LOYOLA UNIVERSITY CHICAGO 1870 AD MAJOREM DEI GLORIAM eCOMMONS

Loyola University Chicago
Loyola eCommons

Computer Science: Faculty Publications and Other Works

Faculty Publications and Other Works by Department

3-2019

An Examination of the Correlation of Exploring Computer Science Course Performance and the Development of Programming Expertise

Steven McGee
The Learning Partnership, mcgee@lponline.net

Ronald I. Greenberg
Loyola University Chicago, Rgreen@luc.edu

Randi McGee-Tekula
The Learning Partnership, rmcgee@lponline.net

Jennifer Duck
The Learning Partnership, jenn@lponline.net

Andrew M. Rasmussen
Chicago Public Schools, arasmussen@cps.edu

See next page for additional authors

Follow this and additional works at: https://ecommons.luc.edu/cs_facpubs

Part of the Computer Sciences Commons, Curriculum and Instruction Commons, Educational Assessment, Evaluation, and Research Commons, Science and Mathematics Education Commons, and the Secondary Education Commons

Author Manuscript

This is a pre-publication author manuscript of the final, published article.

Recommended Citation

Steven McGee, Ronald I. Greenberg, Randi McGee-Tekula, Jennifer Duck, Andrew M. Rasmussen, Lucia Dettori, and Dale F. Reed. An examination of the correlation of Exploring Computer Science course performance and the development of programming expertise. In SIGCSE '19, pages 1067--1073. Association for Computing Machinery, March 2019.

This Article is brought to you for free and open access by the Faculty Publications and Other Works by Department at Loyola eCommons. It has been accepted for inclusion in Computer Science: Faculty Publications and Other Works by an authorized administrator of Loyola eCommons. For more information, please contact ecommons@luc.edu.

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License.

Authors

Steven McGee, Ronald I. Greenberg, Randi McGee-Tekula, Jennifer Duck, Andrew M. Rasmussen, Lucia Dettori, and Dale F. Reed

This article is available at Loyola eCommons: https://ecommons.luc.edu/cs_facpubs/220

An Examination of the Correlation of Exploring Computer Science Course Performance and the Development of Programming Expertise

Steven McGee
The Learning Partnership
Chicago, Illinois
mcgee@lponline.net

Ronald I. Greenberg
Loyola University Chicago
Chicago, Illinois
rig@cs.luc.edu

Randi McGee-Tekula
The Learning Partnership
Chicago, Illinois
rmcgee@lponline.net

Jennifer Duck
The Learning Partnership
Chicago, Illinois
jenn@lponline.net

Andrew M. Rasmussen
Chicago Public Schools
Chicago, Illinois
arasmussen@cps.edu

Lucia Dettori
Chicago Public Schools
Chicago, Illinois
ldettori@cps.edu

Dale F. Reed
University of Illinois at Chicago
Chicago, Illinois
reed@uic.edu

ABSTRACT

This study investigated patterns in the development of computational thinking and programming expertise in the context of the Exploring Computer Science (ECS) program, a high school introductory CS course and professional development program designed to foster deep engagement through equitable inquiry around CS concepts. Prior research on programming expertise has identified three general areas of development — program comprehension, program planning, and program generation. The pedagogical practices in ECS are consistent with problem solving approaches that support the development of programming expertise. The study took place in a large urban district during the 2016–17 school year with 28 ECS teachers and 1,931 students. A validated external assessment was used to measure the development of programming expertise. The results indicate that there were medium-sized, statistically significant increases from pretest to posttest, and there were no statistically significant differences by gender or race/ethnicity. After controlling for prior academic achievement, performance in the ECS course correlated with performance on the posttest. With respect to specific programming concepts, the results also provide evidence on the progression of the development of programming expertise. Students seem to develop comprehension and planning expertise prior to expertise in program generation. In addition, students seem to develop expertise with concrete tasks prior to abstract tasks.

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 the author(s) 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.
SIGCSE '19, February 27–March 2, 2019, Minneapolis, MN, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287415

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**; **Student assessment**; *Model curricula*; *K-12 education*.

