

# Modeling the Learning Progressions of Computational Thinking of Primary Grade Students

Linda Seiter and Brendan Foreman  
John Carroll University  
Department of Mathematics and Computer Science  
University Heights, OH 44118  
lseiter@jcu.edu, bforeman@jcu.edu

## ABSTRACT

We introduce the Progression of Early Computational Thinking (PECT) Model, a framework for understanding and assessing computational thinking in the primary grades (Grades 1 to 6). The model synthesizes measurable evidence from student work with broader, more abstract coding design patterns, which are then mapped onto computational thinking concepts.

We present the results of a pilot-test study of the PECT Model in order to demonstrate its potential efficacy in detecting both differences in computational thinking among students of various ages as well as any clear overall progressions in increasing computational sophistication. Results of this sort are vital for establishing research-based and age-appropriate curricula for students in the primary grades, i.e., developing non-trivial, challenging but not overly daunting lesson plans that utilize the cognitive development stage of each grade level most effectively.

## Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]

## General Terms

Human Factors. Measurement.

## Keywords

Computational Thinking; Assessment; Programming; Scratch.

## 1. INTRODUCTION

Research and anecdotal evidence indicate that introducing concepts of computational thinking into primary grade curriculum creates a vital foundation for the development of future computational skills. However, there has been limited formal research of how computational skills develop in elementary grade children. In particular, a report on Pedagogical Aspects of Computational Thinking noted both a lack of optimal methods of assessing computational thinking in the primary grades, as well as a lack of formal research data concerning the progression of computational thinking concepts in K-12 [19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*ICER '13*, August 12–14, 2013, San Diego, California, USA.  
Copyright © 2013 ACM 978-1-4503-2243-0/13/08  
<http://dx.doi.org/10.1145/2493394.2493403>.

In this paper, we introduce the Progression of Early Computational Thinking (PECT) Model, a framework for understanding and assessing computational thinking in the primary grades (Grades 1 to 6). The model provides the theory for a research-based, age-appropriate computational thinking curriculum. Based on the concept of design patterns as initially described by Gamma et al [12], the PECT Model categorizes computational thinking primarily through the level of skill utilized in design patterns, calculated by measurable evidence from programs written in Scratch [21]. Proficiency levels in the design patterns are then mapped onto more abstract computational thinking concepts, including procedures and algorithms, problem decomposition, parallelization, abstraction and data representation. This methodology thus allows us to assess computational thinking knowledge, skills and abilities both from use of individual programming elements and from commonly utilized contexts via design patterns.

In this paper, we review the current state of research of the assessment of computational thinking. We then describe the PECT Model and its rationale. Lastly, we describe a pilot study of the assessment and analysis of student work from teacher galleries on the Scratch website.

## 2. ASSESSMENT OF COMPUTATIONAL THINKING

The Computer Science Teachers Association has proposed computer science standards for K-12 [9]. The standards for elementary and middle school recommend a focus on problem solving and computational thinking, to be addressed either in discrete computing courses or by adding short modules to existing science, mathematics, and social studies courses.

While there is a growing body of literature that describes experiences in integrating computational thinking into the K-12 curriculum, efforts involving the formal assessment of computational thinking has primarily focused on middle school grades and above. The Planning for the Assessment of Computational Thinking workshop was held in December 2011, with a goal of creating templates to support the design of families of computational thinking assessment tasks [19]. The workshop focused on aligning the model of evidence centered assessment design and design patterns [18, 22] with the secondary school-level Exploring Computer Science curriculum [13].

Basawaptna et al propose a Computational Thinking Pattern Quiz to assess whether computational thinking patterns can be recognized in a non-programming context [4]. A pilot study was

performed during a workshop with middle school teachers and college students, measuring common game design patterns.

Koh et al propose two approaches for the semantic evaluation of computation thinking: Program Behavior Similarity (PBS) and Computational Thinking Pattern Graph (CTPG) [15]. PBS compares games based on their rule sets and calculates program behavior similarity. CTPG offers a higher-level approach using nine gaming patterns (cursor control, generation, absorption, etc.) spiraled in increasing difficulty. The score for each pattern is based on the PBS score between a particular game and a canonical solution. The CTPG allows two solutions to be compared based on the presence of computational thinking patterns.

Brennan and Resnick propose three approaches to assessment of computational thinking in elementary school children: project portfolio analysis, artifact-based interviews, and design scenarios [7]. Project portfolio analysis uses a tool called Scrape [25] to analyze the presence and frequency of various blocks found in Scratch programs. While this technique has benefit, it does not capture the design patterns and algorithmic structure within the programs as this would require a more qualitative analysis of block patterns. Artifact-based interviews support a better understanding of the design patterns employed by an individual student, as well as the processes they use to develop their programs. The final approach, design scenarios, involves showing students a set of programs and having them explain their functionality and demonstrate how to extend them. Brennan and Resnick recommend that computational thinking assessment involve creating and critically examining a collection of projects developed by a child, in particular to see how understanding of computational thinking concepts evolves over time.

