IJCSES

Volume 2, Issue 4

January 2019

ISSN 2513-8359

International Journal of Computer Science Education In Schools

Editors

Prof. Filiz Kalelioglu

Yasemin Allsop

www.ijcses.org

International Journal of Computer Science Education in Schools

January 2019, Vol 2, No 4

DOI: 10.21585/ijcses.v2i4

Table of Contents

Estimating the Effect of a Teacher Training Program on Advanced Placement® Outcomes

Richard S. Brown[1]
Emily Anne Brown[2]

[1] West Coast Analytics
[2] University of North Texas

DOI: 10.21585/ijcses.v2i4.35

Abstract
This study employs a potential outcomes modeling approach to estimate the effect of Code.org's Professional Learning Program on Advanced Placement (AP) Computer Science Principles test taking and qualifying score earned for a recent cohort of 167 schools compared to a matched group of comparison schools. Results indicate substantial and significant increases in both Computer Science AP test taking and qualifying score earning for all students. In addition, the significant effects were even greater for Computer Science AP test taking and qualifying score earned by female and minority students when impact ratios are analyzed separately. This study provides evidence of a teacher training program that is having a significant and important impact on preparing more students to succeed in computer science and improve the future of computer science education in this country.
Keywords: computer science, professional development, teacher training

1. Introduction

Despite the growing need for qualified workers in STEM fields, there remains a significant under-representation of females in STEM fields (Beede, et al., 2011) and specifically in Computer Science careers (Sax, et al., 2017). Similar gaps exist for minority students. Research has shown that targeted training of teachers to provide Computer Science courses can increase the number of minority students enrolled in advanced Computer Science courses (Goode, 2007). Goode argues that there is a critical need to provide professional development to support and encourage minority participation in Computer Science coursework. This study employs a potential outcomes modeling approach to estimate the causal effect of Code.org's Professional Learning Program.

Code.org, a nonprofit 501(c) (3), works across the education spectrum to expand access to computer science and increase participation by women and underrepresented minority populations in computer science coursework. Code.org believes that every student in every school should have the opportunity to learn computer science, just like biology, chemistry or algebra. In addition to developing curricula for grades K-12, Code.org provides professional development for high school educators. The Code.org Professional Learning Program offers both in-person and online support for teachers before and during their first year teaching the Code.org curriculum. To date, several thousand teachers completed the program, with the majority ranking it as among the best professional development of their careers.

The Code.org Professional Learning Program is a multi-pronged approach to ensure the quality and sustainability of the program at scale. The program represents a coordination of three major Code.org efforts -- Regional Partners, Facilitator Development, and Professional Development Workshops -- all built upon the foundations and principles of Code.org curricula which has been designed to meet learning objectives through engagement with equitable classroom practices.

Taken altogether, the Professional Learning Program can be summarized as a year of ongoing Professional Development Workshops for teachers with agendas and activities designed specifically for the Code.org CS Principles Curriculum and teaching philosophies. Workshops are run by Code.org Professional Development Facilitators who receive training in a separate, year-long program devoted to PD leadership development specifically designed to support the Code.org CS Principles Curriculum.

Teachers are supported from the beginning of the program to the end by Code.org Regional Partners who collaborate with facilitators to deliver high quality workshops. Code.org Regional Partners are developed through a multi-year partnership with the aim of building local, sustainable hubs of high quality PD for computer science teachers. Teachers also have additional ongoing supports such as the Code.org Forum, an online professional learning community.

Figure 1. Code.org Professional Learning Program Logic Model

1.1. Goals

The primary goal of the Code.org Professional Learning Program is to support implementation of the Code.org CS Principles Curriculum in schools such that it leads to more students, and a more diverse group of students, taking and earning qualifying scores on the AP Computer Science Principles Exam. Other student goals include generating positive attitudes, self-efficacy, sense of belonging in computer science classrooms, and positive expectation about computer science in their future. A residual outcome would be to increase the number and diversity of students who pursue computing-related opportunities after AP Computer Science Principles, such as taking more computer science classes or seeking employment that requires computer science skills.

The curriculum and associated professional development enable teachers with very little background knowledge in computer science to deliver the course via equitable teaching practices to engage all students. Other goals for teachers include positively affecting teachers' attitudes and self-efficacy toward teaching computer science, as well as their belief-systems about equity in computer science classrooms. The theory underlying these goals is that teachers who engage students with equitable teaching practices coupled with a curriculum rich with resources and activities that support and encourage enactment of those practices will lead to (1) better student learning overall (2) more equitable student engagement and learning.

1.2. Timeline & Implementation

The Code.org Professional Learning Program begins with teachers applying to the program through a Regional Partner starting with January of the year they enter the program. Regional Partners work with Code.org to approve admission to the program based on a number of criteria, the most influential being a stated commitment from the district, or teacher and school principal to offer and teach the course in the upcoming school year. It is important to note that even though the curriculum is designed to support implementation of the AP course, teachers are not required to offer it as an AP course for admission into

the Professional Learning Program. In 2016-17 roughly half of the teachers in the program self-reported that they offered CS Principles as an AP course at their school.

The teacher training begins in earnest during the summer with a five-day in-person workshop in which teachers explore the Code.org curriculum and learning tools, practice and discuss classroom management and teaching strategies, and build a community of educators. Modeled after the "five requirements of transformative learning" outlined in Louckes-Horsley, Stiles, Mundry, Love, & Hewson (2010), a major focus of the professional development program is to practice new teaching strategies as part of workshop activities. In the workshop, teachers deliver lessons from the curriculum to an audience of peers that highlight these teaching practices. Afterward teachers debrief the lesson, allowing them time to reflect with peers about how implementation should be tailored for their own classrooms.
Teachers also reflect on enacting equitable teaching practices in light of the historic inequities faced by underrepresented groups in computer science. The workshops devote time to developing strategies for computer science advocacy and student recruitment strategies with a goal of enrolling students in computer science classes that are representative of their school's population in terms of race, gender, and other demographic factors.

The program continues to support teachers throughout the academic school year though workshops hosted locally by Code.org Regional Partners and run by trained Code.org Professional Development Facilitators. Each academic-year workshop combines further curriculum exploration and planning, and revising goals set during the summer (for example: recruiting and retaining a representative set of students, supporting student needs, assessing student learning, etc.). The workshops focus on elements of the curriculum that are essential for effectively teaching the course, such as exploring new computer science content, developing pedagogical strategies to keep the classroom environment equitable and engaging, and doing AP preparation.

Figure 2. Code.org Timeline of Events

2. Methodology

2.1. Data Sources

Advanced Placement test data from a total of 167 treatment schools from the most recent Code.org cohort plus 167 non-treatment schools were analyzed for this study. Data for treatment and matched comparison schools was provided by the College Board by matching the Code.org treatment schools on the state in which the school is located, total school enrollment, percent of students receiving free or reduced priced lunch, and percentage of minority students at each school. The original list of program schools included 383 schools, of which 167 were matched (43.6%). The lower than anticipated match percentage resulting from stringent matching criteria. Matching criteria required that the comparison school be located in the same state as the treatment school and be within +/- 20% of the total student enrollment of the treatment school. Further, each comparison school must also be within one standard error of the mean of the target treatment school in terms of percentage of minority students and percent of students qualifying for free or reduced priced lunch. Thus, all four criteria had to be met to identify an acceptable comparison school.

2.2. Research Design

This study employs a potential outcomes modeling approach (Rubin, 2005) to estimate the causal effect of program participation on first year improvements in AP test taking and AP qualifying score earning in computer science AP subjects. The potential outcomes model, also called the Rubin Causal Model (RCM) (Holland, 1986), allows for the formal identification for causal inference. This approach estimates the average difference between observed outcomes and potential outcomes (counterfactuals) for each unit in the analysis. This is known as the causal estimand. Potential outcomes modeling has been widely used in a number of social science fields, including education, politics, and public health to estimate causal effects of programs or policies (Glass, Goodman, Hernan, & Samet, 2013; Keele, 2015). In fact, Keele (2015) states, "The RCM is the dominant model of causality in statistics at the moment" (p. 315), while acknowledging there are many other approaches to estimating causality in a statistical framework (e.g., Dawid, 2000; Pearl, 2009).

The goal of propensity score matching within the RCM is to construct a sample of comparison schools that are similar to the treatment schools (Rosenbaum & Rubin, 1985) in terms of their likelihood of selection into treatment. This model has gained popularity in recent years and is frequently used to make causal estimates from observational studies. Rubin (2005) has argued, "the potential outcomes formulation of causal effects, whether in randomized experiments or in observational studies, has achieved widespread acceptance" (p. 329). A propensity score is a scalar value that summarizes the likelihood for a unit to receive a treatment, often based on a large set of variables. In this study, we estimate the propensity score and causal estimands using a weighting approach applied in the Toolkit for Weighting and Analysis of Nonequivalent Groups ("twang") package written in the R programming language (Ridgeway, McCaffrey, Morral, Burgette, & Griffin, 2015).

Previous literature suggests that propensity score models should include all confounding variables, that is, variables that are related to the treatment assignment as well as to the outcome (Rubin, 2007; Rubin & Thomas, 1996; West & Thoemmes, 2010), or all variables that are related to the outcome (Rosenbaum, 2002). Stuart (2010) also argues that one should be generous in including predictors in the propensity score model, because the cost of omitting a variable that might predict the outcome is greater than the cost of including a variable that in fact did not predict the outcome (increase in bias versus slight increase

in standard errors of propensity scores). In this study, school demographic data such as total enrollment, percent minority enrollment, and percent of enrollment qualifying for free or reduced priced lunch provide ample information that may predict the outcomes of this study (i.e., number of students taking Computer Science AP tests and student performance on Computer Science AP tests). Thus, these three variables will be used to balance the treatment and control conditions.

2.3. Data Analytic Approach

The twang approach to propensity score estimation uses generalized boosted models (GBMs), a multivariate nonparametric regression technique, introduced in McCaffrey, Ridgeway, and Morral (2004). This approach is argued to allow for flexible, nonlinear relationships as well as a large number of variables, and shown to perform well under certain settings (see, e.g., Imai & Ratkovic, 2014). In the GBM approach, instead of matching, a weighting approach is used to estimate the treatment effect. One of the advantages of propensity score approaches is that once non-experimental data are used to "design an observational study" the study achieves balance between treatment and control groups as if it were based on an experimental study (Rubin, 2007). Then, the outcome analysis can proceed in the same way as the analysis that would have been done in an experimental study.

However, note that the effects we seek to obtain can either be the average effect of the treatment on the treated (ATT) or the average treatment effect (ATE). Generally, when we use matching strategies based on the estimated propensity scores, we estimate ATT instead of ATE, because we intentionally select and match control group schools that are like treatment schools. However, when we use weighting strategies (as is done with the twang package), depending on weights that are used, either ATT or ATE can be obtained. For this study, we estimated the effects of the program for both ATT and ATE in order to get a sense of not only what the effect of the program was the participating schools, but also what the effect would have been had the program been provided to the control schools as well.

3. Results

The first step in reviewing the results is to check on the extent to which the propensity score weighting approach results in balance across the treatment and control groups in terms of the balancing variables. As mentioned earlier, several variables were used to balance the treatment and control samples. Along with state in which the schools are located, these included: total school enrollment, percentage of students receiving free or reduced priced lunch, and percentage of total student enrollment that are minority students. These variables were chosen as they are predictive of the outcomes of interest in this study. For example, a regression model using total school enrollment, percentage of total enrollment that are minority, and percentage of total enrollment eligible for free or reduced priced lunch significantly predicted total Computer Science (Computer Science A and Computer Science Principles) tests taken at the school; F (3, 333) = 25.12, p<.001, R^2 = .19.