KEYWORDS

high school computer science; Exploring Computer Science; computer programming skills

ACM Reference Format:
Steven McGee, Ronald I. Greenberg, Randi McGee-Tekula, Jennifer Duck, Andrew M. Rasmussen, Lucia Dettori, and Dale F. Reed. 2019. An Examination of the Correlation of Exploring Computer Science Course Performance and the Development of Programming Expertise. In *SIGCSE '19: 50th ACM Technical Symposium on Computer Science Education, February 27–March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287415

1 INTRODUCTION

"*Our economy is rapidly shifting, and both educators and business leaders are increasingly recognizing that computer science (CS) is a 'new basic' skill necessary for economic opportunity and social mobility.*" — White House Blog [24]

With President Obama's announcement of the Computer Science For All Initiative in 2016, there has been a surge in the number of districts that are planning for or newly implementing computer science (CS) offerings at their schools. Since the dawn of the modern, standards-based era [19], this is the first time that a new subject area is gaining prominence as a core subject. As seen in the White House blog entry, much of the rhetoric around the Computer Science for All initiative has focused on broadening participation by increasing the number of women and minorities who pursue computer science as a career. Broadening participation is an important focus given the dismal levels of participation of women and minorities in computer science. However, without attending to the structural barriers to participation [14], programs that propagate traditional approaches

to CS education are unlikely to contribute to the goal of broadening participation.

In a review of computing research, Sheard and colleagues [23] identified a variety of strategies with empirical support on their effectiveness at improving computing outcomes. In particular, they highlight the explicit teaching of planning strategies, pair programming, and collaborative learning. In addition, Robins and colleagues [22] summarize empirical support that a problem solving approach is an effective means to develop programming expertise. On the other hand, Robins and colleagues have also documented that despite this evidence, some CS educators believe that such problem-solving, collaborative approaches to CS can 'dumb down' the curriculum.

Pears and colleagues [20] make a plea for more large-scale, systematic research that examines the relationship between teaching techniques and student outcomes. In order to support generalizations across initiatives, it is important for researchers to be explicit about the pedagogical techniques being used and to use a common framework for characterizing student outcomes. In this study, we examine two research questions related to the correlation between a particular problem-solving approach to teaching high school computer science and the development of programming expertise. (a) To what extent does student performance in the Exploring Computer Science course correlate with performance on the end of course exam? (b) Which aspects of programming expertise are the most difficult for students to develop? In the next section, we discuss a prominent high school computer science curriculum that explicitly incorporates the aforementioned empirically supported pedagogies.

1.1 Exploring Computer Science

The Exploring Computer Science (ECS) curriculum and professional development program was developed at the University of California, Los Angeles and the University of Oregon, with the goal of contributing to broadened participation of women and minorities and increased equity in the field of computer science [14]. Specifically, the ECS curriculum seeks to accomplish this goal of broadening participation by introducing the field of computer science and computational practices in a way that makes the field relevant, engaging, and stimulating for a diverse population of students. The ECS curriculum is composed of activities that are designed to engage students in computer science inquiry around meaningful problems; the ECS professional development program is designed to prepare teachers to implement these inquiry-based activities while also guiding teachers in building a classroom culture that's culturally relevant and inclusive of all students.

When computer science is not taught for deep engagement, but rather as an abstract academic subject, it privileges access to mostly Caucasian, male students [14]. To play an integral role in such classrooms, students must master abstract programming for programming's sake. In contrast, the ECS curriculum is designed to engender deep engagement with important computer science concepts using important features of communities in which youths participate outside the classroom. General technology use outside of school by youths of all races and genders tends to revolve around making social connections and working on practical problems [12]. As suggested by the research reviews cited above, reorienting computer science instruction to focus on problem-solving experiences that are meaningful to students has the potential to increase access to computer science content. In addition, collaborative learning and paired programming techniques create opportunities for students to learn from each other.

