Computer Science Education

ISSN: 0899-3408 (Print) 1744-5175 (Online) Journal homepage: https://www.tandfonline.com/loi/ncse20

The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages

Chen Chen, Paulina Haduong, Karen Brennan, Gerhard Sonnert & Philip Sadler

To cite this article: Chen Chen, Paulina Haduong, Karen Brennan, Gerhard Sonnert & Philip Sadler (2019) The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages, Computer Science Education, 29:1, 23-48, DOI: 10.1080/08993408.2018.1547564

To link to this article: https://doi.org/10.1080/08993408.2018.1547564

Published online: 15 Nov 2018.

COMPUTER SCIENCE EDUCATION
2019, VOL. 29, NO. 1, 23–48
https://doi.org/10.1080/08993408.2018.1547564

Routledge
Taylor & Francis Group

ARTICLE

The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages

Chen Chen[a], Paulina Haduong[b], Karen Brennan[b], Gerhard Sonnert[a] and Philip Sadler[a]

[a]Science Education Department, Harvard-Smithsonian Center for Astrophysics, Cambridge, MA, USA; [b]Harvard Graduate School of Education, Cambridge, MA, USA

ABSTRACT

Background and Context: The relationship between novices' first programming language and their future achievement has drawn increasing interest owing to recent efforts to expand K–12 computing education. This article contributes to this topic by analyzing data from a retrospective study of more than 10,000 undergraduates enrolled in introductory computer science courses at 118 U.S. institutions of higher education.

Objective: We explored the relationship between students' first programming languages and both their final grades in an introductory computer science course and their attitudes about programming.

Method: Multiple matching techniques compared those whose first language was graphical (e.g., Scratch), textual (e.g., Java), or absent prior to college.

Findings: Having any prior programming experience had positive effects on both attitudes about programming and grades in introductory computer science courses. Importantly, students whose first language was graphical had higher grades than did students whose first language was textual, when the languages were introduced in or before early adolescent years.

Implications: Learning any computer language is better than learning none. If programming is to be taught to students before early adolescence, it is advised to start with a graphical language. Future work should investigate the transition between different types of programming languages.

ARTICLE HISTORY
Received 21 March 2018
Accepted 3 November 2018

KEYWORDS
Programming and programming languages; first programming language; pedagogical issues; secondary education; teaching and learning strategies; postsecondary education

1. Introduction

Computer science (CS) courses provide opportunities for learners to develop the computational thinking capacities necessary for working in crucial sectors of

CONTACT Chen Chen chen.chen@cfa.harvard.edu Harvard-Smithsonian Center for Astrophysics

This article has been republished with minor changes. These changes do not impact the academic content of the article.

today's computation-dependent society. However, college introductory computer science courses (henceforth, we will generically label introductory tertiary CS courses CS1) have high attrition rates worldwide and are widely perceived to be difficult (Robins, Rountree, & Rountree, 2003; Watson & Li, 2014). As interest in making coding accessible to all K–12 learners has grown in recent years, increasing numbers of young people are entering college courses with some programming background and experience with programming languages (Lye & Koh, 2014), which may aid in increasing CS1 persistence rates. In this study, we examine how students' first programming languages influence both their attitudes about computing and their academic achievement in CS1 courses.

1.1. Graphical vs. textual programming languages

Over the last four decades, a great deal of CS education research has focused on predictors of success in CS courses (Lau & Yuen, 2011). A wide range of factors, from gender, computer attitudes, and experience (Charlton & Birkett, 1999) to mathematics background and game playing (Wilson, 2002), have been demonstrated to correlate with course performance. Other factors that are associated with success relate to self-efficacy: student perception of understanding (Bergin & Reilly, 2005), student effort (Ventura, 2005), and comfort level (Ventura, 2005).