Franklin et al propose a model for integrating assessment of computational thinking into the design of a Scratch-based curriculum [11]. The model assesses proficiency in event-driven programming, initialization, synchronization and animation. A small pilot test with middle school students show positive results.

Lewis compares two visual programming languages, Scratch and Logo, for differences in attitudinal and learning outcomes among a group of sixth grade students [17]. Scratch afforded the students a better understanding of conditional statements, while Logo fostered higher confidence in programming ability.

Kazakoff and Bers examine the impact of a robotics curriculum on sequencing ability in prekindergarten and kindergarten students [14]. The children were introduced to robotics programming using an interface called CHERP (Creative Hybrid Environment for Robotic Programming) [5]. Evaluation before and after robotics instruction was done using an assessment derived from the Baron-Cohen picture-sequencing test [2]. The study demonstrated that young children can learn to program a robot to complete a variety of tasks, while also improving their score on a sequencing assessment.

Denner, Werner and Ortiz developed a coding scheme to identify the extent to which programs written by middle schools girls corresponded with computer science programming concepts [10]. Each game was coded based on three high-level categories: (1) programming; (2) documenting and understanding software; and (3) designing for usability, with 24 subcategories such as parallelism, use of random, variable test, character appearance changes, etc. The coding of a particular program reflected the extent to which a programming concept was present. Their

research demonstrates that game programming can be effective in engaging middle school students in moderate levels of complex programming activity, moderate levels of usability, but only low levels of code organization and documentation. Wilson, Hainey and Connolly further adapted the coding scheme and analyzed the level of programming proficiency found in 29 Scratch programs developed by children in grades 4 to 7 [23]. Their results show user interaction, loops and conditionals to be the most commonly used programming concepts, with random numbers being among the least frequently used concepts.

### 3. MODELING THE PROGRESSION OF EARLY COMPUTATIONAL THINKING

We propose the Progression of Early Computational Thinking (PECT) Model. This model assumes that every student has a latent (that is, not directly observable) proficiency in Computational Thinking that manifests itself in the student's ability to design and implement programs to complete specific tasks. Thus, there are three levels of variables that a conceptual framework of assessment needs to involve: (1) A broad set of categories that altogether comprise the primary concepts involved with computational thinking; (2) A set of design patterns, which serve as the primary models for completing tasks or goals of each program; and (3) A set of explicit programming constructs that the design patterns rely upon. Accordingly, the PECT model is comprised of three fundamental components of decreasing abstraction, 1) Computational Thinking Concepts, 2) Design Pattern Variables, and 3) Evidence Variables, which altogether provide a useable framework by which Computational Thinking in the primary grades can be observed and analyzed.

The **Evidence Variables** is a set of concrete, ranked characteristics of code written in Scratch that measure levels of sophistication in specific computing categories:

- Looks
- Sound
- Motion
- Variables
- Sequence & Looping
- Boolean Expressions
- Operators
- Conditionals
- Coordination
- User Interface Event
- Parallelization
- Initialize Location
- Initialize Looks

These variables serve as the directly measurable variables of student work, that is, the concrete use of particular Scratch programming blocks written as needed to realize a particular coding design pattern.

The **Design Pattern Variables** is a set of contextual proficiencies based around common coding patterns in Scratch:

- Animate Looks
- Animate Motion
- Conversate
- Collide
- Maintain Score
- User Interaction

These variables measure the sophistication and degree by which a coding strategy or pattern is implemented. Their rankings are dependent on the scores of the relevant Evidence Variables as well as a qualitative analysis of each program. They are used as

the primary method for determining a student’s level of understanding of the relevant Computational Thinking Concepts.

The **Computational Thinking Concepts** is a subset of the computation thinking concepts proposed by the CSTA [3, 8]:

- Procedures and Algorithms
- Problem Decomposition
- Parallelization and Synchronization
- Abstraction
- Data Representation

These variables are more qualitative in structure and serve as a means to understand a student’s overall level of computational thinking. The precise relationships among the elements of the PECT model will be described in the following subsections.

### 3.1 Evidence Variables

The presence of the concrete computational aspects of the student work is measured through use of the Evidence Variables. These variables represent the very basic “nuts and bolts” components of Scratch programming and are roughly organized around the block categories found in Scratch, with some further refinement of groupings. Figure 1 contains the rubric for scoring of Evidence Variables. We adapt and extend the coding scheme proposed by Wilson, Hainey and Connolly [23]. In our model, Evidence Variables are assessed using a rubric based on our experience in teaching Scratch programming to students across the range of K-12 grades. The Evidence Variables for parallelization, initialize location, and initialize looks are scored as 0 (absent) or 1 (present). The remaining Evidence Variables are keyed by a 3-point scale according to what is evident within the student work artifact. The scale organizes a particular Scratch programming category or concept in order to represent increasing levels of computational thinking: 1= Basic, 2=Developing, 3=Proficient. If no evidence according to the rubric of a given Evidence Variable is present in a student work artifact, it receives a score of 0.