At the core of ECS are a set of high-leverage teaching practices [9] that support the three interwoven teaching strands of ECS: equity, inquiry, and CS concepts. The following high-leverage teaching practices enable students to equitably participate in student-led inquiry around important CS concepts: (a) provide a meaningful context for learning; (b) scaffold the development of CS concepts; (c) facilitate peer inquiry and collaboration; and (d) encourage multiple forms of expression [7, pp. 7–8]. Inclusiveness is supported by focusing on ideas that are meaningful to students, and activities in the curriculum provide space for teachers to incorporate students' background and culture. In addition, many activities focus on real-life issues in the community; for example, students can make games that communicate messages about healthy eating or about the plight of undocumented students [15]. Resting on equity are inquiry-based activities in which students are "expected and encouraged to help define the initial conditions of problems, utilize their prior knowledge, work collaboratively, make claims using their own words, and develop multiple representations of particular solutions" [15]. By engaging students in equitable inquiry through the first two strands, students gain access to the domain content of computer science, the third strand.

1.2 Computer Science Content in ECS

For this study, we will use the context of ECS to investigate whether the ECS problem-solving approach correlates with the development of programming expertise. We will also contribute to the literature by engaging in a large-scale, systematic study of ECS implementation. We focused on the development of programming expertise in the context of students in the Chicago Public Schools (CPS) who are participating in an ECS course. CPS is the first school district in the United States to enact computer science as a high school graduation requirement. ECS is a primary course through which students fulfill the requirement. This research has been conducted in the context of the Chicago Alliance For Equity in Computer Science (CAFÉCS), which is an ongoing researcher-practitioner partnership between CPS, The Learning Partnership, DePaul University, Loyola University, and the University of Illinois at Chicago [3, 4]. (Some important prior research results produced by the alliance report on the impact of the ECS course on students' attitudes towards computer science [2], students' choices about future CS coursework [16], and students' development of computational thinking practices [17].)

As recommended by the research review of Robins and colleagues [22], the ECS course first introduces students to general problem solving that is abstracted from any specific language. In the subsequent unit on programming, the ECS sequence generally follows the 'chain of cognitive accomplishments,' involving three general phases as described by Linn and Dalbey [13]. (a) Students first develop comprehension of specific language features by studying their usage within specific case examples. For example,

to investigate the concept of conditionals, the teacher engages students in a game of asking them to stand up if something is true, such as if a student is a girl AND wearing a blue shirt. Students are able to develop an appreciation of the feature of a language before engaging in the syntax of a language. (b) Next, the students focus on planning solutions through problem solving scenarios. For example, students apply their knowledge of conditionals to develop a Rock, Paper, Scissors game. They use pseudocode to plan out the possible winning scenarios based on combinations of rock, paper, and scissors. (c) Lastly, the programming unit culminates in students generating code to develop a programming project of their own choosing. The students define the problem, plan out a solution, and then use the features of the programming language to implement their solution.

The development of expertise in comprehending programs, planning, and generation of programs has been studied in the literature [22]. The bulk of this research focused on comprehension of programs — examining the extent to which students demonstrate understanding of an existing program. Research on planning highlights the need for students to organize their thinking prior to translating program specifications into program code. Less research has been done on program generation in which students create part of or a whole program to meet a set of criteria.

These "cognitive aspects of children and novices learning computational concepts were studied extensively in the 1980s" [10, p. 42] but have received less attention since then [10]. Research on planning has shown that students generally spend very little time planning, which suggests a need to focus explicitly on supporting students' planning efforts. Students are able to reason about surface features of a problem, but have difficulty envisioning the unknowns of given problems. In the area of program generation, research has shown that students have particular difficulty with conditionals and loops [18, 21]. In general, educators need to be realistic about what can be accomplished within an introductory CS course.

2 MEASUREMENT OF PROGRAMMING EXPERTISE

To measure the development of programming expertise, we used assessments that were aligned to the computational thinking concepts in ECS [6]. The assessments were developed and field tested by SRI International over two years using Evidence-Centered Design (ECD), an assessment methodology that is especially advantageous when the knowledge and skills to be measured involve complex, multistep performances. The ECD process involved (1) working with various stakeholders to identify the important computer science skills to measure, (2) mapping those skills to a model of evidence that can support inferences about those skills, and (3) developing tasks that elicit that evidence. The assessments were field tested with 941 students over two years [6]. Separate pretest and posttest forms were created. The pretest contains six tasks that measure students' initial understanding of CS concepts and computational thinking. Across the six tasks there are a total of 19 subtasks that are scored independently. The posttest contains five tasks, two of which were on the pretest and three of which were different. The two common tasks were used to equate the two forms and allow for measurement of growth from pretest to posttest. SRI developed scoring rubrics with student work examples for each of the tasks. Across all of the pretest and posttest tasks, there are a total of 30 question prompts that are each scored individually. The assessments and scoring rubrics can be accessed from the SRI assessment website [11]

A subset of the tasks were used for this analysis to examine the development of student expertise in comprehension, planning, and generation. In keeping with the recommendation of being realistic about the expectations for students, the assessment tasks targeted narrow aspects of each ability that were aligned with the expectations of the ECS curriculum.

