow of control even in a completely new programming context.

This paper suggests a successful approach to using the powerful ideas of Expansive Framing and Preparation for Future Learning (PFL) in introductory CS teaching and learning at the middle school level. Efforts such as the one described in this paper also provide middle school teachers with curricular and pedagogical ideas that promote a deeper engagement with computational thinking concepts, even as they use friendly block-based environments like Scratch. This study also makes an important contribution to the idea of textual but language-agnostic assessments that can be used as PFL assessments after teaching middle school students introductory CS using environments like Scratch and Alice. To conclusively establish the merits of this curricular approach, comparative investigations would be needed with children who are learning programming in Scratch/Alice using other types of curricula.

References

Bransford, J. D., & Schwartz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. In A. Iran-Nejad & P. D. Pearson (Eds.), *Review of Research in Education*, 24 (pp. 61–101). Washington, DC: American Educational Research Association.

Bransford, J. D., Brown, A. L, & Cocking, R. R. (Eds.). (2000). *How people learn*: *Mind, brain, experience and school (expanded edition)*. Washington, DC: National Academy Press.

Campione, J.C., & Brown, A.L. (1990). Guided learning and transfer: Implications for approaches to assessment. In N. Frederiksen, R. Glaser, A. Lesgold, & M. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition* (pp. 141-172). Hillsdale, NJ: Erlbaum

Dann, W. P., et al. (2012). Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 141–146.

Engle, R. A., Lam, D. P., Meyer, X. S., & Nix, S. E. (2012). How does expansive framing promote transfer? Several proposed explanations and a research agenda for investigating them. *Educational Psychologist*, 47(3), 215-231.

Ericson, B., & McKlin, T. (2012). Effective and sustainable computing summer camps. *Proceedings of the 43rd ACM technical symposium on CS Education*, 289-294.

Gentner, D., Loewenstein, J., & Thompson, L. (2003). Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, *95*(2), 393–408.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: a review of the state of the field. *Educational Researcher*, 42(1), 38-43.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, 2(4), 429-458.

Pea, R. D. (1987). Socializing the knowledge transfer problem. *Int'l Journal of Ed. Research*, 11(6), 639-663.

Pellegrino, J. W. & Hilton, M. L. (Eds.). (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century*. Washington, DC: National Academies Press.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: a review and discussion. *Computer Science Education*, *13*(2), 137-172.

Schwartz, D. L., Bransford, J. D. & Sears, D. (2005). Efficiency and innovation in transfer. In J. Mestre. (Ed.), *Transfer of learning from a modern multidisciplinary perspective* (pp. 1-51). Greenwich, CT, Information Age Publishing.

Schwartz, D. L. & Martin, T. (2004). Inventing to prepare for future learning: The hidden efficiency of encouraging original student production in statistics instruction. *Cognition and Instruction*, *22(2)*, 129-184.

Touretzky, D. S., Marghitu, D., Ludi, S., Bernstein, D., & Ni, L. (2013). Accelerating K-12 computational thinking using scaffolding, staging, and abstraction. In *Proceeding of the 44th ACM technical symposium on computer science education* (pp. 609-614). New York: ACM.

Wing, J. 2006. Computational Thinking. *Communications of the ACM. 49*(3), 33-36.

Zur Bargury, I., Pârv, B. & Lanzberg, D. (2013). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of ITiCSE'13*, 267-272. ACM.