	1 -Basic	2 - Developing	3 -Proficient
Looks	Say, think.	Next Costume. Show. Hide.	Switch to costume. Set, change color/size/etc.
Sound	Play sound,note,etc.	Plays Play until done.	
Motion	Move, goto sprite, point,turn.	Goto x,y. Glide to x,y.	Set,change X, Y.
Variables	Scratch variable (sprite, mouse pointer, answer, etc)	New variable (set,change)	New list.
Sequence & Looping	Sequence	Repeat, Forever.	Forever If. Repeat Until.
Boolean Expressions	Sensing operators.	<, =, >.	And, or, not.
Operators	Math	String and random	List
Conditional	If.	If... else....	Nested If/ If... Else...
Coordination	Wait.	Broadcast, When I Receive.	Wait Until.
User Interface Event	Green Flag clicked.	Key press, sprite clicked.	Ask and wait.
Parallelization	2 scripts start on same event.		
Initialize location	Set location properties (x,y,etc.) on green flag.		
Initialize looks	Set looks properties (costume, visibility,etc.) on green flag.		

Figure 1 - Evidence Variables

For example, consider the Evidence Variable for the *Looks* category. The Basic level includes the *Say* and *Think* blocks, which are simple action blocks that cause an observable behavior when the program is executed. A child can use these blocks without considering the concept of object/sprite state or properties. The Developing level includes the *next costume*, *show* and *hide* blocks. These blocks modify a given sprite’s property (costume or visibility), which requires a child to understand that objects have properties or state that can be manipulated by the program. The proficient level includes the *switch to costume*, *set* and *change* blocks that require a child to comprehend the assignment of a property to a specific value.



Figure 2 - Sample Program

Figure 2 contains a sample Scratch program that will be used to demonstrate the Evidence Variable coding scheme. The program simulates a conversation between two sprites: a duck (sprite1) and a cow (sprite2). Each sprite has a script that is executed on a “Green Flag Clicked” event, which is how a Scratch program is typically started. Scratch supports multithreading, thus the two scripts execute concurrently. The cow waits while the duck says “Guess what I am doing this summer”, then the duck waits while the cow says “what?”. The scripts continue with the two sprites taking turns speaking. The program falls within the “Basic” proficiency level for each individual Evidence Variable present, as shown in Figure 3.

EVIDENCE VARIABLE	PROFICIENCY SCORE
Looks	1 – Say
Sequencing and Looping	1 – Sequence
Coordination	1 – Wait
User Interface Event	1 – Green flag clicked
Parallelization	1 – Two scripts start on same event

Figure 3 -- Sample Program Evidence Variable Score

### 3.2 Design Pattern Variables

A child’s proficiency in computational thinking is necessarily more complex than a simple aggregation of the understanding and use of discrete, independent Knowledge, Skills and Abilities (KSA’s e.g., use of the various Scratch blocks) as measured through Evidence Variables. Rather each KSA works within certain frequently implemented abstract contexts, namely, design patterns; and proficiency in computational thinking involves primarily the abilities (1) to recognize the need for and to use a specific design pattern for a specific problem or context and (2) to use commands, syntax, variables and other fundamental programming elements correctly within the chosen design pattern.

Student proficiency is thus further measured using a set of contextual cognitive proficiencies called Design Pattern Variables, which are based around common coding patterns in Scratch. A given student's overall proficiency of computational thinking is measured as an average of her or his proficiencies in each of the Design Pattern Variables. Figure 4 lists the set of Design Pattern Variables presently included in the PECT model.

Animate Looks	Basic		Developing	Proficient	
	change appearance		initialize and change	state synchronized	event synchronized
Looks Initialization	0	0	1	1	1
Sequencing and Looping	1	2	1	3	1
Looks	2	2	3	3	3
Coordination	1	0	1	0	2
Parallelization	0	0	0	1	0

Animate Motion	Basic		Developing	Proficient
	change location		initialize and change	relative movement
Location Initialization	0	0	1	1
Sequencing and Looping	1	2	1	1
Motion	1	1	2	3
Coordination	1	0	1	0

Conversate	Basic		Developing	Proficient	
	monologue		time synchronized dialog	state synchronized	event synchronized
Sequencing and Looping	1		1	3	1
Looks or Sound	1(looks), 2(sound)		1,2	1,2	1,2
Coordination	0		1	0	2
Parallelization	0		1	1	0

Collide	Basic		Developing	Proficient
	discrete test		continuous, asynchronous	discrete, blocking
Sequencing and Looping	1		3	0
Boolean Expression	1		1	1
Conditional	1		0	0
Coordination	0		0	3
Parallelization	0		1	1

Maintain Score	Basic		Developing	Proficient	
	count		test counter	state synchronized	event synchronized
Sequencing and Looping	1		1	3	1
Variable	2		2	2	2
Conditional	0		1	1	1
Boolean Expressions	0		2	2	2
Coordination	0		0	0	2
Parallelization	0		0	1	1

User Interaction	Basic		Developing	Proficient
	key press		mouse	keyboard input
Sequencing and Looping	1		1	1
Motion	0		1	0
User Interface Event	2		0	3
Variable	0		mouse pointer	answer

Figure 4 - Design Pattern Variables

It should be noted that the Design Pattern Variables focus on the traditional story-telling and gaming perspectives of Scratch. As further development of the PECT Model occurs, it is hoped that other non-gaming design patterns will emerge. However, of primary importance, is that levels of proficiency in these Design Pattern Variables in the aggregate indicate some level of proficiency in overall computational thinking.