Within program comprehension, there were two tasks that were assessed. First students were provided with a written algorithm. They were given an input value and asked to determine what the program output would be. Next students were given a scenario in which three drivers with cars needed to pick up 12 passengers for a concert and no more than five people could be in one car. Given an approach to efficiently picking up the passengers, students need to determine whether the algorithm will meet the criteria.

Under planning, students used the same driver and passenger scenario. They were asked to determine which inputs were provided in the scenario and which important inputs were not provided in the scenario. In a separate task, students were asked to provide requirements for a program that would help a teacher track student information.

For program generation, students were provided with an algorithm that is abstracted from any given language. They are then asked to decide which step in the algorithm would use a Scratch conditional block and which step would use a Repeat-Until Loop. Based on the chain of cognitive accomplishments cited above, we hypothesized that comprehension would be the least complex, followed by planning and then generation. As discussed below, this hypothesis about levels of complexity of the tasks was tested using the Rasch scaling method to estimate the relative level of difficulty of each set of tasks.

3 METHODS

This study was undertaken by CAFÉCS in the context of the implementation of ECS in CPS. There were 90 teachers who taught an ECS course during the 2016–17 school year to 6,425 students. The school district invited all of the teachers to administer the pretests at the beginning of the school year and the posttests at the end of the school year. There were 28 teachers with 1,931 students who administered both the pretests and posttests to their students. These teachers and their students were included in the study. The remaining ECS teachers were dropped from the study since they provided only partial data or no data.

Table 1 shows the demographic characteristics of the students who were included in the study, and, for comparison, the demographic characteristics of the remaining ECS students as well as the total district high school population. Statistical comparisons of each demographic characteristic were conducted between the ECS research participants and ECS non-participants. Asterisks indicate those demographic characteristics in which there was a statistically significant difference between the research participants and non-participants.

Table 1: Demographic characteristics of ECS research participants in comparison to non-research ECS students and all district high school students. An asterisk in the research sample column indicates that value is statistically significantly different from the ECS sample.

Characteristics	District	Other ECS	Research Sample
Number of students	**109,053**	**4494**	**1931**
Female	—	45%	42%
Caucasian	8%	12%	15%*
African-American	40%	29%	20%*
Hispanic	46%	46%	58%*
Asian	4%	8%	5%
Low Income	83%	86%	82%*
Special Ed	16%	13%	14%
ESL	8%	9%	9%
Freshman	25%	66%	71%*
Attendance	88%	92%	91%
GPA	—	2.7	2.7

In general, there were fewer females than males who completed the ECS course, but there was no statistical difference between research participants and non-participants. The largest racial demographic group in ECS were Hispanic students followed by African-American students. In the research sample, there was a larger percentage of Hispanic and Caucasian students and a lower percentage of African-American students than the non-research participants. The research population had a lower percentage of low-income students. The proportion of special education, and English language learners was similar as was the rates of attendance and the overall GPA. Given the large sample size, there were sufficient numbers of students in each demographic category to be able to investigate differences in outcomes based on gender and race/ethnicity.

3.1 ECS Professional Development

Curriculum materials and activities represent one component of the ECS program. Implementation of ECS was supported by a robust professional development program. Given the significant shift in the nature of computer science teaching required for successful implementation of ECS, teachers need extended professional development to successfully adapt to the ECS model of teaching [8]. The ECS PD program is intentionally designed to prepare teachers to implement the inquiry-based activities while also guiding them to build a classroom culture that is inclusive of all students [8]. Professional development begins with a weeklong summer workshop prior to implementing ECS. There are five key components of the ECS professional development model, the first being that teachers engage in the process of collaborative inquiry in small groups in the same way that students will engage in inquiry. The second component is that, throughout the first week, teachers participate in inquiry specifically through a teacher-learner-observer model. Each small group is assigned a lesson in which the group co-plans and teaches the lesson to the rest of the participants, who experience the lesson as learners. After the lesson, all the participants engage in reflective discussion about the experience from the point of view of the three ECS teaching strands (equity, inquiry, and CS content). These first two components of ECS professional development are consistent with what Desimone and Garet [1] call active learning in professional development. Their review of professional development found that active learning was an important component of professional development as it significantly influenced changes in teacher practices.