Notably, several studies have demonstrated that prior programming experience can be a key predictor of success in introductory CS courses, as well as of novice attitudes towards computing. Wilson and Shrock (2001) found that formal learning experiences correlated positively with success, whereas Taylor & Mounfield (1994) discovered that all prior experiences with programming appeared to be beneficial for female-identified students, while only certain experiences benefited male-identified students. Prior experience has also been shown to affect perceived self-efficacy, which in turn affects course grades (Wiedenbeck, 2005). Tafliovich, Campbell, and Petersen (2013) suggested that the impact of prior experience "is more complex: it affects students' expectations, work habits, attitude and confidence, and perceptions of self and peers" (p. 244).

The extent to which prior programming experience predicts success, however, may depend upon what programming language a novice is first introduced to. The choice of first programming language has been shown to have lasting impact on how students understand key concepts or write programs, factors that could be important to later success in computing (McIver, 2000). Both Kunkle and Allen (2016) and Morrison and Newman (2001) found significant differences in student performance, based on differences in first programming language.

Consequently, the question of which language to teach first has long been a controversial topic. What Gal-Ezer and Harel (1998) called a "culture war" is

still debated today (Alexandron, Armoni, Gordon, & Harel, 2012; Weintrop & Wilensky, 2015a, 2015b). Some studies (e.g. Goosen, 2008) have contrasted student outcomes with object-oriented (e.g. C++, Java) and procedural languages (e.g. Pascal, C), while other studies (e.g. Robins et al., 2003) have questioned whether a first language ought to be one designed for learning or one used in industry. Taking a moderate stance in their study of the effects of CS1 language choices, Dingle and Zander (2000) emphasized that they were not saying that "any language is unusable or inappropriate, just that some have different effects than others" (p. 93).

As Robins et al. (2003), citing Gilmore and Green (1984), point out, studies of these different effects "should be seen in the context that there is not likely to be any universally 'best' programming notation for comprehension, but that a given notation may assist the comprehension of certain kinds of information by highlighting it in some way in the program code" (p. 145).

In this study, we focus on the difference between textual and graphical languages. Established languages such as C, C++, Python, and Java continue to be popular in both academia and industry (Ben Arfa Rabai, Cohen, & Mili, 2015; TIOBE, 2016). These languages are categorized as *textual programming languages* because they are written and modified in text editors. As such, textual programming languages require a novice to master not only the logical framework associated with the language, but also the language's idiosyncratic syntax – and syntactical fluency is typically required to produce even simple programs. Though textual programming languages may be more challenging to learn initially, some studies have shown that novices whose first programming language is textual may have an easier time transitioning to other textual languages (Enbody & Punch, 2010; Powers, Ecott, & Hirshfield, 2007). Because the majority of intermediate and advanced courses use textual languages, these studies suggest that a textual programming language may be preferable for introductory programming experiences.

However, recent years have seen a rise in development and use of *graphical programming languages*, particularly in K–12 classrooms. Some studies suggest that some graphical programming languages may be better for introductory experiences because they are easier for novices to use (Bau, Gray, Kelleher, Sheldon, & Turbak, 2017; Kelleher & Pausch, 2005; Malan & Leitner, 2007). Graphical languages vary greatly in how they represent the "code"/language itself: through drag-and-drop blocks, flowcharts, or diagrams (Dehouck, 2016; Nickerson, 1995). For the purposes of this study, we are primarily interested in block-based languages that have been explicitly designed for novice programmers, such as Alice, Storytelling Alice, Scratch, Snap!, and Blockly. In Scratch and similar languages, programs are composed by dragging and dropping specific blocks into logical sequences. Blocks lock together like puzzle pieces and can only be combined in syntactically appropriate ways, enabling novices to focus on the logic of their constructions rather than on syntax (Resnick et al.,

2009). Instead of being frustrated by syntactic errors, novices can explore advanced computational concepts more easily, which may promote greater "learnability" of programming and also greater diversity in computational participation (Bau et al., 2017; Kelleher & Pausch, 2005).