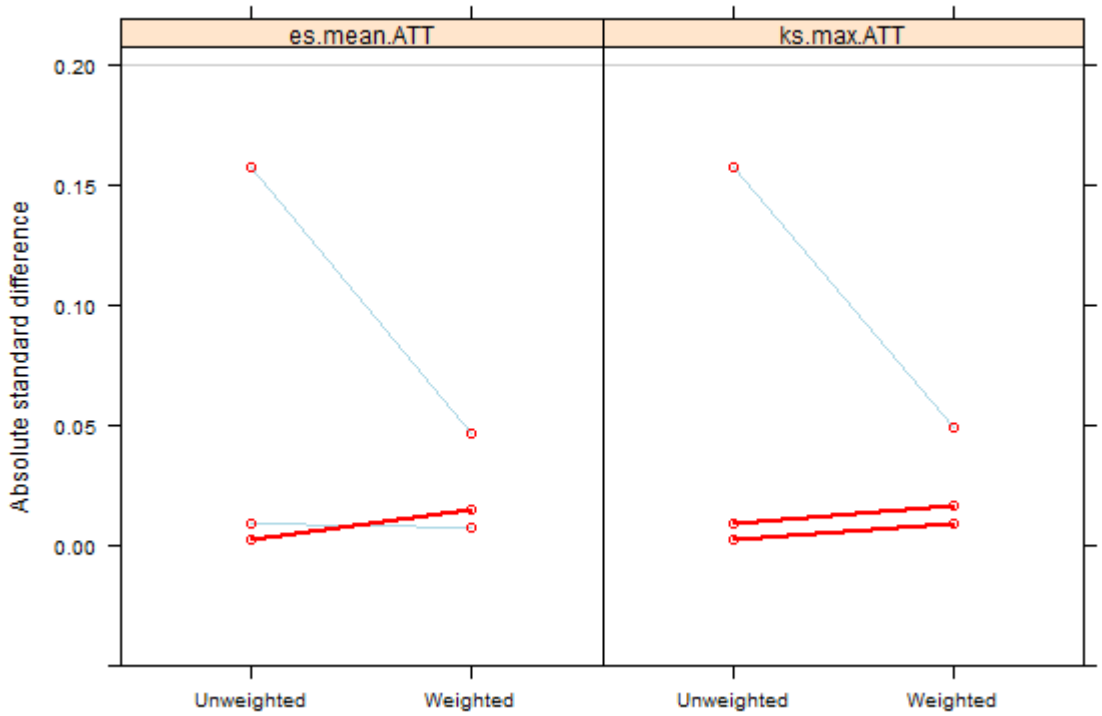


Figure 3. Balance plot for ATT analyses

Treatment and control groups were fairly balanced prior to weighting on total enrollment (M=1354.94 for treatment; M=1221.48 for controls); t(332)=-1.53, p=.128. These minor differences were virtually eliminated through weighting (see Figure 3 for balance plot for ATT analyses). No substantial differences between treatment and control schools existed in percentage of minorities (M=47.53% for treatment; M=47.42% for controls), t(332)=-0.03, p=.977 or for percent of students qualifying for free or reduced price lunch (M=49.56% for treatment; M=49.82% for controls); t(332)=0.09, p=.929. After propensity score weighting (ATT estimation), the treatment and control schools were comparable in terms of all three balancing variables. Specifically, the average total enrollment for the weighted samples was 1354.94 and 1315.26 for treatment and control respectively. Likewise, the average percent minority enrollment was balanced at 47.5 for the treatment schools and 47.0 for the control schools; and the

average percent qualifying for free or reduced priced lunch was 49.6 and 49.8 for treatment and control schools, respectively. Perfect balance is not to be expected. Austin cautions, "as with randomization, one should not expect that perfect balance will be achieved for all measured baseline variables between treated and untreated subjects in the matched sample" (Austin, 2008, p. 2040).

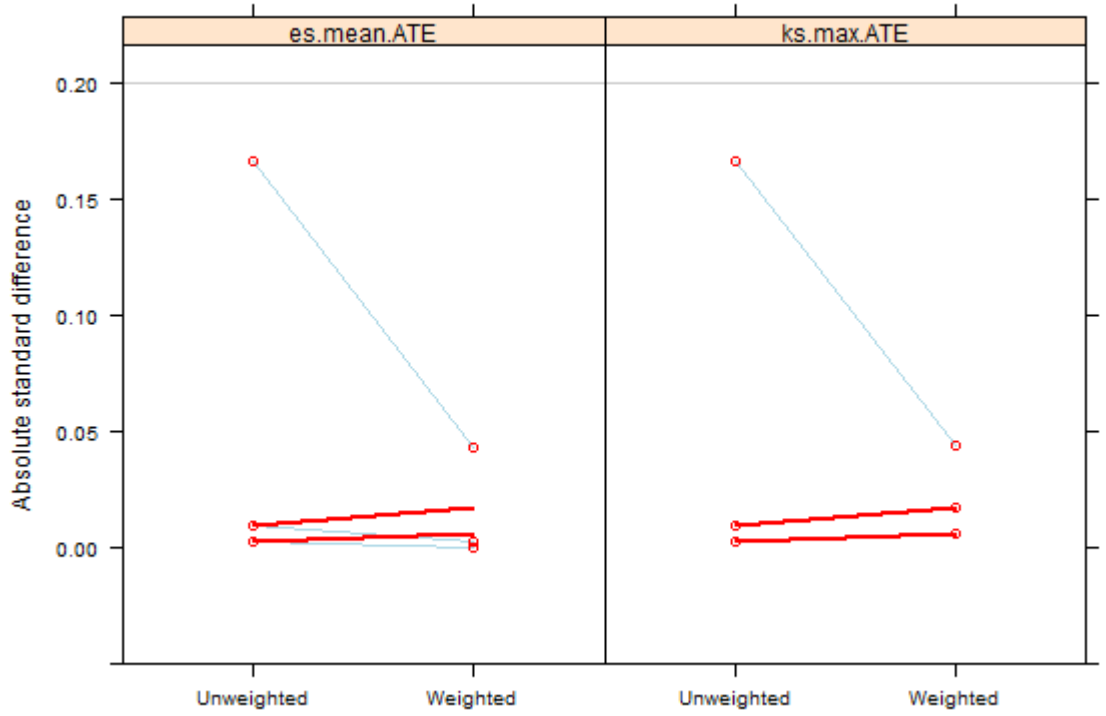


Figure 4. Balance plot for ATE analyses

Treatment and control samples were equally well balanced using the ATE propensity score estimation procedure (see Figure 4). Specifically, for the ATE estimation, the average total enrollment for the weighted samples was 1298.89 and 1263.27 for treatment and control respectively. Likewise, the average percent minority enrollment was balanced at 47.1 for the treatment schools and 47.3 for the control schools; and the average percent qualifying for free or reduced priced lunch was 49.4 and 49.9 for treatment and control schools, respectively. Given the adequately balanced samples with the ATE procedure, we will present the causal estimates from both the ATT and ATE procedures in this report.

Table 1. ATT Estimates for Test Participation by Course

	Estimate	*t* value	*p* value <	Cohen's *d*
All Computer Science				
All Students	17.96	6.72	0.001	0.735
Female Students	5.28	6.07	0.001	0.664
Black Students	1.53	4.08	0.001	0.446
Hispanic Students	5.04	4.95	0.001	0.542
Computer Science Principles				
All Students	16.27	8.03	0.001	0.879
Female Students	5.00	7.10	0.001	0.777
Black Students	1.46	4.18	0.001	0.457
Hispanic Students	4.92	5.62	0.001	0.615
Computer Science A				
All Students	1.69	1.24	**0.215 (NS)**	0.136
Female Students	0.27	0.72	**0.470 (NS)**	0.079
Black Students	0.07	1.07	**0.284 (NS)**	0.117
Hispanic Students	0.12	0.43	**0.668 (NS)**	0.047

The results of the logistic regressions for the average treatment on the treated (ATT) effect are presented in Table 1 above, which shows the impact of the program on average school Computer Science, Computer Science Principles, and Computer Science a Advanced Placement test taking. Table 2 shows the impact of the Code.org program on average number of earned qualifying scores of 3 or better on these same AP tests. Similar analyses were conducted for average treatment effects (ATE), the results of which are provided in Tables 3 and 4.

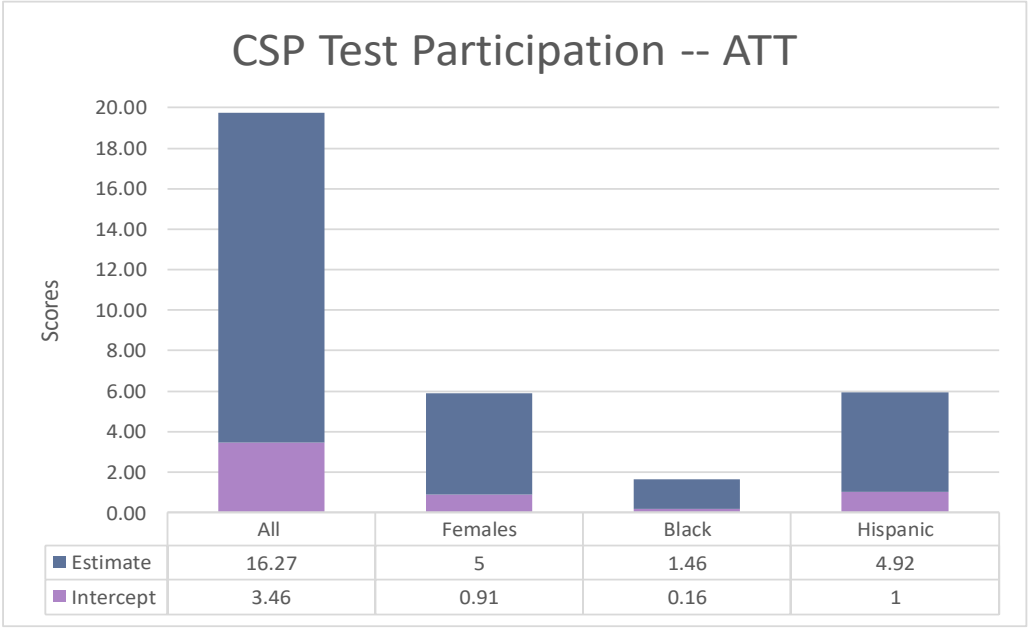


Figure 5. Effect on Computer Science Principles AP Test Participation

As indicated in Table 1, the average number of AP test taking for Computer Science Principles was dramatically higher for all students in the treatment schools following program implementation. On average, participation in the Code.org program generated an average increase of almost 18 additional AP Computer Science tests taken in the 2016-2017 school year; $t(332) = 6.72$, $p < .001$. Moreover, these effects persist when looking at student subgroups. For female students, the increase in Computer Science test taking as a result of program participation is an average of 5.28 tests per school; $t(332) = 6.07$, $p < .001$. For Black students the increase is an average of 1.53 tests; $t(332) = 4.08$, $p < .001$ and for Hispanics it is more than 5 additional tests; $t(332) = 4.95$, $p < .001$. All of the estimates are highly significant statistically, with standardized effect sizes at or above .40 (Cohen's *d*), indicating a moderate to large causal effect of the program on student AP test taking in Computer Science courses. Upon closer inspection, it is clear that virtually all of the effect on increased test participation in Computer Science courses is a function of increasing participation in Computer Science Principles and not in increased participation in Computer Science A, which is consistent with the Code.org model. In fact, there was no discernable impact of program participation on Computer Science A test taking for all students; $t(332) = 1.24$, $p=.215$, female students; $t(332) = 0.72$, $p=.470$, Black students; $t(332) = 1.07$, $p=.284$, or Hispanic students; $t(332) = 0.43$, $p=.668$. In contrast, the effect of the program on Computer Science Principles (CSP) was highly significant for all students and every student subgroup analyzed, thus the effect was not a result of generalized increases in Computer Science participation, but rather a function of targeted Computer Science Principles participation. Moreover, the Cohen's *d* effect sizes ranged from moderate ($d=.46$) to large ($d=.88$).