The third component of ECS professional development is explicit discussion and reflection on equitable practices. During the workshop, the teachers read sections of Stuck in the Shallow End [14], which provides rich case study descriptions of the roots of inequity in computer science. The fourth and fifth components of ECS PD are meant to sustain teacher development over long time spans, which is another key dimension of effective PD [1]. The fourth component is ongoing professional development during the school year and a second weeklong workshop the summer after their first year of implementation. The fifth component of ECS PD is the development of a professional learning community. It begins in the summer workshop through the formation of small groups that engage in collaborative inquiry. It is also built up through the trust that teachers develop as they engage in tough, open discussions about equity as well as through open, honest feedback on lesson design and implementation during the workshops.

3.2 Assessments

During the 2016–17 school year, teachers administered the SRI-developed ECS pretest at the beginning of the year and the posttest at the end of the year. SRI has developed rubrics for each of the assessment tasks. These rubrics are designed for classroom teachers to grade their students' assessments. SRI reports that it takes teachers about 5 minutes per student to score the assessments. In order to aggregate assessment result across teachers, we used independent scorers to grade the assessments.

We hired The Graide Network to score the pretests and posttests. The Graide Network recruited and trained 26 undergraduate preservice teachers to score the performances tasks. The scorers were provided training on each of the rubrics prior to scoring. As part of the training, each scorer scored a common set of 80 pretest responses from each question prompt in order to equate the severity of the scorers. For the posttest, we had overlapping subsets of scorers rate the same students. We used the Facets software version 3.71.4 to conduct Many-Facet Rasch Measurement analysis (MFRM) [5] to scale the student responses at each administration. Facets develops a model based on how well the student performed across the range of question prompts with set difficulties taking into account the severity of the scorer relative to the other scorers. Within MFRM, the goal is not for scorers to arrive at agreement on the scores, but instead to model the variation in how the scorers interpreted the rubrics. As long as the raters are internally consistent in how they apply the rubric, Facets can adjust the students' scores based on the severity or leniency of the scores relative to other scorers. We used the pretest tasks as the benchmark for scaling item difficulty. For scaling of the posttest scores, the item difficulties of the two common tasks were fixed based on the pretest scales. The overall model fit of the students, tasks, and scorers at each administration

was high. For ease of interpretation, the logit scale produced by Facets was converted to a scale ranging from 0 to 25.

4 RESULTS

To set the context for investigating the development of specific aspects of programming expertise, we first examine the overall pretest to posttest performance as well as examine the extent to which students' performance in the course predicts posttest performance.

In the first model we test the growth of computational thinking from pretest to posttest. The average pretest score was 11.7 out of 25 and the average posttest score was 13.8 for a growth of more than two points. We used a paired t-test to determine that this growth was statistically significant (t(1930)=24.5, $p < 0.001$) with a medium effect size of 0.6 standard deviations, adjusted for the correlation between the pretest and posttest.

In the second model, we investigated the extent to which students' course performance correlates with the development of computational thinking after controlling for student characteristics. Since students were nested within teachers, we conducted hierarchical linear modeling (HLM) on the posttest performance using WHLM software version 7.24q. There are two levels to the HLM model (see Figure 1 for the HLM equation). Given that students of a particular teacher have a shared experience, the HLM analyses account for this shared variance of students within a class by developing a linear model of each student characteristic for the population of students of a given teacher. HLM then aggregates the intercepts and slopes across all of the teachers to model the relationships of each variable to the posttest performance.