In addition to the potential effects of first programming languages on academic performance, these first experiences with computer programming may also impact students' attitudes toward CS and technology more broadly (Schulte & Knobelsdorf, 2007). As Lye and Koh (2014) suggest, increasing student interest in CS may increase CS1 enrollment. Anderson, Lankshear, Timms, and Courtney (2008) found that high school girls perceived CS to be boring and expressed a strong aversion to computers, while Statter and Armoni (2017) argued that teaching CS to middle school children could combat gender stereotypes about science, technology, engineering, and mathematics (STEM) careers. Furthermore, several studies suggest that only minimal CS exposure may be required to change attitudes (Lee et al., 2014; Du, Wimmer, & Rada., 2016;; Grover, Pea, & Cooper, 2014), though Hewner and Guzdial (2008) offered a rich example of introductory CS courses impacting only the attitudes of CS majors (who might already be positively predisposed to computing). Introductory experiences may affect students' attitudes about computing – and student attitudes, in turn, may predict academic success. Gomes, Santos, and Mendes (2012), for example, found strong correlations of students' performance with their personal perceptions of competence (in programming), as well as with their motivation.

Researchers and practitioners in computer science education have argued that graphical programming languages are particularly helpful for young learners because they are esteemed to be more enjoyable, less frustrating in terms of cognitive load, and to contribute to gently building intuition and interests regarding computational thinking (Bers, 2010; Brennan, Valverde, Prempeh, Roque, & Chung, 2011; Fessakis, Gouli, & Mavroudi, 2013; Sáez-López, Román-González, & Vázquez-Cano, 2016; Weintrop & Wilensky, 2017; Wilson & Moffat, 2010). Some advocates are so confident of the promise of graphical programming languages that they believe graphical languages should be the introductory languages for all programming novices regardless of age (Harvey & Mönig, 2010; Papadakis, Kalogiannakis, Orfanakis & Zaranix, 2014). Many college level introductory computer science courses (e.g. CS50 at Harvard University) have adopted graphical programming languages as a stepping stone to textual programming languages (Malan & Leitner, 2007; Rizvi & Humphries, 2012; Rizvi, Humphries, Major, Jones, & Lauzun, 2011; Uludag, Karakus, & Turner, 2011; Wolz, Leitner, Malan, & Maloney, 2009). Nevertheless, whether the effectiveness of graphical introductory languages is age sensitive remains a question uninvestigated.

Notwithstanding the immediate benefits of graphical first programming languages, many researchers are skeptical of their long-term benefits. Some argued that the lack of syntax in graphical languages engenders bad habits, such as bottom-up development, extremely fine-grained programming (Meerbaun, Armoni & Ben-Ari, 2011), overlooking syntax error (Powers et al., 2007), cluttered global variables, uncommunicative naming, long scripts, and duplicated or unused code (Techapalokul, 2017). Such habits were suspected to be counterproductive to the future transition to a formal programming language.

1.2. Research questions

Because there are many different possible predictors of success, understanding the connections between prior experience, attitudes, and student performance can be challenging. Because most previous studies have focused on only one or two (usually small) cohorts of students, their results have been difficult to generalize to larger populations. Based on a large, national U.S. sample, this article examines the differences in CS attitudes and achievement between those CS1 students whose initial programming language was textual (i.e. the *textual* group), those whose initial programming language was graphical (the *graphical* group), and those who have not yet learned any programming language (the *control* group). Specifically, we examine three central research questions:

- **RQ 1**: What is the relationship between first programming language (*textual, graphical*, and *control*) and attitudes about computing at the start of an introductory CS course (CS1)?
- **RQ 2**: How does academic success in CS1 courses vary by first programming language (*textual, graphical*, and *control*)?
- **RQ 3**: How do the above-mentioned potential effects of first programming language interact with the age of introduction?