For each Design Pattern Variable, a student is assessed on the following qualitative scale:

1. Basic - Student has a functional understanding of the design pattern and most of its corollary KSA's.
2. Developing - Student has an advanced but not complete understanding of the design pattern and most of its corollary KSA's.
3. Proficient - Student has a complete understanding of the design pattern and its corollary KSA's.

We explain the rubric in Figure 4 using the *Conversate* pattern as an example. *Basic* proficiency represents a single sprite monologue, which requires sequential flow of a single script that uses either a sequence of "say" blocks (looks=1) or recorded sound blocks (sound=2) to simulate a sprite speaking. *Developing* proficiency represents a dialog between two or more sprites and involves the timed coordination of parallel scripts using Scratch's *wait* block (coordination = 1), as demonstrated in the example program in Figure 2. The *Proficient* level represents a conversation that is synchronized either by conditionally testing object or program state within a loop (sequencing and looping=3), or by using a broadcast event (coordination = 2).

The sample Scratch program from Figure 2 obtains a score of 2 (Developing Proficiency) for the *Conversate* design pattern. The program has Evidence Variable scores of Sequencing and Looping=1 (sequence), Looks =1 (say), Coordination=1 (wait) and Parallelization=1 (multiple scripts start on green flag clicked event). Thus while the individual Evidence Variables used in the program are scored at the Basic proficiency (Figure 3), the combined use of the blocks to implement a dialog allows the program to be scored at the Developing proficiency for the *Conversate* design pattern.

Note that the rubric in Figure 4 serves as a guideline for describing the minimal requirements for each Design Pattern Variable. For each program, we use the Evidence Variable scores to infer the potential score for a Design Pattern Variable, but we also perform a qualitative analysis to confirm the presence and correct implementation of the pattern.

### 3.3 Computational Thinking Concepts

The Computational Thinking Concepts represent a set of general categories: Procedures and Algorithms, Problem Decomposition, Parallelization, Abstraction, and Data Representation. The concepts are cognitive in that they are based on latent traits of cognitive activity. Furthermore, they are focused on computational thinking abstractions, the evidence of which may be found in programs in contrast with the processing skills needed to create and modify programs. Process skills – such as debugging and testing -- are of course as prominent and important in computational thinking as the concept domains. However, measuring them is presently outside of the scope of our research.

### 3.4 Component Synthesis

The PECT Model synthesizes its various components using Evidence-Centered Assessment Design for creating, implementing and analyzing assessments of student work [18]. Each Computational Thinking Concept is associated with one or more Design Pattern Variables, corresponding to patterns that utilize the cognitive knowledge and activity of the given concept. Student understanding within each Computational Thinking

Concept is measured by proficiency levels of the Design Pattern Variables, as described in Figure 5.

Basic proficiency for each Design Pattern Variable indicates a child’s initial cognition of procedures and algorithms, i.e. scripts that contain a series of ordered steps to solve a problem. Basic proficiency of the Animate Looks and Animate Motion patterns also serves as evidence of an initial understanding of data representation, as each sprite has properties that change during the animation. In addition to algorithmic thinking, basic proficiency of the Maintain Score design pattern requires an understanding of data representation through the creation and manipulation of a new variable.

	Procedures and Algorithms	Problem Decomposition	Parallelization and Synchronization	Abstraction	Data Representation
<b>Basic</b>					
Animate Looks	sequence				sprite properties
Animate Motion	sequence				sprite properties
Conversate	sequence				
Collide	conditional				
Maintain Score	initialize, then increment				new integer variable
User Interaction	sequence triggered by user interface				
<b>Developing</b>					
Animate Looks	initialize, then change			initial state	property assignment
Animate Motion	initialize, then change			initial state, 2D canvas	property assignment
Conversate	alternating say, wait sequence		multithreaded, blocking, synchronize on time delay		
Collide	test for collision in infinite loop	modularize collision detection	multithreaded, asynchronous	program state, continuous event	
Maintain Score	increment, then test				new integer variable
User Interaction	move relative to mouse pointer				mouse pointer variable
<b>Proficient</b>					
Animate Looks	trigger animation as needed	modularize sprite animation	multithreaded, asynchronous, synchronize on state and event	program state and events	property assignment
Animate Motion	move relative to current location				relative property assignment
Conversate	trigger speaker as needed	modularize speaker role	multithreaded, asynchronous, synchronize on state and event	program state and events	
Collide	wait for collision		multithreaded, blocking	program state, discrete event	
Maintain Score	global reaction to score	modularize scene change	multithreaded, asynchronous, synchronize on state and event	program state and events	new integer variable
User Interaction	prompt, then use value		block on keyboard input		input storage variable