At the first level are the student characteristics, which include each student's pretest score, grade level, gender (female), race (African American or Hispanic versus other races), participation in special education, participation in free or reduced lunch program (FRL), which serves as an indicator of low-income status, participation in the English language learning program (ELL), annual attendance rate, cumulative GPA in the year in which the student completed ECS, and grade the student received in the ECS course. The pretest score, attendance, cumulative GPA, and ECS course grade are group mean centered. The level 1 random effect is represented by r_{ij}. At the second level are the teachers. There are no teacher characteristics included in the model. The pretest score coefficient and ECS course grade coefficient are random effects, represented as u_{1j} and u_{11j}, respectively. These random effects allow the slopes of the two coefficients to vary across teachers at level 2. All of the other factors are fixed effects. After controlling for student characteristics, the HLM analyses provide evidence on the extent to which performance in the ECS course correlated with performance on the posttest.

We used students' course grades as an indicator of course performance. In addition, we examined whether there were differences in posttest performance by students of different gender and racial/ethnic backgrounds. Table 2 shows the results of the analysis. After controlling for pretest performance, there were no statistically significant differences in posttest performance by gender, race/ethnicity or level of family income. There were statistically significant negative differences in posttest performance for ELL and special education students. After also controlling for students' overall academic performance as measured by their GPA as well as their school attendance, how well students performed in the course had a statistically significant correlation with posttest performance.

5 DISCUSSION

Given that students significantly increased their overall performance from pretest to posttest and their posttest performance was correlated with performance in the ECS course after controlling for demographic and other behavioral characteristics, we can now examine the results of students' performance on the specific set of programming tasks on the posttest assessment. The process of Rasch scaling provides an estimate of each student's cognitive ability related to the computer science concepts within ECS as measured by the assessments. In addition, the process of Rasch scaling provides an estimate of the difficulty of each task for the population of assessment takers. In the Rasch context, a student is considered competent at a task if the student's ability is greater than or equal to the difficulty of the task. Table 3 shows the percentage of students at pretest and posttest whose estimated Rasch ability level is greater than or equal to the level of difficulty of the tasks in each category of programming competency.

Across all categories, there were more students who achieved competency at posttest than at pretest. Generally the tasks related to comprehension and planning were easier than the tasks related to program generation as indicated by the fact that there is a higher percentage of students whose ability level is greater than or equal to the difficulty level of those tasks. These results are consistent with the 'chain of cognitive accomplishments,' in which tasks related to language features and planning are easier than tasks related to program generation. In addition, tasks that were concrete in nature were generally easier with most students demonstrating competency by posttest. These tasks include determining the output from a written algorithm, identifying the available inputs from a problem scenario, and deciding the general requirements for a teacher's program to track student information. Tasks that were more abstract in nature were generally more difficult, with less than half of the students demonstrating competency on those tasks. These tasks include testing an algorithm against the specifications, identifying inputs that are not given in the problem scenario, and use of loops and conditionals.

6 CONCLUSIONS

School districts across the United States are responding to the call to increase access to computer science for all high school students. It is important for school districts to be mindful of the research on effective practices for the development of computer science expertise as well as research on setting reasonable expectations for student development of expertise.

In this research, we examined the impact of ECS on the development of programming expertise in the context of a large-scale implementation. ECS is a prominent high school introductory compute science course that is closely aligned to effective pedagogical practices that have theoretical and empirical support. A primary goal of the ECS curriculum and professional development program is to contribute to broadened participation of women and minorities and increased equity in the field of computer science. The

$$
\begin{aligned}
POSTTEST_{ij} \quad = \quad & \gamma_{00} + \gamma_{10} * PRETEST_{ij} + \gamma_{20} * FEMALE_{ij} + \gamma_{30} * GRADE\_LEVEL_{ij} + \gamma_{40} * HISPANIC_{ij} \\
& + \gamma_{50} * AFRICAN - AMERICAN_{ij} + \gamma_{60} * IS\_SPECIAL\_ED_{ij} + \gamma_{70} * IS\_FRL_{ij} + \gamma_{80} * IS\_ELL_{ij} + \gamma_{90} * ATTENDANCE_{ij} \\
& + \gamma_{100} * GPA_{ij} + \gamma_{110} * ECS\_GRADE_{ij} + u_{0j} + u_{1j} * PRETEST_{ij} + u_{11j} * ECS\_GRADE_{ij} + r_{ij}
\end{aligned}
$$

Figure 1: HLM model equation.