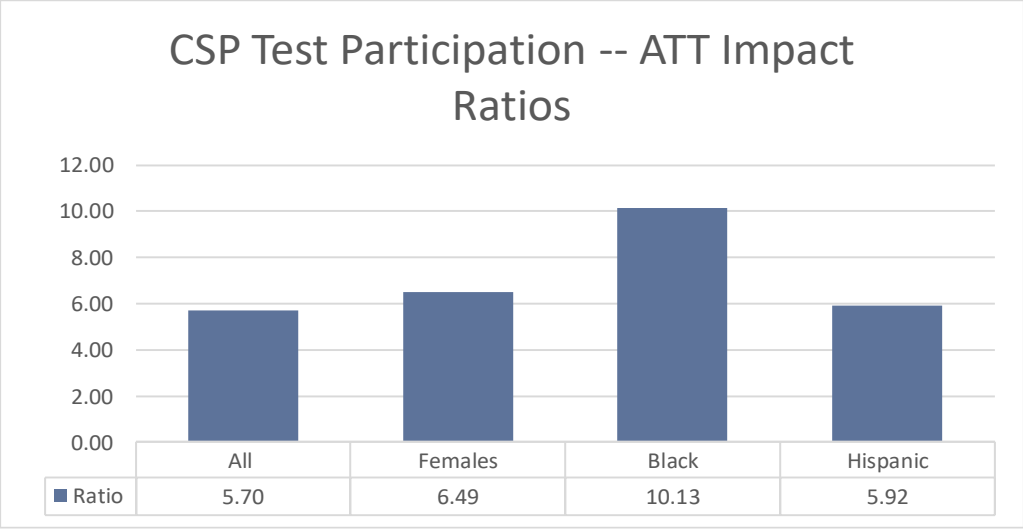


Figure 6. Impact Ratios for Student Subgroups on Computer Science Principles Test Participation

Although the standardized effect size estimates were smaller when viewing minority student test taking effects relative to effects for all students or for female students only, they are nonetheless highly significant and substantial. In fact, Figure 6 shows the impact ratios for Computer Science Principles test taking by student group. This shows that the relative impact is greatest for minority students. Whereas the program effect, in essence, increases test participation for all students by a factor of more than 5, the effect is almost twice that for Black students (10.13). That is to say, the program increased the number of Black students taking Computer Science Principles more than ten-fold on average across the treatment schools. In addition, the program increased the number of Hispanic students taking

Computer Science Principles tests nearly six-fold.

Table 2. ATT Estimates for Qualifying Scores earned by Course

	Estimate	*t* value	*p* value <	Cohen's *d*
All Computer Science				
All Students	11.77	5.92	0.001	0.648
Female Students	3.01	5.31	0.001	0.581
Black Students	0.43	4.02	0.001	0.440
Hispanic Students	2.24	4.96	0.001	0.543
Computer Science Principles				
All Students	10.41	6.73	0.001	0.736
Female Students	2.68	5.91	0.001	0.647
Black Students	0.40	4.14	0.001	0.453
Hispanic Students	2.25	5.37	0.001	0.588
Computer Science A				
All Students	1.36	1.35	**0.179 (NS)**	0.148
Female Students	0.39	1.18	**0.239 (NS)**	0.129
Black Students	0.03	1.03	**0.305 (NS)**	0.113
Hispanic Students	-0.01	-0.05	**0.961 (NS)**	-0.005

Similarly, impressive results were found for program effects on the number of qualifying scores earned in program schools. In addition to increasing the number of students taking Computer Science AP tests, the Code.org program increased the number of qualifying scores earned by students in Computer Science AP courses. Table 2 demonstrates that program schools reported an average of 11.77 more qualifying scores in all Computer Science courses ($t(332) = 5.92$, $p < .001$) and an average of 10.41 more qualifying scores of Computer Science Principles for all students ($t(332) = 6.73$, $p < .001$), both of which were highly statistically significant. Further, as with test taking effects, the impact on qualifying scores was persistent for each student subgroup, with moderate to large effect sizes demonstrated for Computer Science Principles and no discernable effect on the number of qualifying scores earned in Computer Science A.

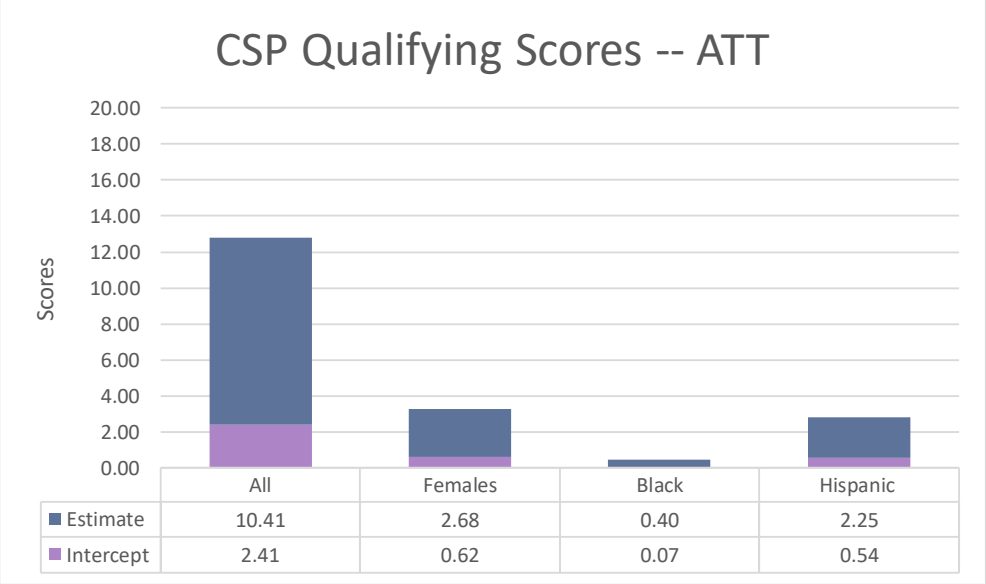


Figure 7. Effect of program on Computer Science Principles qualifying scores earned

Figure 7 shows the impact of participation in the Code.org program on qualifying scores earned in Computer Science Principles in the treatment schools by student subgroup relative to what would have been expected had the program not been implemented in the treatment schools. On average, the program resulted in 2.68 more qualifying scores for female students; $t(332) = 5.91$, $p < .001$, 0.40 more qualifying scores for Black students; $t(332) = 4.14$, $p < .001$, and 2.25 more qualifying scores per school for Hispanic students; $t(332) = 5.37$, $p < .001$. Although these values are smaller compared to the effect for all students, they are nonetheless highly significant substantial effects. The effect sizes for these groups are all in the moderate range (d=.45 to d=.65).

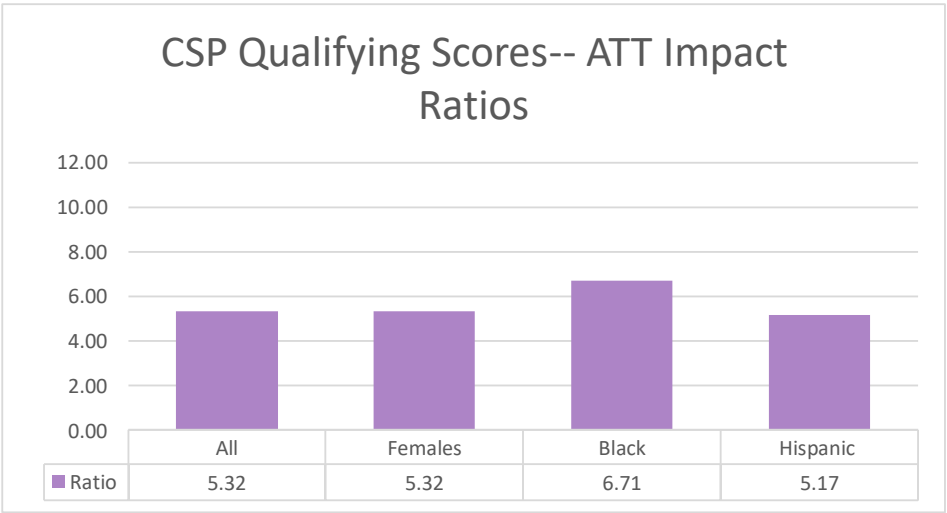


Figure 8. Impact Ratios for Student Subgroups on Computer Science Principles Qualifying Scores

Further, the impact ratios for at least one minority subgroup are greater than for non-minority students. As Figure 8 shows, whereas the program results in a more than five-fold increase in the number of qualifying scores in Computer Science Principles for all students, Black students saw an increase of more than 6.7 times what would have happened without participation in the Code.org program.

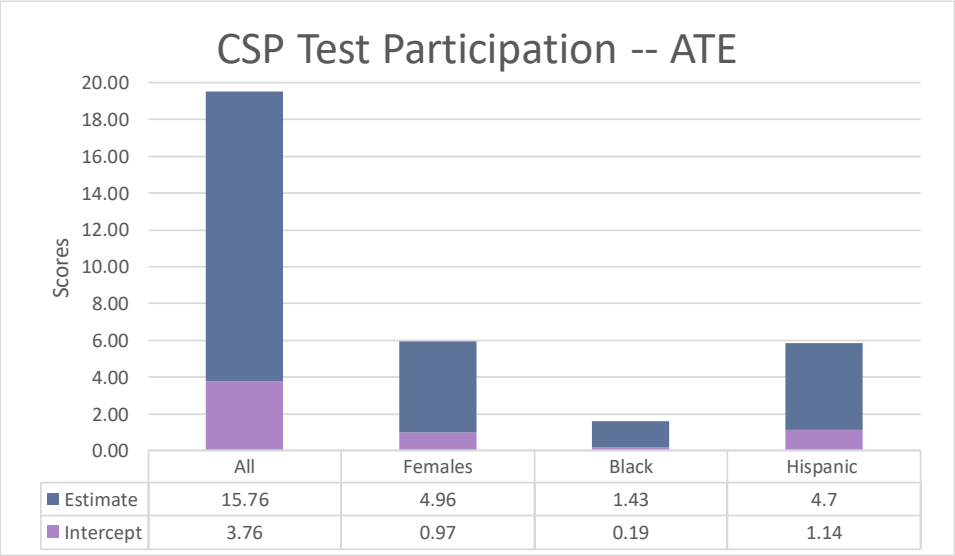


Figure 9. Average Treatment Effect on Computer Science Principles AP Test Participation

These average treatment on the treated (ATT) estimates show that program participation substantially increased the number of Advanced Placement Computer Science Principles tests taken and qualifying scores earned for students in the treatment schools. In addition to these estimates, we estimated the average treatment effect (ATE), which is the expected average effect of the program if it had been presented to the control schools as well. The results of these analyses regarding test participation are presented in Table 3. Consistent with program expectations, program implementation in the full sample would significantly improve Computer Science Principles participation for all students and all student subgroups, but would not impact test participation in Computer Science A for any group. On average, program implementation in all schools in the sample would have resulted in an additional 15.76 Computer Science Principles tests; $t(332) = 7.42$, $p < .001$, an additional 4.96 tests among female students; $t(332) = 6.70$, $p < .001$, an additional 1.43 tests for Black students; $t(332) = 4.03$, $p < .001$, and an additional 4.7 tests for Hispanic students; $t(332) = 5.03$, $p < .001$ (see Figure 9).