**Figure 5 – Design Pattern Variables mapped to Computational Thinking Concepts**

Beyond basic proficiency, each subsequent level relies on and serves as evidence of additional Computational Thinking Concepts. For example, the Developing proficiency level for both Animate Looks and Animate Motion require proper initialization of a sprite’s properties (appearance or location) when the program first starts. This requires both algorithmic thinking (initialize before changing), as well as a more advanced understanding of data representation (assigning a value to a property), which is a

difficult Computational Thinking Concept for young children to grasp. It is not unusual to see a child manually initializing the properties of a sprite prior to every execution of their program, by either dragging the sprite to a starting location and/or directly setting the initial costume and appearance through the Scratch user interface. Eventually they grasp how to programmatically initialize the sprite’s state by using the appropriate property assignment blocks.

#### 4. PRESENT STUDY

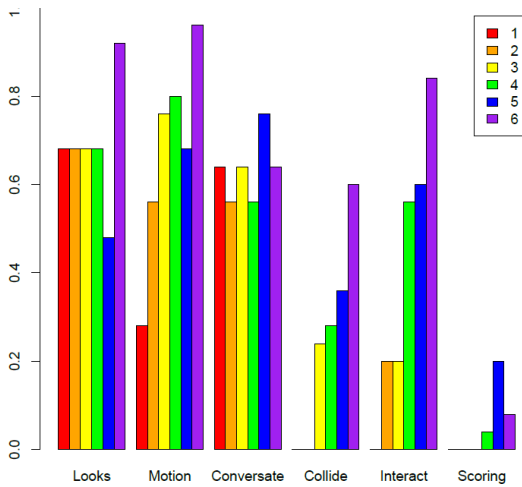
The present study was conducted in order to pilot-test the PECT Model for codifying computational thinking and its overall efficacy in detecting both differences in computational thought among students of various ages, along with any clear overall progressions in ascending computational sophistication from younger to older age groups. Results of this sort are vital for establishing research-based and age-appropriate computational curricula.

The present study consisted of choosing 150 projects from teacher galleries on the Scratch website, assessing the proficiency levels of the Evidence Variables and Design Pattern Variables of each project as codified by the PECT Model, and then analyzing the overall results across grade levels. Initially, 25 projects were chosen from each grade (grades one through six). The method for choosing these projects was to select from teacher galleries, where the teacher clearly specified that all projects within a particular gallery were representative of a single grade. No more than 5 student projects were selected per gallery. This is clearly not a simple random sample of the available data but rather a way to generate some sample projects in order to test the efficacy of the rubrics created for the PECT Model. As described in the next section, we will be conducting future studies in which the production and collection of representative student work will be done in a controlled process.

Since we wanted to get an overall sense of the range of computational thinking across each grade level without spurious correlation that might arise from students adapting other students’ projects, any projects from the same classroom with significant similarities were discarded and replaced with other projects. Also, remixed projects, pair programming, and projects that used sprites with pre-defined scripts were discarded.

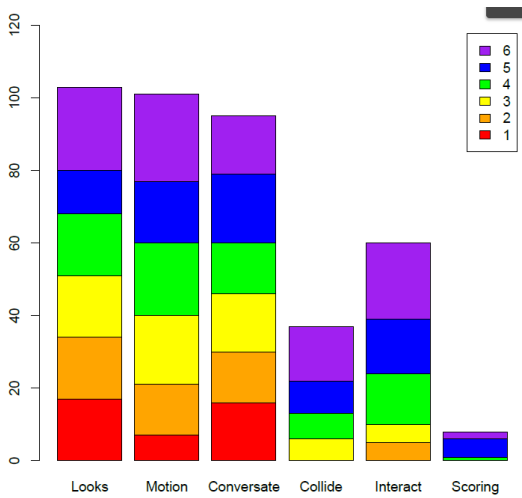
During the assessment component of this study, the rubrics for the Evidence Variables and Design Pattern Variables were modified as inaccuracies with the initial rubrics were made evident from the real-world data. These modifications were made in order to better illustrate what computational thinking KSA’s were manifesting themselves within student work, that is, to provide a more illustrative image of what each student was thinking and producing with their project. For example, the Looks Evidence Variable was refined to clearly separate blocks that involve an explicit value assignment of a Sprite property (switch costume1) from blocks that did not involve direct value assignment (next costume, show, hide). A similar refinement was made to the Motion Evidence Variable.