Table 2: HLM Model results for the student posttest scores by student characteristics. The factors in bold are statistically significant.

Characteristic	Coefficient	Standard Error	t-ratio	p-value
Average	**14.74**	**1.0**	$t(27) = 14.69$	$p < 0.001$
Student Characteristics				
Pretest	**0.15**	**0.03**	$t(27) = 4.97$	$p < 0.001$
Grade Level	−0.08	0.09	$t(221) = -0.83$	$p = 0.407$
Female	−0.17	0.13	$t(252) = -1.34$	$p = 0.181$
Hispanic	0.26	0.18	$t(183) = 1.44$	$p = 0.151$
African-American	−0.25	0.23	$t(287) = -1.08$	$p = 0.283$
Free or Reduced Lunch	−0.04	0.09	$t(177) = -0.51$	$p = 0.611$
ESL	**−0.66**	**0.24**	$t(106) = -2.78$	$p = 0.007$
Special Education	**−1.05**	**0.18**	$t(628) = -5.68$	$p < 0.001$
Rate of Attendance	**−2.59**	**0.89**	$t(39) = -2.95$	$p = 0.005$
Cummulative GPA	**0.72**	**0.15**	$t(61) = 4.70$	$p < 0.001$
ECS Course Grade	**0.26**	**0.13**	$t(27) = 2.04$	$p = 0.051$

Table 3: Percentage of students demonstrating competency on each dimension of programming expertise at pretest and posttest.

Problem type	Pretest	Posttest
Comprehension		
Determine Output	62%	87%
Will Output Meet Requirement?	14%	34%
Planning		
Available Inputs	46%	78%
Generate Requirements	38%	69%
Missing Inputs	20%	45%
Generation		
Conditional	22%	48%
Loop	7%	21%

curriculum is composed of activities that are designed to engage students in computer science inquiry around meaningful problems in the context of a classroom culture that's culturally relevant and inclusive of all students.

We conducted a large-scale, systematic study, as recommended in the literature [20], to examine whether a curriculum that uses a problem solving approach can support the development of programming expertise. Overall, students achieved medium-sized, statistically significant learning gains from pretest to posttest and those learning gains were spread equitably across gender and race/ethnicity. These learning gains were correlated with students' academic performance in the course after controlling for students' prior academic performance. These results provide evidence that a problem solving approach can support the development of programming expertise.

These results also provide evidence on the progression of the development of programming expertise. Students seem to develop comprehension and planning expertise prior to expertise in program generation. In addition, students seem to develop expertise with concrete tasks prior to abstract tasks. These results provide evidence for the current sequencing of activities in ECS that are consistent with the 'chain of cognitive accomplishments' approach.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under grant numbers CNS-1842085, CNS-1738572, CNS-1738776, CNS-1738691, CNS-1738515, DRL-1640215, CNS-1542971, and CNS-1543217. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Laura M. Desimone and Michael S. Garet. 2015. Best Practices in Teachers' Professional Development. *Psychology, Society, and Education* 7, 3 (2015), 252–263.

[2] Lucia Dettori, Ronald I. Greenberg, Steven McGee, and Dale Reed. 2016. The Impact of The Exploring Computer Science Instructional Model in Chicago Public Schools. *Computing in Science & Engineering (Special Issue: Best of RESPECT 2015)* 18, 2 (March/April 2016), 10–17. https://doi.org/10.1109/MCSE.2016.39.

[3] Lucia Dettori, Ronald I. Greenberg, Steven McGee, Dale Reed, Brenda Wilkerson, and Don Yanek. 2018. CS as a Graduation Requirement: Catalyst for Systemic Change. In *SIGCSE '18*. Association for Computing Machinery, 406–407. https://doi.org/10.1145/3159450.3159646.

[4] Lucia Dettori, Don Yanek, Helen Hu, and Dennis Brylow. 2018. The role of researcher-practitioner partnerships in CS4All: Lessons from the field. In *SIGCSE '18*. Association for Computing Machinery, 674–5. https://doi.org/10.1145/3159450.3159626.