Table 3. ATE Estimates for Test Participation by Course

	Estimate	*t* value	*p* value <	Cohen's *d*
All Computer Science				
All Students	17.07	6.57	0.001	0.719
Female Students	5.19	6.00	0.001	0.657
Black Students	1.47	3.85	0.001	0.421
Hispanic Students	4.71	4.33	0.001	0.474
Computer Science Principles				
All Students	15.76	7.42	0.001	0.812
Female Students	4.96	6.70	0.001	0.733
Black Students	1.43	4.03	0.001	0.441
Hispanic Students	4.70	5.03	0.001	0.550
Computer Science A				
All Students	1.31	1.09	**0.275 (NS)**	0.119
Female Students	0.22	0.69	**0.494 (NS)**	0.076
Black Students	0.04	0.59	**0.553 (NS)**	0.065
Hispanic Students	0.01	0.05	**0.961 (NS)**	0.005

As was seen with the ATT estimates, the average treatment effect estimates produced a much greater impact ratio for Black student Computer Science Principles test participation than for the overall collection of students or for Female or Hispanic students. Figure 10 shows that for the full student population, the treatment increased Computer Science Principles test participation more than 500% for all students and for Hispanic students in particular, but the increase for Female students exceeded 600% and for Black students test participation increased more than 800% greater than would be observed without program participation.

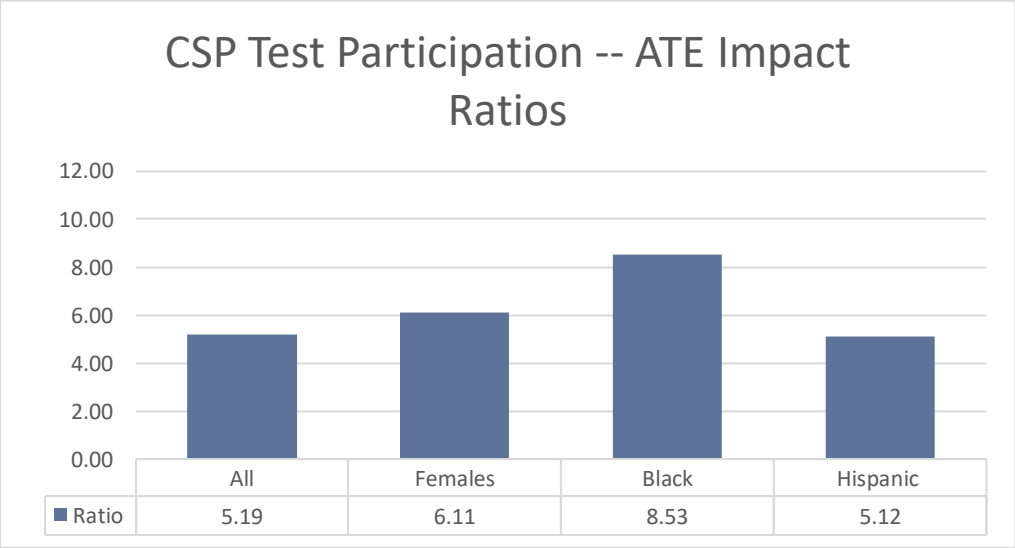


Figure 10. Impact Ratios for Student Subgroups on Computer Science Principles Test Participation.

A comparable pattern of findings was observed for Computer Science Principles qualifying scores using the average treatment effect estimates as was found with test participation using the same ATE estimand (see Table 4). Program participation would increase the average number of qualifying scores by more than 10 per school in the overall sample; $t(332) = 6.05$, $p < .001$, by an average of 2.69 for female students; $t(332) = 5.46$, $p < .001$, by an average of .39 for Black students; $t(332) = 3.83$, $p < .001$, and by an average of more than 2 qualifying scores for Hispanic students; $t(332) = 4.52$, $p < .001$ (see Figure 11). Each of these projected improvements are highly statistically significant. As with the ATT estimates, no significant improvement in Computer Science A qualifying scores is anticipated by program participation.

The impact ratios using the ATE approach, while still substantial, are lower than for the ATT estimation procedure (Figure 12). For all students, the number of qualifying scores is projected to be 4.91 times larger with the ATE approach as compared with 5.32 times larger with the ATT approach. Likewise, the ratio for females is 5.20 for ATE versus 5.32 for ATT. For minority students, the ratios are considerably lower with the average treatment effect approach compared to the average treatment on the treated approach (5.88 vs. 6.71 for Black students; 4.35 vs. 5.17 for Hispanic students). Notwithstanding these discrepancies in estimation procedures, the program effects on the number of Computer Science Principles qualifying scores remain large and significant.

Table 4. ATE Estimates for Qualifying Scores earned by Course

	Estimate	*t* value	*p* value <	Cohen's *d*
All Computer Science				
All Students	11.14	5.57	0.001	0.610
Female Students	2.97	5.18	0.001	0.567
Black Students	0.41	3.68	0.001	0.403
Hispanic Students	2.05	4.22	0.001	0.462
Computer Science Principles				
All Students	10.08	6.05	0.001	0.662
Female Students	2.69	5.46	0.001	0.598
Black Students	0.39	3.83	0.001	0.419
Hispanic Students	2.08	4.52	0.001	0.495
Computer Science A				
All Students	1.05	1.15	**0.251 (NS)**	0.126
Female Students	0.28	1.19	**0.237 (NS)**	0.130
Black Students	0.02	0.76	**0.448 (NS)**	0.083
Hispanic Students	-0.03	-0.27	**0.785 (NS)**	-0.030

In sum, the results of this study indicate substantial and significant increases in both AP test taking and qualifying score earning for all students following the implementation of the Code.org professional development program. In addition, significant program effects for Computer Science Principles AP test taking and qualifying score earning were found for female students and minority students when analyzed separately. Average effect sizes (Cohen's *d*) for treatment effects over both average treatment on treated (ATT) and average treatment effects for all students (ATE), all subgroups of students, and both outcomes, and all disciplines was *d*=.62, showing a substantial positive causal impact. The effects are stronger when looking only at the average treatment on the treated (ATT) effects, where the average effect size for first year effects was *d*=.64 across all subsamples and outcomes analyzed. The mean effect size for all analyses with the ATE approach was slightly smaller at *d*=.59, which still indicates a moderate effect size.

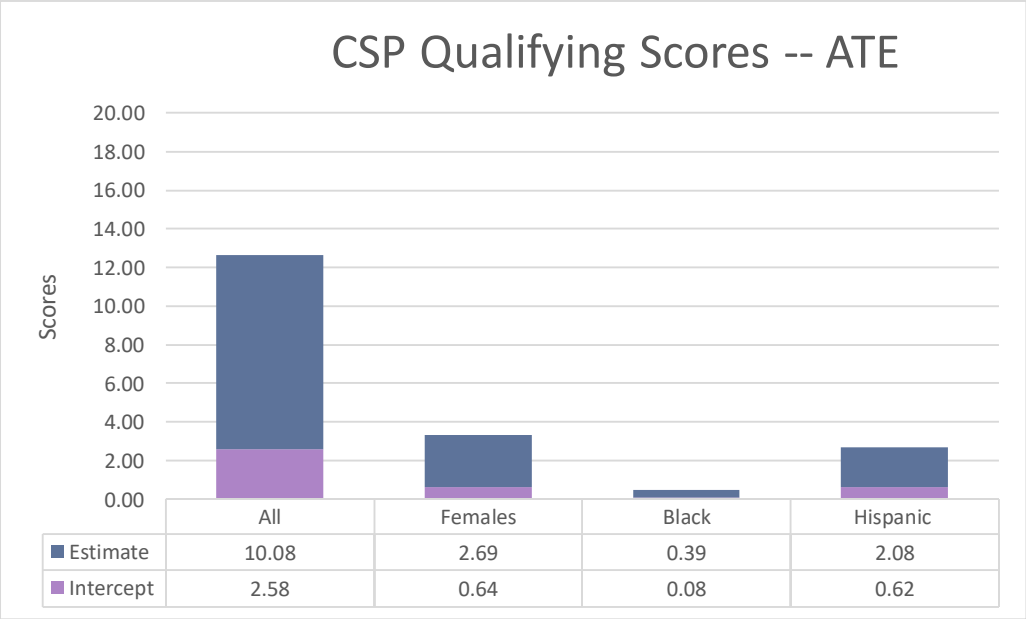


Figure 11. Average Treatment Effect of program on Computer Science Principles qualifying scores earned

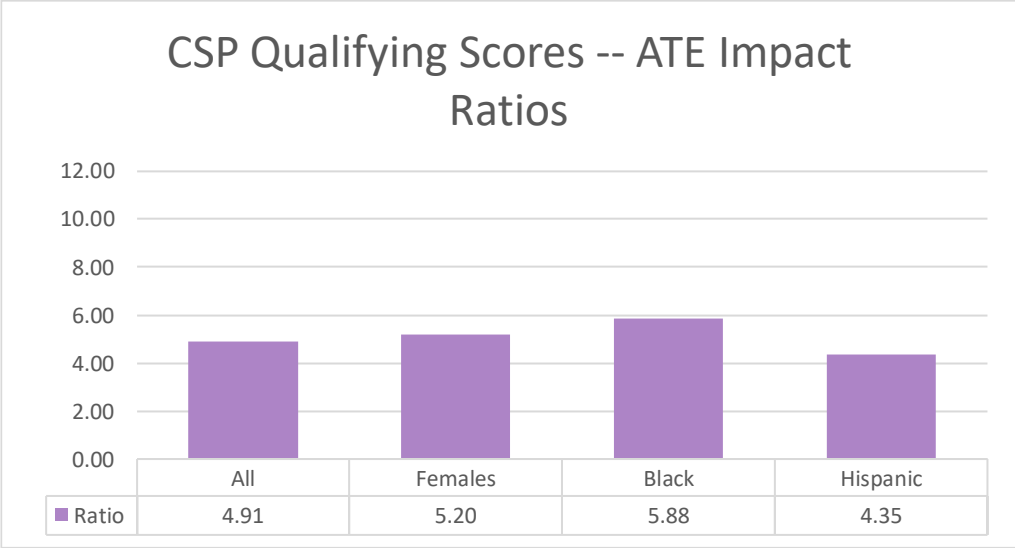


Figure 12. Impact Ratios for Student Subgroups on Computer Science Principles Qualifying Scores

4. Discussion

This study provides evidence that the Code.org teacher preparation program increases the number of AP tests taken and the number of AP qualifying scores earned by the students of the participating teachers. This is consistent with prior research that has shown that teacher professional development can, in certain contexts, positively impact student outcomes generally (Yoon, K. S., Duncan, T., Lee, S. W.-Y., Scarloss, B., & Shapley, K., 2007) and in computer science specifically (Mouza, C., Marzocchi, A, Pan, Y., & Pollock, L., 2016). In and of itself, these results are important, but these increases may lead to additional advantages for these students. Research shows that students who take AP courses have a

greater likelihood of attending college (Mattern, Marini, & Shaw, 2013). Mattern, et. al state, "… the odds of enrolling in a four-year institution increased by 171% for students who took one AP Exam compared with students who took no AP exams. The increase in odds was even higher for students who took more than one AP exam" (Mattern, Marini, & Shaw, 2013, p. 5). Students participating in AP classes also earn better grades in college (Shaw, Marini, & Mattern, 2013), and have a greater likelihood of persisting in and graduating from college (Dougherty, Mellor, & Jian, 2006; Hargrove, Godin, & Dodd, 2008). In addition, students who earn qualifying scores on AP tests outperform matched Non-AP students on many college outcome measures (Murphy & Dodd, 2009). Future research should explore these longer term potential impacts of this training program.