Once the rubrics had been suitably modified for efficacy, several striking aspects of the student projects came to the surface. First, the frequencies of each Design Pattern Variable by grade level were generated (Figure 6). As might be expected, most of the Design Pattern Variables were generally used more frequently at each increasing grade level. Furthermore, this data seems to indicate that certain patterns are best suited for certain grades.



**Figure 6 -- Frequency of use of the six Design Pattern Variables among the student projects in Grades 1-6**

This is made more evident by Figure 7, which shows the grade breakdown of the usage of each Design Pattern Variable. Note that Conversate, Animate Looks and Animate Motion were each utilized rather uniformly across all six grades. However, Collision and Scoring, both of which utilize some advanced concepts (conditionals and variables), were under-represented by all grades except 5 and 6.



**Figure 7 – Grade breakdown of Design Pattern Variable Use**

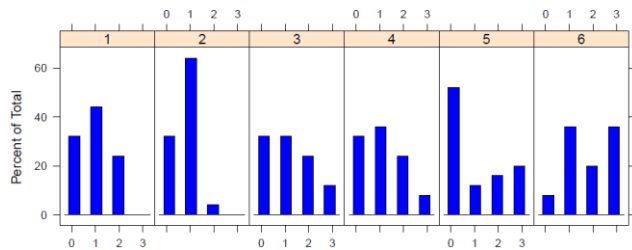
Conversate, Animate Looks and Animate Motion were the only patterns used for a majority of the projects within each grade, with the combined frequency of use generally increasing from Grade 1 to 6. A very curious phenomenon occurs with Conversate -- namely, the frequency of use would seem to drop in Grade 6. This may be due to students losing interest in the use of this pattern as they shift to learn new patterns. Finally, the sparse use of the Maintain Score design pattern seems to indicate it is too advanced for most students within primary grades. In light of the Computational Thinking Concepts involved with this pattern (Figure 5), the PECT Model seems to give us insight as to why. Namely, the pattern requires an understanding of Data Representation (variable creation and assignment), which may be more challenging to grasp than other Computational Thinking Concepts.

By looking at the individual proficiency levels of Animate Looks and Conversate by grade, we achieve a better understanding of the development of computational thinking at these early grades. Figure 8 gives the summary statistics of the proficiency levels by grade for projects that utilized the Animate Looks pattern (Recall that 0=Not Used, 1=Basic, 2=Developing, 3=Proficient). The resulting histograms are given in Figure 9. Note the first column for each grade given in Figure 9 represents the percent of students that did *not* use the pattern, while the remaining columns represent Basic, Developing and Proficient levels of use.

	1	2	3	4	5	6
Min	0	0	0	0	0	0
1 <sup>st</sup> Quart	0	0	0	0	0	1
Median	1	1	1	1	0	2
Mean	0.92	0.72	1.16	1.08	1.04	1.84
3 <sup>rd</sup> Quart	1	1	2	2	2	3
Max	2	2	3	3	3	3

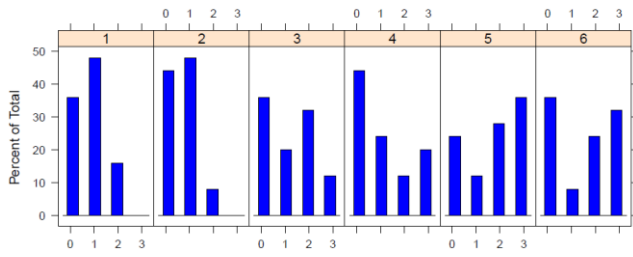
**Figure 8 -- Summary statistics of the proficiency levels of Animate Looks by grade level**

This data seems to indicate that within Grades 3 and 4, several students began to develop a more sophisticated understanding of how to design and use aspects of animation. That is, with maturity, the population of students moved from a mostly basic understanding to a relatively uniform distribution of understanding across Levels 1, 2 and 3. This interpretation is evident both in the increase of mean from Grade 3 to Grade 4 as well as in the further uniform aspect of the histograms from Grade 3 to Grade 4. Recall that level 2 proficiency (Developing) for Animate Looks requires initialization of the sprite's appearance, which involves algorithmic thinking (initialize, then change), data representation (explicit assignment of sprite property), as well as the abstract concept of program state (sprite properties when green flag clicked). We see little evidence of this level of proficiency until the third and fourth grade. Note that Animate Looks does not rely on knowledge of the coordinate system; it simply requires the manipulation of costumes, size or visibility. Thus the delay in proficiency is potentially related to a lack of computational thinking rather than mathematical skills.



**Figure 9 -- Proficiency of Animate Looks by grade**

A similar phenomenon occurs with the Conversate pattern in which the proficiency levels become much more uniform from Grade 3 and Grade 4 as can be seen in Figure 10. Recall that the first column represents the percentage of students that did *not* use the design pattern in their program.



**Figure 10 -- Proficiency of Conversate by grade**

The histograms for the remaining Design Pattern Variables are given in Figure 11. Referring back to Figure 5, we see the histograms confirm that proficiency levels for patterns that involve concepts such as parallelization, problem decomposition, abstraction and data representation tend to be delayed until the later grades.