Note that, in our study, the treatment of interest was the introduction of the first programming language. We investigate the treatment effect (and its interaction with the age of introduction) regardless of the events that occurred after the treatment. This will be further discussed in the Results and Limitations sections below.

2. Data and methods

2.1. Sample demographics

The following analysis uses data from Project FICSIT (Factors Influencing College Success in Information Technology). 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. enrolling in of 2-and 4-year colleges in three different size bins (small, medium and large). We contacted 1080 different institutions (279 two-year and 801 four-year) that offered introductory courses in CS in the fall semester of 2015; 138 of them agreed to participate (30 two-year and 108 four-year), and 118 of them (23 two-year and 95 four-year) actually returned the survey. Thus, our final sample consisted of 118 institutions. Within the first 2 weeks of class, 10,203 students filled out a 52-item, in-class survey relevant to their prior CS experience and current attitudes about computing, along with demographic questions. The students' final course grades were added by their instructors at the close of the semester. This set was comprised of 27% female (n = 2,660) and 73% male (n = 7,219) students, and the ethnic breakdown was 12% Hispanic,52% percent White non-Hispanic, 7% percent Black non-Hispanic, 23% percent Asian/Pacific Islanders non-Hispanic, and 6% percent other races (e.g. American Indian/ Alaska Natives, multiple races).

2.2. Predictor, outcomes, and controls

Our chief variable of interest is the students' first programming language, which was phrased as, "If you have any programming experience, what was the first programming language you learned? □ Alice □ Visual Basic □ Mindstorms □ Python □ Scratch □ Java □ C++ □ Other." Of those answering this question, 43% (N = 3,528) had no prior programming language experience (*control*), 5% (N = 406) had a graphical first programming language (*graphical)*, and 52% (N = 4282) had a textual first programming language (*textual*). Graphical programming languages included in the survey were Alice, Scratch, and Mindstorms. Textual programming languages included in the survey were Visual Basic, Python, Java, and C++ . Entries that answered "other" without specifying the programming language were excluded from the analysis.

Two outcome variables were examined: CS grade and CS attitude. CS grade is the final grade each student received at the end of the introductory computer science course, expressed on a 100-point scale (conversion was applied for institutions awarding letter grades: A+ = 98.5, A = 95, A- = 91.5, B+ = 88.5, B = 85, B- = 81.5, C+ = 78.5, C = 75, C- = 71.5, D+ = 68.5, D = 65, D- = 61.5, F = 40). Potential clustering effects were compensated by a hierarchical approach, as discussed in detail in the result section. CS attitude was measured by using the items on which students rated their attitudes toward CS, using a 6-point rating scale (from "strongly disagree" to "strongly agree"). Inspiration for some attitude items was drawn from the Computing Research Association (CRA) Undergraduate Survey and the Scientific Attitude Inventory (Moore & Hill Foy, 1997). Examples of attitude items include (see full list in Table 1):

Table 1. Summary of descriptive statistics of the covariates.

	First Programming Language		
	None	Textual	Graphic
Male	66.80%	76.50%	78.00%
Access to Computer at Home	91.50%	92.70%	96.00%
Went to Public High School	78.60%	76.60%	73.30%
High School Offered CS Course(s)	54.50%	66.90%	75.90%
First Language is English	69.00%	69.20%	76.60%
Parents' Jobs relate to CS	20.50%	25.50%	35.50%
Race. White	55.20%	58.30%	61.10%
Race. Black	9.50%	7.50%	10.20%
Race.Hispanic	12.20%	7.10%	5.40%
Race.Asian	23.10%	27.10%	23.30%
Parents can help with programming	6.09%	9.56%	14.35%
Parents Supportive of Learning CS (0 = "not supportive at all," 5 = "very supportive")	2.79(1.74)	3.27(1.67)	3.78(1.39)
Father Edu	2.37 (1.29)	2.61(1.26)	2.74(1.24)
Mother Edu (0 = "did not finish high school," 1 = "high school" 2 = "some college," 3 = "4 years college," 4 = "grad school")	2.35(1.22)	2.54(1.19)	2.71(1.19)
Age Introduced to Programming Language (1 = "< 6," 2 = "6–10," 3 = "11–14," 4 = "15–18")	NA	3.77(1.04)	3.34(1.04)

- I feel I belong in computer science.
- Computer science is boring for me.
- I feel comfortable doing computer science.
- Computer science is a creative activity.