This work is significant for many reasons. First, it demonstrates the use of propensity score potential outcomes modeling to observational data to yield meaningful and significant causal estimates of a popular professional development program's effectiveness in a context where randomized assignment to treatment condition is either infeasible or impractical. Secondly, this study provides evidence that Code.org's Professional Development Program for CS Principles is having significant and important impacts on preparing more students to succeed in Computer Science careers and improving the future of Computer Science education in this country. More students, notably female and minority students, are engaging in, and succeeding in, Computer Science Principles as a result of implementing this program in schools across the country. From an impact ratio perspective, the program is having a greater impact for these groups of students.

Acknowledgments

Source: Derived from data provided by the College Board. Copyright © 2017 The College Board. www.collegeboard.org.

References

Austin, P. C. (2008). A critical appraisal of propensity-score matching in the medical literature between 1996 and 2003. Statistics in Medicine, 27, 2037-2049.

Beede, D. N., Julian, T. A., Langdon, D., McKittrick, G., Khan, B., and Doms, M. E., Women in STEM: A Gender Gap to Innovation (August 1, 2011). Economics and Statistics Administration Issue Brief No. 04-11. Available at SSRN: https://ssrn.com/abstract=1964782 or http://dx.doi.org/10.2139/ssrn.1964782

Dawid, A. P. (2000). Causal inference without counterfactuals. Journal of the American Statistical Association, 95(450), 407-424.

Dougherty, C., Mellor, L. & Jian, S. (2006). The relationship between Advanced Placement and college graduation. (National Center for Educational Accountability: 2005 AP Study Series, Report 1). Austin, TX: National Center for Educational Accountability.

Glass, T.A., Goodman, S.N., Hernan, M.A., & Samet, J.M. (2013). Causal inference in public health. Annual Review of Public Health, 34, 61-75.

Goode, J. (2007). If You Build Teachers, Will Students Come? The Role of Teachers in Broadening Computer Science Learning for Urban Youth. Journal of Educational Computing Research, 36(1), 65-88.

Hargrove, L., Godin, D., Dodd, B. (2008). College outcomes comparisons by AP and non-AP high school experiences (College Board Research Report 2008-3). New York: The College Board.

Holland, P. (1986). Statistics and causal inference. Journal of the American Statistical Association, 81(396), 945-960.

Imai, K., & Ratkovic, M. (2014). Covariate balancing propensity score. Journal of the Royal Statistical Society: Series B. 53, 597-610.

Keele, L. (2015). The statistics of causal inference: A view from political methodology. Political Analysis, 23, 313-335.

Louckes-Horsely, S., Stiles, K.E., Mundry, S., Love, N., Hewson, P.W. (2010). Designing Professional Development for Teachers of Science and Mathematics (3rd edition). 69-70.

Mattern, K.D., Marini, J.P., & Shaw, E.J. (2013). The relationship between AP Exam performance and college outcomes. (College Board Research Report 2009-4) New York: The College Board.

Mattern, K.D., Shaw, E.J., & Xiong, X. (2009). Are AP students more likely to graduate from college on time? (College Board Research Report 2013-5) New York: The College Board.

McCaffrey, D. F., Ridgeway, G., & Morral, A. R. (2004). Propensity score estimation with boosted regression for evaluating causal effects in observational studies. Psychological Methods, 9, 403-425.

Morgan, R., & Klaric, J. (2007). AP students in college: An analysis of five-year academic careers (College Board Research Report No. 2007-04). New York: The College Board.

Mouza, C., Marzocchi, A, Pan, Y., & Pollock, L. (2016). Development, Implementation, and Outcomes of an Equitable Computer Science After-School Program: Findings from Middle-School Students. Journal of Research on Technology in Education. 48. 1-21. 10.1080/15391523.2016.1146561.

Murphy, D., & Dodd, B. (2009). A comparison of college performance of matched AP and non-AP student groups. (College Board Research Report No. 2009-6). New York: The College Board.

Pearl, J. (2009). Causality: models, reasoning, and inference. 2nd Edition. New York: Cambridge

University Press.

Ridgeway, G., McCaffrey, D., Morral, A., Burgette, L, & Griffin, B. (2015). "twang: Toolkit for weighting and analysis of nonequivalent groups." Available at http://cran.r-project.org/web/packages/twang/twang.pdf.

Rosenbaum, P. R. (2002). Observational Studies, 2nd ed. Springer, New York.

Rosenbaum, P.R., & Rubin, D. B. (1985). Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. The American Statistician, 39, 33-38.

Rubin, D. B. (2005). Causal inference using potential outcomes: Design, modeling, decisions. Journal of the American Statistical Association, 100(469), 322-331.

Rubin, D. B. (2007). The design versus the analysis of observational studies for causal effects: Parallels with the design of randomized trials. Statistical Medicine, 26, 20-36.

Rubin, D. B., & Thomas, N. (1996). Matching using estimated propensity scores, relating theory to practice. Biometrics, 52, 249-264.

Sax, L. J., Lehman, K. J., Jacobs, J. A., Kanny, M. A., Lim, G., Monje-Paulson, L., & Zimmerman, H. B. (2017). Anatomy of an Enduring Gender Gap: The Evolution of Women's Participation in Computer Science, The Journal of Higher Education, 88:2, 258-293, DOI: 10.1080/00221546.2016.1257306

Shaw, E. J., Marini, J. P., & Mattern, K.D. (2013). Exploring the utility of Advanced Placement participation and performance in college admission decisions. Educational and Psychological Measurement, 73, 229-253.

Stuart, E. A. (2010). Matching methods for causal inference: a review and a look forward. Statistical Science, 25(1), 1-21.

West, S. G., & Thoemmes, F. (2010). Campbell's and Rubin's perspectives on causal inference. Psychological Methods, 15(1), 18-37.

Yoon, K. S., Duncan, T., Lee, S. W.-Y., Scarloss, B., & Shapley, K. (2007). Reviewing the evidence on how teacher professional development affects student achievement (Issues & Answers Report, REL 2007–No. 033). Washington, DC: U.S. Department of Education, Institute of Education Sciences, National Center for Education Evaluation and Regional Assistance, Regional Educational Laboratory Southwest. Retrieved from http://ies.ed.gov/ncee/edlabs

"Cowboy" and "Cowgirl" Programming: The Effects of Precollege Programming Experiences on Success in College Computer Science

Chen Chen[1*]

Stuart Jeckel[2*]

Gerhard Sonnert[1]

Philip M. Sadler[1]

[1]Harvard-Smithsonian Center for Astrophysics

[2]Harvard Graduate School of Education

Note: [*]Chen Chen and Stuart Jeckel equally contributed to this article and are considered co-first authors.

DOI: 10.21585/ijcses.v2i4.34

Abstract

This study examines the relationship between students' pre-college experience with computers and their later success in introductory computer science classes in college. Data were drawn from a nationally representative sample of 10,197 students enrolled in computer science at 118 colleges and universities in the United States. We found that students taking introductory college computer science classes who had programmed on their own prior to college had a more positive attitude toward computer science, lower odds of dropping out, and earned higher grades, compared with students who had learned to program in a pre-college class, but had never programmed on own, or those who had never learned programming before college. Moreover, nearly half of the effect on final grades was mediated by a positive attitude toward computing.

Keywords: self-directed, performance, experience, programming, computer science

1. Introduction

> *"…it is often the case that students come into the institution with a self-taught 'cowboy' style of programming that in no way resembles the good programming practices which we are trying to convey. In teaching, we have to make sure that those students change their habits over time. Changing one's habits, however, is harder than learning new ones."*
>
> —Kölling, 1999, p. 4

Preparing students for careers that require college degrees in science, technology, engineering, and mathematics (STEM) is a dominant concern of K-12 education in the United States (Handelsman & Smith, 2016). Computing lies at the epicenter of this concern (Smith, 2016), as labor force experts have issued dire warnings that the overall size of the American-trained future workforce in the computer science and IT fields is too small to maintain the nation's status as a leader in this area (National Research Council, 2012).

Students interested in majoring in computer science (CS) or other STEM fields want to do well in their introductory college CS course, because inferior performance may lead them to abandon career plans and change majors. However, learning to program is difficult—it requires thinking in a completely new way (Robins et al., 2003) known as computational thinking (Grover & Pea, 2013). It also requires that students be comfortable with learning by trial and error, (Anderson & Gegg-Harrison, 2013; Cooper, Dann, & Pausch, 2003), learning from open communities, and creating and participating in socially engaged practice ("the social turn"; Kafai & Burke, 2013). It has been found that it takes a novice programmer about 10 years to become an expert (Winslow 1996). Researchers have noticed a bimodal distribution in course grades (Lilja, 2001; Lister & Leaney, 2003; Carey, 2010), with failure rates commonly ranging between 30% to 60% (Bennedsen & Caspersen, 2007; Dehnadi & Bornat, 2006; Robins, 2010, Watson & Li, 2014).

A large body of research over the past 40 years has examined factors that predict success in introductory CS courses. Learning theory research suggests a growth mindset versus fixed mindset about learning may be a factor in general student success (Dweck, 2006). The CS education literature has identified several predictors of success in programming, namely, mathematics ability (Beaubouef, 2002; Byrne and Lyons, 2001;Konvalina, Wileman, & Stephens, 1983; Werth, 1986; Wilson & Shrock, 2001), problem-solving ability (Nowaczyk, 1984), high aptitude (determined by SAT, ACT, or researcher-administered tests; Wileman et al., 1981), perceived self-efficacy (Wiedenbeck, 2005), strong previous academic performance (Byrne & Lyons, 2001), and psychological factors, such as student comfort level (Ventura, 2005).