Basic proficiency of the Collide design pattern begins to present itself in third grade and shows evidence of algorithmic thinking that involves conditional logic. The Developing level requires modularization of the collision detection to a separate script, which demonstrates Problem Decomposition. The collision detection script is typically executed concurrently with scripts that animate a sprite, thus providing evidence of an understanding of parallelization and asynchronous threading. The Proficient level requires the use of an advanced synchronization block (wait until), which is too abstract for all but a few students.

Basic proficiency for User Interaction, which involves the implementation of a key press event handler, is evident in second and third grade. The Developing proficiency involves animation relative to the mouse pointer, which requires the use of a built in Scratch variable (mouse pointer). The Proficient level involves prompting for user input with the use of the Ask block, which requires the use of a predefined Scratch variable to store the answer. The histograms indicate that proficiency levels that involve Data Representation (variable referencing and assignment), also required for the Maintain Score pattern, is not present until later grades.

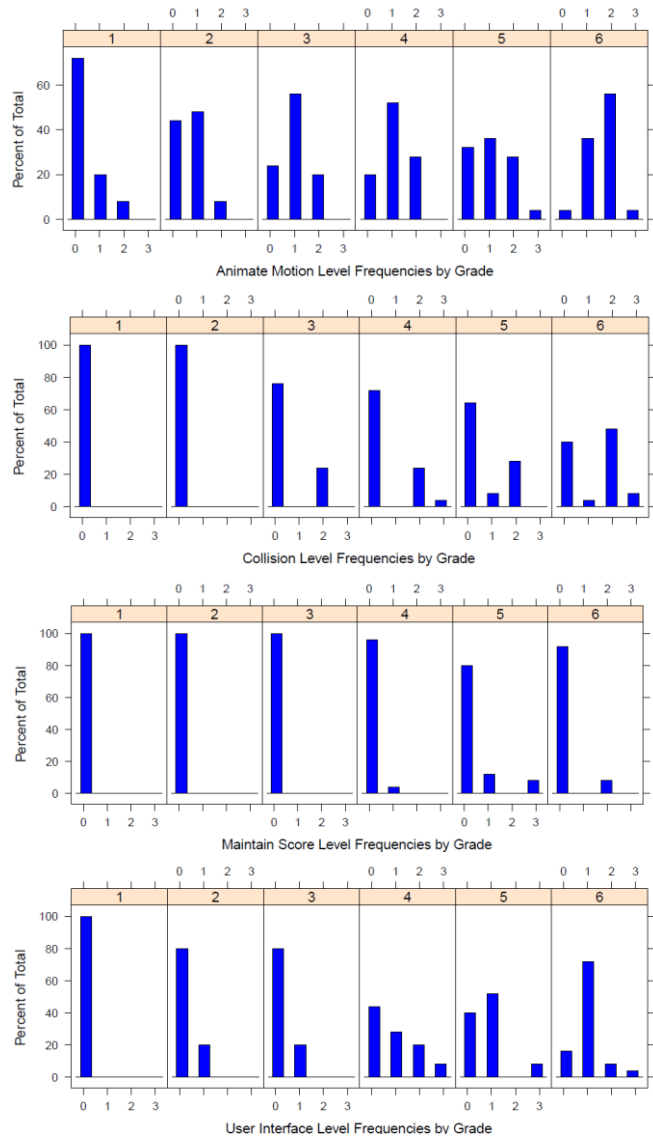
These results become much more striking when one recalls that it is highly unlikely that any of the student authors of these projects were aware of the notion of design patterns when creating these projects. Rather in the pursuit of achieving certain objectives they utilized the concepts of various examples, and this usage clearly increased in sophistication as grade level increased as represented by the consistency of the results of the generated rubric data.

We thus see that, by delineating the student work through the use of Design Pattern Variables, the PECT Model is potentially an effective model to use in studying and understanding the development of computational thinking.

## 5. FUTURE WORK

The next step for the development of the PECT Model is to generate a controlled, unbiased set of student work across Grades 1 to 6 of large enough size for reliability and validity verification. To this purpose, we are collaborating with two local public school districts in creating a three-year project, one of the end goals of which is generating a sizeable set of student work to which to apply the PECT Model rubrics. Scratch will be used to integrate computational thinking into the overall curriculum of the students. Each student will create a portfolio of computing artifacts across different subject matter including mathematics, science, language arts, social studies and the creative arts. The student work from

these projects and activities will be collected and analyzed. This will involve (1) analyzing student projects and assessing them for computational benchmarks using the PECT Model; and (2) developing an overall picture of how each student developed in their computational thinking during the year.