[5] Thomas Eckes. 2011. *Introduction to Many-Facet Rasch Measurement*. Peter Lang.

[6] Exploring Computer Science. 2018. Where is ECS? http://www.exploringcs.org/for-teachers-districts/ecs-now. Accessed April 4, 2018.

[7] Joanna Goode and Gail Chapman. 2016. Exploring Computer Science (Version 7.0). http://www.exploringcs.org/curriculum.

[8] Joanna Goode, Jane Margolis, and Gail Chapman. 2014. Curriculum is Not Enough: The Educational Theory and Research Foundation of the Exploring Computer Science Professional Development Model. In *SIGCSE '14*. Association for Computing Machinery, 493–498.

[9] Pam Grossman, Karen Hammerness, and Morva McDonald. 2009. Redefining teaching, re-imagining teacher education. *Teachers and Teaching: Theory and Practice* 15, 2 (2009), 273–289.

[10] Shuchi Grover and Roy D. Pea. 2013. Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher* 42, 1 (2013), 38–43.

[11] SRI International and Principled Assessment of Computational Thinking (PACT). 2018. PACT | Resources. https://pact.sri.com/ecs-assessments.html. Last accessed 12/1/2018.

[12] Mizuko Ito, Heather Horst, Matteo Bittanti, danah boyd, Becky Herr-Stephenson, Patricia G. Lange, C. J. Pascoe, and Laura Robinson. 2008. Living and Learning with New Media: Summary of Findings from the Digital Youth Project. John D. and Catherine T. MacArthur Foundation Reports on Digital Media and Learning. http://www.macfound.org/media/article_pdfs/DML_ETHNOG_WHITEPAPER.PDF.

[13] Marcia C. Linn and John Dalbey. 1989. Cognitive Consequences of Programming Instruction. In *Studying the novice programmer*, James C. Spohrer and Elliot I. Soloway (Eds.). Lawrence Earlbaum Associates, Chapter 4, 57–81.

[14] Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kimberly Nao. 2010. *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press.

[15] Jane Margolis, Jean J. Ryoo, Cueponcaxochitl D. M. Sandoval, Clifford Lee, Joanna Goode, and Gail Chapman. 2012. Beyond Access: Broadening Participation in High School Computer Science. *ACM Inroads* 3, 4 (Dec. 2012), 72–78. https://doi.org/10.1145/2381083.2381102.

[16] Steven McGee, Randi McGee-Tekula, Jennifer Duck, Ronald I. Greenberg, Lucia Dettori, Dale F. Reed, Brenda Wilkerson, Don Yanek, Andrew M. Rasmussen, and Gail Chapman. 2017. Does a Taste of Computing Increase Computer Science Enrollment? *Computing in Science & Engineering (Special Issue: Best of RESPECT 2016)* 19, 3 (April 2017), 8–18. https://doi.org/10.1109/MCSE.2017.50.

[17] Steven McGee, Randi McGee-Tekula, Jennifer Duck, Catherine McGee, Lucia Dettori, Ronald I. Greenberg, Eric Snow, Daisy Rutstein, Dale Reed, Brenda Wilkerson, Don Yanek, Andrew M. Rasmussen, and Dennis Brylow. 2018. Equal Outcomes 4 All: A Study of Student Learning in ECS. In *SIGCSE '18*. Association for Computing Machinery, 50–55. https://doi.org/10.1145/3159450.3159529.

[18] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2013. Learning computer science concepts with Scratch. *Computer Science Education* 23, 3 (2013), 239–264. https://doi.org/10.1080/08993408.2013.832022.

[19] National Commission on Excellence in Education. 1983. A nation at risk: The imperative for educational reform. US Department of Education. http://www2.ed.gov/pubs/NatAtRisk.

[20] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin* 39, 4 (Dec. 2007), 204–223. http://doi.acm.org/10.1145/1345375.1345441.

[21] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Computing Education* 18, 1 (Oct. 2017), 1:1–1:24. http://doi.acm.org/10.1145/3077618.

[22] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200.

[23] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. 2009. Analysis of Research into the Teaching and Learning of Programming. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*. 93–104. http://doi.acm.org/10.1145/1584322.1584334.

[24] Megan Smith. 2008. Computer Science For All. White House Blog. https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all.