Of all the factors tested in the CS education literature, previous programming experience stands out as the single factor most consistently predicting student success in these courses (Wiedenbeck, 2005 [n=120]; Byrne & Lyons, 2001 [n=110]; Wilson & Shrock, 2001 [n=105]; and Hagan & Markham, 2000 [n=75]). Furthermore, "prior *self-initiated* computer experience, [mostly gained outside of school through "hacking" and unguided exploration], was highly predictive of university-level introductory CS course performance" (Kersteen, Linn, Clancy, & Hardyck, 1988, p. 328). Other researchers in the 1980s had also demonstrated that students were able to acquire a considerable amount of computational thinking skills by learning in a purely self-discovery environment without any teaching intervention (Kurland & Pea, 1985; Papert, 1980). The underlying philosophy is consistent with learning theories such as constructivism (Piaget & Inhelder, 1969), active learning (Harmin & Toth, 2006), and learning by design (Goldman, Eguchi, & Sklar, 2004), and quickly became the cornerstone of educational interventions using LEGO-Mindstorm (Kabatova & Pekarova, 2010) and robot design (Mubin et al., 2013). This philosophy, however, stands in contradiction to Kölling's (1999) assertion of the drawbacks of "cowboy programming" (in the title of this article, we expanded the original term adding "cowgirl programming" to steer clear of what some might consider gender-biased language, We have combined both, for convenience in places, with "cowhand" programming)—if we define cowhand programming as self-initiated programming practice, unguided and outside of school. In any case, just how much a self-initiated/self-discovery cowhand programming experience improves, or hampers students' long-term programming efficacy and performance is yet to be examined. Researchers who took a middle ground advocated that self-discovery-based learning should be incorporated with a guided and supportive curriculum (Fay & Mayer, 1994; Lee & Thompson, 1997; Mayer, 2004). Nevertheless, the comparative efficacy of cowhand-oriented and instruction-oriented approaches, the interplay of the two approaches, and the optimal sequence of the two approaches, remain hotly debated (Chase & Klahr, 2017; Dean & Kuhn, 2007; Furtak et al., 2012; Klahr, 2010; Roll et al., 2017; Tobias & Duffy, 2009; Weaver et al., 2017).

A clearer understanding of the specific ways of attaining programming experience that helps students succeed in later CS coursework will benefit practitioners who operate or create programs, students who must choose in what fashion to expend their time and energy, and college CS instructors who seek to understand their students' learning styles and to create the best learning opportunities for their students. This is more important today than ever because of the rising investment in K-12 CS offerings, as well as the proliferation of free, interactive, responsive, web-based programming education platforms like CodeAcademy, code.org, and Khan Academy, where one can learn programming on one's own.

While CS education at the K-12 level has expanded significantly in the United States (e.g., CS 10k, AP Computer Science: Principles), across the European Union (e.g., European Coding Initiative), in the United Kingdom (e.g., Computing at School), Australia (e.g., Digital Technologies), and Mexico (Escherle, Ramirez-Ramirez, Basawapatna, Assaf, Repenning, Maiello, Endo & Nolazco-Florez, 2016),

little is known about the degree to which such initiatives are successful beyond the immediate measurement of student enjoyment. A more rigorous approach to evaluating the effectiveness of precollege experiences is to measure their impact on performance in later introductory college coursework (Tai, Sadler, & Mintzes, 2006). In this way, many different types of in-school and out-of-school initiatives and experiences can be compared, and resources can be better allocated to those programs that are the most effective, rather than relying on anecdotal reports to make policy decisions. In the case of computer science, examining the degree to which precollege experiences predict performance in introductory college CS courses nationwide, provides a metric that is both fair and meaningful.

In this study, we ask if pre-college cowhand programming experience is associated with different college level CS attitude and performance, compared with a) students who had never had any programming experience, b) students who had learned programming only in a previous class, but never cowhand, and c) students who had both in-class and cowhand programming experience (in each possible sequence).

2. Data collection

The "Factors Influencing College Success in Information Technology" (FICSIT) study, which was conducted at Harvard University with funding from the National Science Foundation (grant number 1339200), supplied the data for this article. The FICSIT study sent out a 52-item survey to examine the pre-college computer experiences of students. The study's investigators obtained responses from a stratified random sample of 10,197 students enrolled in 118 2- and 4-year colleges and universities across the United States. The Integrated Postsecondary Education Data System of the National Center for Educational Statistics provided a listing of post-secondary institutions that was used to build a stratified random sample reflecting the proportion of students in the U.S. enrolled in 2-and 4-year colleges in three different size bins (small, medium, and large). We contacted 1080 different institutions (279 2-year and 801 4-year) that offered introductory courses in CS in the fall semester of 2015; 138 of them agreed to participate (30 2-year and 108 4-year), and 118 of them (23 2-year and 95 4-year) actually returned the survey. Thus, our final sample consisted of 118 institutions. Within the first two weeks of class, 10,197 students filled out a 52-item, paper-and-pencil, in-class survey about their prior CS experiences and current attitudes about computing, along with background questions about a wide variety of their educational experiences, as well as family background and demographic characteristics. Instructors provided students' final grade in the college introductory CS course or indicated that the student had dropped the course. We excluded respondents who had missing values in more than 50% of the questionnaire or in the key predictor variables or outcome variables. As a result, the final sample size in our analysis was 8,891.

3. Variables and modeling

This section explains the variables and analysis methods. In a nutshell, the predictor of interest was participants' programming experience, but we also controlled for a wide range of background variables. The predictor variables and controlled variables were collected retrospectively. The dependent or outcome variables were 1) the grade in computer science that they eventually received in the introductory CS course by the end of the semester (filled in by their instructors); 2) participants' attitude toward computer science at the time they answered the survey; and 3) course completion. Students in the sample were clustered in the models by their course instructor. To account for the significant autonomy instructors often have in grading, a random intercept, two-level hierarchical approach was used in all models. We first built hierarchical regression models to predict each of the dependent variables separately. We further constructed a path model to predict multiple dependent variables simultaneously.

3.1. Predictor of interest

While this study considered many student pre-college experiences with computing as possible predictors of performance, the final models examined programming experience in terms of "cowhand programming" and in-class computing experience. Our definition of cowhand programming was that a student had programmed by him/herself without the supervision of classroom teachers. It could happen prior or after taking computing classes, or not in combination with any in-class experience. The type of programming experience was determined based on participants' answers to questions about "taking a class about computer programming," "have programmed on my own," and the order of the two. There were five groups: 1) *control* (n=6824), students who had never taken any class about computer programming, nor had practiced programming outside of class; 2) *cowboy-or-cowgirl-only* (479), students who had programmed on their own, but had never taken any class about computer programming; 3) cowhand→class (n=250), students who had first programmed on their own prior to taking a class about computer programming; 4) class→cowhand (n=422), students who had started programming on their own during or after taking a class; 5) Class-only (n=916), students who had taken classes about computer programming, but had never programmed on their own. We did not know the particular programming languages that the participants learned because such information was not specified in the questionnaire. We discuss this limitation by the end of the article.

3.2. Control variables

Several variables were used in the model to account for significant differences in student backgrounds. Gender was coded as a dichotomous variable, zero for female and one for male. Race/ethnicity was divided into five categories coded as a set of dummy variables—Asian, Hispanic, black, and white, with white as the baseline.

Socioeconomic status (SES) entered the model through the proxy of parents' education. The parents' highest level of education was indicated by students on a scale from zero through five: zero representing "did not finish high school," one—"high school graduate," two—"some college," three—"4 years of college," and four—"graduate level education."

Mathematics preparation has often been found to be associated with CS course performance (Byrne & Lyons, 2001; Werth, 1986; Wilson & Shrock, 2001). The SAT/ACT mathematics score was used as a proxy for the students' academic preparation and aptitude. If students reported ACT scores, but no SAT scores, their ACT scores were converted to the SAT scale according to a concordance published by the College Board (1999). In our sample, the correlation between SAT/ACT mathematics score and overall SAT/ACT score was 0.25.

We also controlled for whether the student was born in the United States, whether the first language was English, whether the student went to a public high school, whether the student had access to computer at home, and whether the job of any of the parents was related to CS. In all statistical models, we have controlled for the covariates listed above.

3.3 Dependent variables

This study estimated with three distinct dependent variables. The first was a hierarchical linear regression model, for which the dependent variable was the numerical grade in the course in introductory college CS at the end the semester. Institutions in the sample used different measures in awarding grades, with 11% using single letter grades (i.e., A, B, C, D, F), 40% using "+" and "-" to augment letter grades (e.g., A+, B-), and 49% using a 100-point scale. All letter grades reported were converted to the same 0 to 100-point scale (e.g., B+=88.5, B=85, B-=81.5). For uniformity, all grades below 60 were coded as 55, representing an "F."

The second was another hierarchical linear regression model. The dependent variable was positive attitude toward CS, which was represented by the first component score based on a principal component analysis of 23 items. Overall, this is a measurement of the level of interest, efficacy, and comfort in regard to computing. The scale was developed by the authors, Inspiration for some attitude items was drawn from the Computing Research Association (CRA) Undergraduate Survey and the Scientific Attitude Inventory (Moore & Hill Foy, 1997). The scale had a very good internal consistency with a Cronbach alpha value of 0.91. We used the first component because it has an eigenvalue of 9.12 and successfully explained 38% of the variance, whereas the second component only has an eigenvalue of 1.60 explaining 6.4% of the variance. Table 1 presents the wording and loading of each item.

Table 1. Wording and PCA loading of each items in the scale measuring positive attitude towards CS

	Positive Attitude (PC1)	
Items	Loading	Contrib%
1. Computer science is interesting to me	0.84	7.75
2. I look forward to taking computer science	0.83	7.70
3. I feel I belong in computer science	0.80	7.00
4. I am confident I can do well in computer science	0.76	6.25
5. Computer science is a creative activity	0.75	6.20
6. Computer science can help in solving problems	0.74	5.96
7. I enjoy using algorithms to solve computational problems	0.73	5.79
8. Computer science allows me to develop models from abstractions	0.73	5.82
9. I feel comfortable doing computer science	0.72	5.64
10. Computer science facilitates the creation of knowledge	0.70	5.47
11. Computer science is relevant to real life	0.63	4.38
12. I do not get discouraged from setbacks in computer science	0.62	4.30
13. I wish I did not have to take computer science	-0.61	4.21
14. Computer science is boring for me	-0.59	3.93
15. I feel accepted by my peers in computer science	0.56	3.43

16. Computer science provides new ways to communicate globally	0.55	3.36
17. The internet fosters collaboration worldwide	0.49	2.64
18. I am comfortable asking classmates for help	0.43	1.96
19. Computer science people are intelligent	0.39	1.71
20. Most people can understand computer science	0.39	1.67
21. I am comfortable asking my professors/TAs/tutors for help	0.37	1.54
22. Computer science makes me nervous	-0.32	1.17
23. Most people in computer science are nerds or geeks	0.02	0.01

The third model was a hierarchical logistic regression model, with a binary dependent variable indicating whether students either completed their college CS course or dropped out of the course—5.54% of sampled students.

Lastly, we specified a path model to test the relationship among cowhand programming experience, positive attitude towards CS, and final grade. Specifically, we examined whether the effect of prior experience on the final grade was mediated by positive attitude.