**Figure 11 – Proficiency of Remaining Design Patterns**

The final project phase will consist of synthesizing the data collected during the project and creating trajectories of development for each of the Computational Thinking Concepts by age. The data will be analyzed to best describe the overall proficiency levels of the associated Design Pattern Variables and corollary Evidence Variables cross-referenced with age and student progression across the academic year. Correlation amongst the Design Pattern Variables and Evidence Variables will be analyzed and cross-referenced with age and student progression. Of particular interest will be the investigations of how proficiency levels of the Evidence Variables are related to the Design Pattern Variable being utilized and of how these relationships influence achievement within the Computational

Thinking Concepts. From this analysis, a set of most likely proficiency levels of both Design Pattern Variables and Evidence Variables by age will be generated using Bayesian statistical techniques. Furthermore, growth of computational thinking during an academic year within individual students will be investigated. The prototype progression of early computational thinking within each Computational Thinking Concept will be generated from this analysis.

## 6. REFERENCES

- [1] Astrachan, O., Barnes, T., Garcia, D., Paul, J., Simon, B., and Snyder, L. CS principles: piloting a new course at national scale. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, pp.397-398. 2011.
- [2] Baron-Cohen, S., Leslie, A., Frith, U. Mechanical, Behavioural and Intentional understanding of picture stories in autistic children. *British Journal of Developmental Psychology*, 4(2), pp.113-125, 1986.
- [3] Barr, V. and Stephenson, C. Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *Inroads*, 2(1):48-54, February, 2011.
- [4] Basawapatna, A., Koh, K., Repenning, A., Webb, D., and Sekeres Marshall, K. Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 245-250. 2011.
- [5] Bers, M.U. The TangibleK Robotics Program: Applied Computational Thinking for Young Children. *Early Childhood Research and Practice*, 12(2), 2010.
- [6] Brennan, K., Chung, M. Creative Computing, a design-based introduction to computational thinking. Retrieved from <http://scratched.media.mit.edu/resources/scratch-curriculum-guide-draft>, 2011.
- [7] Brennan, K., Resnick, M. New frameworks for studying and assessing the development of computational thinking. *2012 Annual Meeting of the American Educational Research Association (AERA'12)*, Vancouver, Canada, 2012.
- [8] Computer Science Teachers Association. Computational Thinking. 2012. <http://csta.acm.org/Curriculum/sub/CompThinking.html>.
- [9] —. "K-12 Computer Science Standards." 2011. <http://csta.acm.org/Curriculum/sub/K12Standards.html>.
- [10] Denner, J., Werner, L. and Ortiz, E. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, Volume 58, Issue 1, pp. 240-249, January, 2011.
- [11] Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., Dreschler, G., Aldana, G., Assessment of computer science learning in a scratch-based outreach program. In *Proceedings of the 44<sup>th</sup> SIGCSE technical symposium on Computer Science education, SIGCSE '13*. ACM, 2013.
- [12] Gamma, J., Helm, R., Johnson, R., Vlissides, J. Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., 1995.
- [13] Goode, J. and Chapman, G. Exploring Computer Science. Computer Science Equity Alliance, Los Angeles. <http://www.exploringcs.org/>, 2009.
- [14] Kazakoff, E., and Bers, M. Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia*, 21(4), 371-391, 2012.
- [15] Koh, K., Basawapatna, A., Bennett, V., and Repenning, A. Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. In *Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '10)*. IEEE Computer Society, Washington, DC, USA, 59-66. <http://dx.doi.org/10.1109/VLHCC.2010.17>, 2010.
- [16] Lee, K. Childhood Cognitive Development: The Essential Readings. Wiley-Blackwell, Cornwall, UK. 2000.
- [17] Lewis, C. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*. ACM, New York, NY, USA, 346-350. <http://doi.acm.org/10.1145/1734263.1734383>, 2010.
- [18] Mislevy, R. J., Almond, R. G., and Lukas, J. F. A brief introduction to evidence-centered design. Research Report RR-03-16, Educational Testing Service, Princeton, NJ, 2003.
- [19] National Research Council. Report of a Workshop on Pedagogical Aspects of Computational Thinking. Washington, DC: The National Academies Press, 2011.
- [20] Rathus, S. A. Childhood and Adolescence: Voyages in Development. Wadsworth, Belmont, California, 2010.
- [21] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. Scratch: Programming for All. *Communications of the ACM*, vol. 52, no. 11, pp. 60-67, Nov. 2009.
- [22] Snow, E., Haertel, G., Fulkerson, D. Feng, M., & Nichols, P. Leveraging evidence-centered design in large-scale and formative assessment practices. Paper presented at the *2010 annual meeting of the National Council on Measurement in Education (NCME)*, Denver, CO, 2010.
- [23] Wilson, A., Hainey, T., Connolly, T. M. Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts, *6th European Conference on Games-based Learning (ECGBL)*, pp. 4-5, Cork, Ireland, October, 2012.
- [24] Wing, J. Computational Thinking. *Communications of the ACM*, 3 ed.:33-35, March, 2006.
- [25] Wolz, U., Hallberg, C., & Taylor, B. Scrape: A tool for visualizing the code of Scratch programs. Poster presented at the *42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, 2011.