"Cowboy" and "Cowgirl" Programming and Success in College Computer Science

Chen Chen[1]
Stuart Jeckel[2]
Gerhard Sonnert[3]
Philip M Sadler[4]

[1] Harvard Smithsonian Center for Astrophysics
[2] Harvard Graduate School of Education
[3] Harvard Smithsonian Center for Astrophysics
[4] Harvard Smithsonian Center for Astrophysics

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, Maiello, & 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, $\chi 2=17.70$, $p<0.001$; for Cowhand→Class versus Class-Only, $\chi 2=13.11$, $p<0.001$; for Class→Cowhand, $\chi 2=9.59$, $p=0.002$). 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 ($\beta=2.13$, $se=0.47$, $p<0001$) 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.

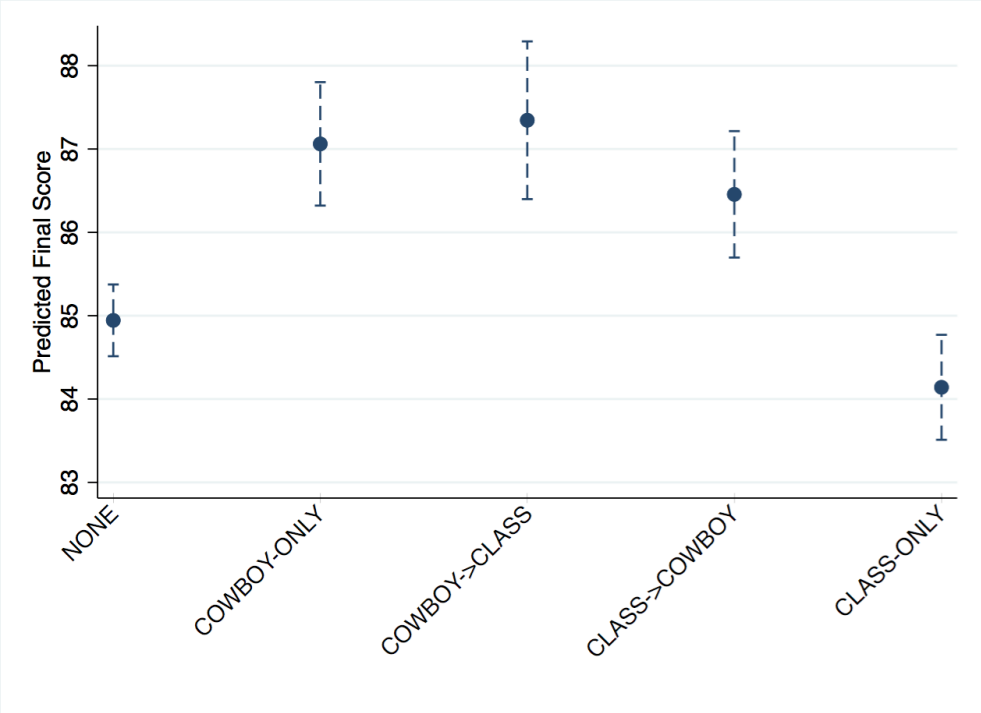

Figure 1. Predicted final grade (± 1 s.e) by different programming experience based on the hierarchical regression model.

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 odd of dropout

	M1.Final Grade			M3.Positive Attitude			M3.Dropout		
	b	*s.e.*	*p-value*	*b*	*s.e.*	*p-value*	*Odd Ratio*	*s.e.*	*p-value*
Cowboy-Only	2.23	0.68	**0.001**	1.29	0.14	**0.0001**	0.52	0.16	**0.04**
Cowboy->Class	2.43	0.92	**0.01**	1.31	0.19	**0.0001**	0.64	0.25	0.25
Class->Cowboy	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
s_e^2(Level-2)	22.84	0.38		0.89	0.07		2.85	0.21	
s_ε^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.

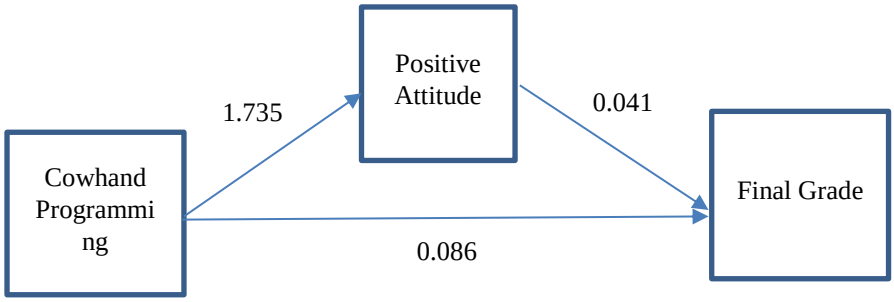

Figure 2. Path diagram of the effect from Cowhand programming to final grade partially mediated by positive attitude towards computer science. Other covariates and two-level clustering were accounted for in the estimation but not shown in the diagram.

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.

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., & 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. http://doi.org/10.1145/1272848.1272879

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. doi:10.1145/611892.611930

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*, DOI: 10.1080/08993408.2018.1547564

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 June, 2009, from http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

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

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. http://doi.org/10.1145/353519.343063

Handelsman, J., & Smith, M. (2016, February 11). STEM for all. Retrieved December 12, 2016, 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. http://doi.org/10.1145/953055.5701.

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. 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: The Official Journal of the National Association for 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. http://doi.org/10.1016/S0020-7373(84)80064-4

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. http://doi.org/10.1080/08993401003612167

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. http://doi.org/10.1076/csed.13.2.137.14200

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. http://doi.org/10.1080/08993400500224419

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. http://doi.org/10.1145/2591708.2591749

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. doi:10.1145/366413.364581

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. doi:10.1080/08993408.2016.1257418.