4. Results

4.1 *Descriptive statistics*

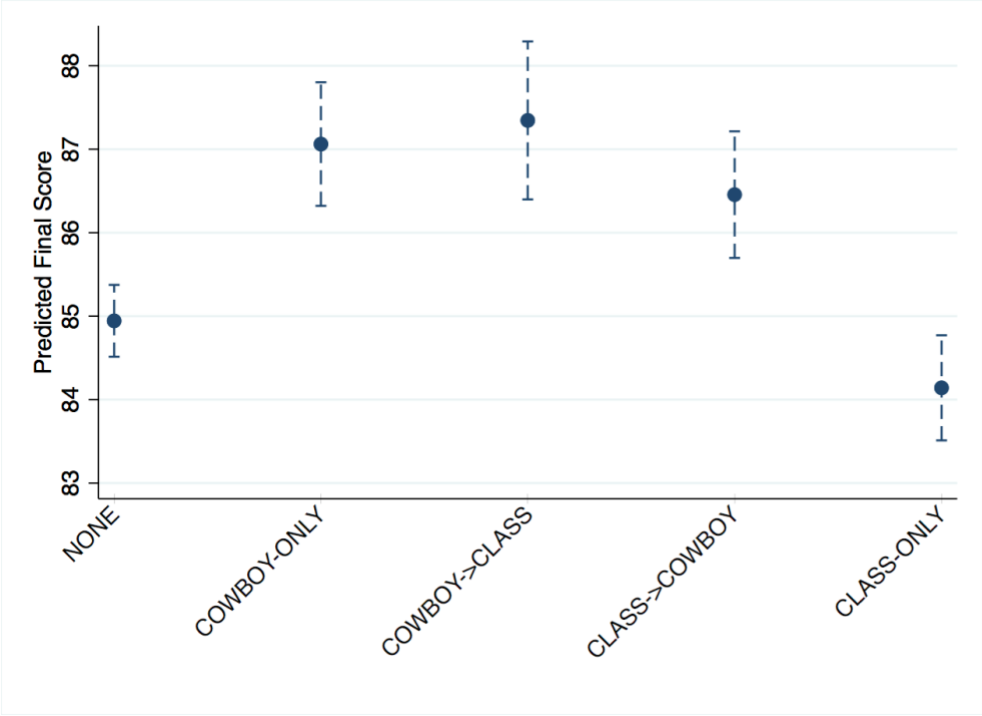
The mean grade for those who completed the course was 84.34 (sd=13.83) on a 100-point scale. The mean mathematics SAT/ACT score was 639.27 (sd=126.53), approximately 1.39 standard deviation (in standard deviation units of our sample) higher than the national average of 500 (College Board, 2014). Two-thirds of the sample had at least one parent with either 4 years of college or a post-graduate education. With just a 5.5% dropout rate, this U.S. study's findings are at odds with Bennedsen and Caspersen's (2007) and Watson and Li's (2014) global studies, which estimated a roughly 33% dropout rate for introductory programming courses. Other descriptive summaries, broken down by programming experience groups, are shown in Table 2.

Table 2. Descriptive summary by group

	None	Cowhand Only	Cowhand-> Class	Class-> Cowhand	Class Only
Definition:					
Taking class	No	No	Yes	Yes	Yes
Practicing programming	No	Yes	Prior class	In/After class	No
N	6824	479	250	422	916
Dropout Rate	5.80%	3.50%	4.40%	4.70%	5.80%
Final Grade (sd)	84.01 (13.94)	87.46 (12.33)	86.75 (11.79)	86.21 (12.37)	82.40 (14.88)
Male	70.76%	85.14%	82.11%	80.14%	68.34%
Born in US	75.40%	82.67%	81.20%	84.59%	68.62%
English is First Language	69.50%	79.33%	79.60%	76.06%	66.32%
Public High School	77.40%	73.27%	71.60%	77.25%	79.49%
Race. Asian	24.20%	24.63%	26.80%	23.93%	19.87%
Hispanic	9.60%	2.64%	4.00%	6.23%	15.28%
Black	8.40%	5.33%	4.80%	6.10%	15.48%
White	57.80%	67.40%	64.40%	63.74%	49.37%
SAT Math Score ...(sd)	634.54 (125.71)	683.11 (104.64)	693.12 (109.07)	678.72 (108.73)	561.25 (142.78)
CS Help Outside of School	6.60%	16.07%	20.40%	15.87%	8.70%
Parents' Jobs Relate to CS	2.10%	32.35%	46%	33.88%	23.01%
Access Computer at Home	9.10%	97.28%	92.33%	96.19%	83.55%

4.2. Hierarchical models

Our first hierarchical regression model (M1) showed that all three groups that had cowhand programming experience did not statistically differ from each other in the final grade. All three of them outperformed the control group. They also outperformed the Class-Only group based on our post-hoc test (for Cowhand-Only versus Class-Only, χ^2=17.70, p<0.001; for Cowhand→Class versus Class-Only, χ^2=13.11, p<0.001; for Class→Cowhand, χ^2=9.59, p=0.002). Figure 1 shows the predicted score by programming experience groups, controlling for other covariates at the mean. This result also translates to a significant main effect of cowhand programming (β=2.13, se=0.47, p<0001) if we aggregate the five groups into two groups (Cowhand versus non-Cowhand). On average, students who had programmed on their own scored 2.13 points higher than students who had not, an effect size of 0.15.



Similarly, in M2, the three groups that had cowhand programming experience did not differ from each other on the scale of positive attitude towards CS, whereas they scored higher than the control and the Class-Only group.

In the hierarchical logistic regression predicting student dropout (M3), we found that the Cowhand-Only group was the only group that to yield significantly lower odds of dropping out than the control group. The parameters for all three models are shown in Table 3.

Table 3. Hierarchical regression model predicting the final grade and the odds of dropout

	M1.Final Grade			M2.Positive Attitude			M3.Dropout		
	□	*s.e.*	*p-value*	*b*	*s.e.*	*p-value*	*Odd Ratio*	*s.e.*	*p-value*
Cowhand-Only	2.23	0.68	**0.001**	1.29	0.14	**0.0001**	0.52	0.16	**0.04**
Cowhand->Class	2.43	0.92	**0.01**	1.31	0.19	**0.0001**	0.64	0.25	0.25
Class->Cowhand	1.59	0.7	**0.03**	1.19	0.15	**0.0001**	0.61	0.17	0.08
Class-Only	-0.8	0.77	0.227	-0.16	0.12	0.17	0.89	0.3	0.75
Male	-1.04	0.36	**0.004**	0.94	0.07	**0.0001**	0.94	0.13	0.67
US born	-0.75	0.49	0.13	-0.37	0.11	**0.001**	1.65	0.32	**0.01**
Language	0.59	0.47	0.21	-0.02	0.1	0.8	0.79	0.14	0.19
Pub-School	0.27	0.38	0.48	0.01	0.08	0.88	1	0.15	0.99
Asian	-0.83	0.44	0.06	-0.16	0.09	0.1	0.85	0.14	0.37
Hisp	-1.09	0.55	**0.04**	0.27	0.11	0.02	0.84	0.17	0.42
Black	-4.51	0.63	**0.0001**	-0.05	0.13	0.71	0.84	0.22	0.5
Ed Father	0.41	0.17	**0.01**	-0.09	0.04	0.02	0.93	0.06	0.26
Ed Mother	-0.08	0.18	0.65	-0.03	0.04	0.05	0.96	0.07	0.53
SAT score	0.02	0.001	**0.0001**	0.003	0.0003	**0.0001**	0.99	0.0004	**0.0001**
Help	0.23	0.58	0.69	0.28	0.13	**0.03**	0.6	0.16	0.06
Parent job	-0.59	0.38	0.12	-0.13	0.08	0.12	0.93	0.14	0.64
Access to computer	0.14	0.75	0.85	0.08	0.16	0.61	0.52	0.13	**0.01**
Intercept	68.11	1.44	**0.0001**	-3.79	0.32	**0.0001**	0.21	0.12	0.006
□□^{2}(Level-2)	22.84	0.38		0.89	0.07		2.85	0.21	
□□^{2}(Level-1)	147.38	0.11		2.53	0.02				

Lastly, we used the path analysis to examine a possible mediation of the effect from cowhand programming experience to final grade via positive attitude. Because M1 and M2 have both shown that the three groups with cowhand programming experience did not differ from each other, and that Class Only and the control group did not differ from each other, we decided to simplify our model to merge the five groups into two groups (Cowhand versus Non-Cowhand) to model the cowhand programming effect in general. We ran a path analysis that simultaneously tested for the effect of cowhand programming on positive attitude, the effect of cowhand programming on final grade, and the effect of

positive attitude on final grade, while controlling for all covariates and accounting for the two-level clustering.

As shown in Figure 2 (for presentation purpose this figure does not show the covariates and clustering; they were nevertheless accounted for in the model), when modeling the effects simultaneously, cowhand programming had significant positive effect on positive attitude (std.est=1.735, robust.se=0.113, p<0.001) and final grade (std.est=0.086, robust.se=0.035, p=0.015), and the positive attitude also had a positive effect on final grade (std.est=0.041, robust.se=0.005, p<0.001). There was a significant mediation effect from cowhand programming to final grade via positive attitude (std.est=0.070, robust.se=0.009, p<0.001), whereas the total effect from cowhand programming to final grade was 0.157. In other words, 44.58% of the total effect from cowhand programming to final grade was mediated by positive attitude.

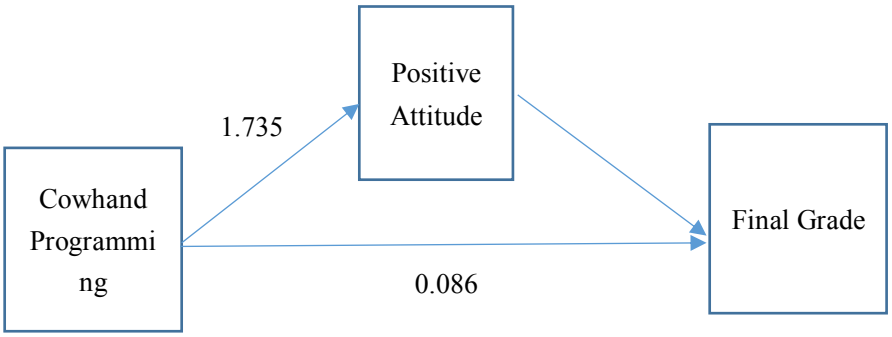


Figure 2. Path diagram of the effect from Cowhand programming to final grade partially mediated by positive attitude towards computer science. Other covariates and two-level clustering were accounted for in the estimation but not shown in the diagram.

5. Discussion and conclusion

In *Paradoxes of Education in a Republic,* philosopher Eva Brann (1979) wrote, in 1979, "Those who lack a good grounding must learn laboriously and lengthily in schools what the well-educated learn quickly by themselves." She mentioned computer programming explicitly as an example of a field in which this is the case. We have found little to refute this nearly half-century-old claim. At the pre-college level, self-directed learning still may be the best way to learn to program.

Why might it be the case that cowhand programmers outperformed the other students on all fronts? It is reasonable to suspect that cowhand programmers have a greater perceived self-efficacy and comfort level with programming. Moreover, by programming alone, cowhand programmers have expressed

greater interest in, and commitment to, the subject, which may help them push through the difficult portions of their college CS course when many of their peers do not. This interpretation is supported by the mediation effect of positive attitude, which measures the level of interest, efficacy, and comfort. Previous research has also shown such a mentality to be associated with success in CS (Ramalingam, LaBelle, & Wiedenbeck, 2004; Ventura, 2005; Wiedenbeck, 2005). In addition, because college requires a more independent working style than does high school, it is possible that cowhand programmers are more suited to college success in general. College work may require more independence from classroom instructors, but it is not independent of the outside world. This is especially true for CS, which enables, and heavily relies upon, the open source online community. The development of search engines, as well as the ease with which learners can access the help of experts through message boards and websites like Stack Exchange, means that one is hardly learning alone anymore when one is programming "on one's own." Students who have had the experience of programming independent of a classroom instructor may be more familiar and comfortable using a variety of tools to "geek out and mess around," an attribute that is crucial for new generation programmers and innovators (Ito et al., 2009; Liggett, 2014; McKenna & Bergie, 2016).