Covariates matched for include gender, race/ethnicity, access to a computer at home, parental education, parents' use of computers in their vocation, parent support in the learning of computer science, if parents were able to provide help with programming, attending a public or non-public high school, if students were born in the United States, if their first language was English, and if any computer science course was offered in their high school. Those who had prior programming experience were asked to report the age of introduction (choosing from "< 6," "6–10," "11–14," "15–18"). This variable is matched for in the comparison between *graphical* and *textual* groups. Table 1 presents the descriptive statistics of the covariates (rows) before matching by first programming language groups (columns).

2.3. Matching

Establishing causal mechanisms can be challenging in observational studies, particularly because factors may covary with one another. In the case of this dataset, students who learn a computer language prior to college are more likely to have attended a high school that offered a CS course. Hence, modeling the grade earned in college computer science using the variable of earlier exposure to a computer language could be biased, because those who

attended a high school offering CS would have a greater probability of earlier exposure. In randomized experiments, we could randomize the assignment of students to different high schools (and other conditions), but this is generally impossible or unethical in educational studies. Hence, we can use *post-facto* methods to construct comparative groups with differing exposure to computer languages prior to college that otherwise have similar backgrounds. A direct application of ordinary regression cannot disentangle the confounding relationships. Ordinary regression analysis can also be skewed by extreme values at the tails of the regression line. This weakness becomes especially noticeable when sample sizes differ greatly, such as in our case, and covariates overlap poorly between groups – the regression line will be heavily leveraged by the group with more extreme values at the tails (see detailed explanation in Ho, Imai, King, & Stuart, 2007). One appropriate approach, known as *matching*, seeks to force covariates between two groups to be as similar as possible by choosing pairs of subjects (one from each group) who are as close to each other as possible on all the included covariates, and then discarding unmatched observations. When covariates between two groups are well balanced, no collinearity exists between covariates and group status, and an observational data set can therefore be approached as if created by random assignment. Although caution is still recommended before making any strong causal claims, matching before running a regression or *t*-test will strengthen the robustness and accuracy of the estimation of treatment effects. It is noteworthy that, although students were nested within different schools, multilevel regression was not applied because college enrollment happens after K–12 introductory programming experiences; accounting for school and any post-treatment variables would bias our inferences about the relationship between introductory experiences and student achievement (see thorough explanation in Montgomery, Nyhan, & Torres, 2018, and in; Rosenbaum, 1984).

Nearest neighbor matching and propensity score (pscore) matching appear to be the most widely used techniques (Thoemmes & Kim, 2011). Recent studies have shown that pscore matching often increases imbalance and bias (King & Nielson, 2016). Moreover, because of concerns that different matching techniques might produce varying results, methodologists have been advocating a comparison among the results of multiple matching methods in order to demonstrate the robustness of the estimation against the change of matching methods. For such reasons, we applied four common matching techniques (i.e. pscore matching, nearest neighbor matching, full matching, and optimal matching) and ascertained the degrees of consensus among them. We believe it is useful to apply multiple matching methods because each matching method trims the data based on different criteria. A thorough review of matching techniques and the case for comparing multiple techniques can be found in Stuart (2010).

Nearly all matching techniques require complete data entries, but some data were missing from the dataset. There were cases (N = 2998) in which at least one of the covariates was missing, as well as cases (N = 1441) where first programming language information was unknown (such as situations where students reported "other" computer languages without specification); these cases were excluded from the analysis. Thus, the analyzed sample size was 5764; 306 students had learned a graphical programming language first, 2995 had learned a textual language first, and 2463 had not learned any programming languages prior to CS1.