Compared with the cowhand-only group, our study did not find any distinctive advantage for cowhand programming in combination with classroom learning in whichever sequence. This might be due to the lack of information about the pedagogy adopted in the prior computing classes. Nevertheless, our finding strongly suggests that to learn in class without any independent hands-on experience is a rather ineffective strategy.

The major limitation of this study was that we could not narrow down the language, environment and pedagogy in which the cowhand and in-class groups were introduced to computer programming. Because of the varying complexity of underlying data structures and algorithms, the specific programming languages the students are exposed to could make a significant difference in their attitude toward CS subjects in general. For example, students who had experience with Alice, which uses stories and games to teach logic and primitive data structures, would have very different understanding and expectations of CS subjects from those who had exposure to a modern high-level programming language like C++ or Java. It is not uncommon for CS majors to drop out or change majors simply because they are frustrated by the very first programming language they encounter. As a matter of fact, many schools have moved from the strongly typed (stricter typing rules) languages, such as C++ or Java, to scripting languages like Python in their CS1 curriculum, to improve retention. Future studies should carefully observe the context in which student learn their first programming language by themselves, with family, online or in class.

Cowhand programmers' stronger college CS performance suggests that, whatever bad habits they may have developed by lacking a teacher to guide them, as Kölling (1999) suggests, those are more than

compensated for by what they learn on their own. However, the claim that cowhand programmers have a tendency pick up negative habits may be far less tenable today than it may have been in 1999, at the time of his writing. Recent advances in graphic based languages, such as Scratch, as introductory languages, have greatly reduced the demand for syntactic memory. Accompanied with well-designed tutorials, challenges and forums online, these introductory languages have empowered beginners to self-explore and self-teach at a young age. Many scholars once expressed concerns, similar to Kölling's, that non-professional languages may introduce bad syntactic habits, such as missing semicolons or cluttered global variables (Meerbaun, Armoni & Ben-Ari, 2011; Powers, Ecott & Hirshfield, 2007; Techapalokul, 2017). An increasing number of studies, however, has shown that such an introductory language environment both promotes interest in programming (Bers, 2010; Brennan et al., 2011; Fessakis, Gouli & Mavroudi, 2013; Sáez-López, Román-González & Vázquez-Cano, 2016; Weintrop & Wilensky, 2017; Wilson & Moffat, 2012) and has long-term performance benefits in programming (Chen, Haduong, Brennan, Sonnert & Sadler, 2018). Based on our findings, and the recent development of beginner friendly programming environments, we recommend that 1) introduction to computer programming should not wait until taking a formal course; students should be encouraged to explore by themselves. 2) Getting one's hands "dirty" is more important than keeping the code "clean" and well formatted. Only following instruction in a pre-college classroom without more independent hand-on experience may place students at a disadvantage compared with those who have hand-on experience, but are not class-trained. 3) To program in cowhand style does not only lead to acquisition of knowledge and skills, but more importantly appears to cultivate a positive attitude towards computer science, which translates to better performance in programming in the long term.

The demand for professional programmers, the interest in majoring in CS in college, and the general push by parents and policy makers for K-12 offerings in computer science have all increased (Yadav, 2016; Strickland, 2014; Taylor & Miller, 2015). In the meantime, many secondary CS teachers are currently undertrained, with limited teacher preparation for teaching pre-college computer science in the United States (Yadav, 2016). Despite the scarcity of resources, students have pursued self-teaching through innovative channels such as Code Academy, Dev Bootcamp, and open resources (McKenna & Bergie, 2016). We believe such a self-initiated approach should be supported wherever and whenever possible by pointing students towards the best available learning resources and encouraging them to work on their own with the help of responsive tools, online community support, and well-designed instruction.

Acknowledgments

This work was supported by the National Science Foundation (grant number 1339200). Any opinions, findings, and conclusions in this article are the authors' and do not necessarily reflect the views of the National Science Foundation. Without the extraordinary contributions of many people, the FICSIT

project would not have been possible. We thank the members of the FICSIT team: Wendy Berland, Hal Coyle, Zahra Hazari, Annette Trenga, and Bruce Ward. We would also like to thank several STEM educators and researchers who provided advice or counsel: Lecia Barker (Chair of Advisory Committee), Randy Battat, Joanne Cohoon, Maria Litvin, Clayton Lewis, Irene Porro, Kelly Powers, Lucy Sanders, Susanne Steiger–Escoba, Jane Stout, Charles Alcock, and Janice Cuny. Last but not least, we are grateful to the many college computer science professors and their students who gave up a portion of a class to provide data.

References

Anderson, N., & Gegg-Harrison, T. (2013). Learning computer science in the comfort zone of proximal development. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 495-500). ACM.

Ito, M., Baumer, S., Bittanti, M., Cody, R., Stephenson, B.H., Horst, H.A., Lange, P.G., Mahendran, D., Martínez, K.Z., Pascoe, C.J. and Perkel, D. (2009). *Hanging out, messing around, and geeking out: Kids living and learning with new media*. MIT press.

Beaubouef, T. (2002). Why computer science students need math. *ACM SIGCSE Bulletin, 34*(4), 57-59.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 32-36.

Beyer, S., Rynes, K., Perrault, J., Hay, K., & Haller, S. (2003, January). Gender differences in computer science students. *ACM SIGCSE Bulletin, 35*(1), 49-53.

Brann, E. T. (1979). *Paradoxes of education in a republic*. University of Chicago Press.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin, 33*(3), 49-52.

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2018). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education*, 1-26.

College Board. (2014). 2014 College-bound seniors: Total group profile report. Retrieved December 13, 2016, from https://secure-media.collegeboard.org/digitalServices/pdf/sat/TotalGroup-2014.pdf

College Board Office of Research and Development. (1999). Concordance between SAT I and ACT scores for individual students (Report RN-07, June 1999). College Board.

Dehnadi, S., & Bornat, R. (2006). The camel has two humps. Paper presented at the *LittlePPIG 2006 workshop*, Coventry, UK. Retrieved from http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

Dweck, C. S. (2006). *Mindset: The new psychology of success*. Random House.

Escherle, N. A., Ramirez-Ramirez, S. I., Basawapatna, A. R., Assaf, D., Repenning, A., Maiello, C., Endo, Y. C., & Nolazco-Flores, J. A. (2016, February). Piloting computer science education week in Mexico. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 431-436). ACM.

Fay, A. L., & Mayer, R. E. (1994). Benefits of teaching design skills before teaching Logo computer programming: Evidence for syntax-independent learning. *Journal of Educational Computing Research*, *11*(3), 187-210.

Goldman, R., Eguchi, A., & Sklar, E. (2004). Using educational robotics to engage inner-city students with technology. In *Proceedings of the 6th international conference on Learning sciences* (pp. 214-221). International Society of the Learning Sciences.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin, 32*(3), 25–28.

Handelsman, J., & Smith, M. (2016, February 11). STEM for all. Retrieved from https://www.whitehouse.gov/blog/2016/02/11/stem-all

Harmin, M., & Toth, M. (2006). *Inspiring active learning: A complete handbook for today's teachers*. ASCD.

Honour Werth, L. (1986). Predicting student performance in a beginning computer science class. In *Proceedings of the 17th ACM Technical Symposium on Computer Science Education - SIGCSE '86* (pp. 138–143). ACM.

Kabátová, M., & Pekárová, J. (2010). Lessons learnt with LEGO Mindstorms: From beginner to teaching robotics. *AT&P Journal PLUS 2*, 51-56.

Kersteen, Z. A., Linn, M. C., Clancy, M., & Hardyck, C. (1988). Previous experience and the learning of computer programming: The computer helps those who help themselves. *Journal of Educational Computing Research*, *4*(3), 321-333.

Kölling, M. (1999). The problem of teaching object-oriented programming. *Journal of Object Oriented Programming, 11*(8), 8-15.

Konvalina, S., Wileman, S. A., & Stephens, L. J. (1983) Math proficiency: A key to success for computer science students. *Communications of the ACM*, *26*(5), 377-382.

Lee, M. O. C., & Thompson, A. (1997). Guided instruction in LOGO programming and the development of cognitive monitoring strategies among college students. *Journal of Educational Computing*

Research, *16*(2), 125-144.

Liggett, J. B. (2014). *Geek as a constructed identity and a crucial component of STEM persistence*. Master of Science thesis, University of North Texas.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning?. *American Psychologist*, *59*(1), 14-19.

McKenna, B. W., & Bergie, L. (2016). Creating the next generation of innovators. *Publications & Research* Paper 6. Retrieved from http://digitalcommons.imsa.edu/stratinnov_pr/6.

Moore, R. W., & Foy, R. L. H. (1997). The scientific attitude inventory: A revision (SAI II). *Journal of Research in Science Teaching*, *34*(4), 327-336.

Mubin, O., Stevens, C. J., Shahid, S., Al Mahmud, A., & Dong, J. J. (2013). A review of the applicability of robots in education. *Journal of Technology in Education and Learning*, *1*(209-0015), 1-7.

National Research Council (2012). *Report of a Workshop on Science, Technology, Engineering, and Mathematics (STEM) Workforce Needs for the U.S. Department of Defense and the U.S. Defense Industrial Base.* The National Academies Press.

Nowaczyk, R. H. (1984). The relationship of problem-solving ability and course performance among novice programmers. *International Journal of Man-Machine Studies, 21*(2), 149–160.

Piaget, J., & Inhelder, B. (2008). *The psychology of the child*. Basic Books.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *ACM SIGCSE Bulletin, 36*(3), 171-175.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education, 20*(1), 37-71.

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

Smith, M. (2016, January 30). Computer science for all. Retrieved December 12, 2016, from https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

Strickland, D. (2014, October 7). L.A. United announces larger focus on computer science for K-12. *Los Angeles United School District*. Retrieved from http://home.lausd.net/apps/news/article/407400

Taylor, K., & Miller, C. C. (2015, September 15). De Blasio to announce 10-year deadline to offer computer science to all students. *The New York Times*. Retrieved from http://www.nytimes.com/2015/09/16/nyregion/de-blasio-to-announce-10-year-deadline-to-o er-computer-science-to-all-students.html

Tai, R. H., Sadler, P.M., & Mintzes, J. J. (2006). Factors influencing college science success. *Journal of College Science Teaching, 35*(8), 56-60.

Ventura, P. R. (2005). Identifying predictors of success for an objects-first CS1. *Computer Science Education, 15*(3), 223–243.

Watson, C., & Li, F. (2014, June). Failure rates in Introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 39–44). ACM.

Wiedenbeck, S. (2005, October). Factors affecting the success of non-majors in learning to program. In *Proceedings of the First International Workshop on Computing Education Research* (pp. 13-24). ACM.

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *ACM SIGCSE Bulletin, 33*(1), 184–188.

Winslow, L. E. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin, 28*(3), 17- 22.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education, 26*(4), 235-254.

IJCSES

Volume 2, Issue 4

January 2019

ISSN 2513-8359