Because we have three values for the treatment variable (*graphical, textual,* and *control*), we separately applied matching between every two of the three, to generate three separate matched datasets (graphical and control; textual and control, graphical and textual). Afterward, we applied regression analysis on each of the matched datasets (M1 for *graphical* and *control*; M2 for *textual* and *control*; M3 for *graphical* and *textual*).

2.4. Principal component analysis

Twenty-three survey items related to student attitudes toward computer science were examined using principal component analysis, a common method in analyzing the pattern of correlations among a large number of variables. PCA projects each variable onto a few orthogonal components, such that the variance of the projected data is maximized (Hotelling, 1933).

3. Results

3.1. Examining the balance and choosing the sample

The balance of each covariate between the treatment and control groups was checked after applying each matching technique to ensure that matching had successfully generated a balanced sample. Before matching, 8 out of the 17 covariates had significant mean differences. This indicates that the exposure to early computing experiences depends to a substantial degree on the young persons' background and characteristics. Each of the four matching techniques generated a well-balanced dataset. t tests of the mean differences in covariates between groups demonstrated that all mean differences were minimal and nonsignificant (at $p \leq 0.05$). We first used the sample after optimal matching as our prototypical sample, because optimal matching was principally based on nearest matching, but performed better than nearest matching at minimizing the total distance between matched pairs (Gu & Rosenbaum, 1993; Stuart, 2010). At the end of this section, we also report the results from other matching methods, as a robustness check, to examine the consensus between different matching algorithms.

3.2. RQ1: student attitudes

To answer research question one, we applied principal component analysis of the 23 items measuring attitudes toward computing. The first component, termed *positive attitude* (eigenvalue = 9.12), successfully explained 38.0% of the variance, while the second component (eigenvalue = 1.60), only accounted for 6.4% of the variance. In the second component, only four items had loadings higher than 0.40, and those loadings were of similar size and direction as the respective loadings on the first component. Therefore, we concluded that one component solution sufficiently summarized the measurement of attitude towards CS. A full description of item-level loadings and contributions is provided in Table 2. For each observation, we computed the component values by summing the standardized score of each item weighted by their respective loadings from the PCA analysis. The component values were standardized to mean of 0 and variance of 1 for the ease of interpreting the effect sizes.

At the beginning of their CS1 courses, students who scored high on *positive attitude* tended to agree with items about being comfortable, confident, and belonging to CS, they considered CS to be relevant and useful for social communication, they also thought highly of the internal algorithms of computer programming. In general, these students were optimistic and excited about learning CS. Conversely, students who scored on the lower end of *positive attitude* tended to be less enthusiastic about CS.

As shown in Table 3 (regression analysis based on the sample after optimal matching), *Graphical* (M1: $\beta = 0.78$, s.e. = 0.18, $p < 0.001$) and *textual* (M2:

Table 2. Loading of each attitude item based on principal component analysis.

	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 don't 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

Table 3. Regression models predicting the positive attitude towards CS after applying optimal matching.

	Positive Attitude towards CS		
	M1 (Graphic vs. Control β(s.e.)	M2 (Textual vs. Control) β(s.e.)	M3 (Graphic vs. Textual) β(s.e.)
Treatment Effect			
(Graphic vs. Control)	0.78***(0.18)		
(Textual vs. Control)		1.08***(0.08)	
(Graphic vs. Textual)			−0.06(0.19)
Introduction age			−0.05*(0.02)
Gender	1.34***(0.22)	1.82***(0.09)	1.32***(0.23)
Father Education	−0.14(0.11)	0.03(0.05)	−0.03(0.11)
Mother Education	0.15**(0.11)	−0.06(0.05)	0.04(0.11)
Access to Computer at Home	2.58***(0.79)	0.23(0.21)	3.78***(0.94)
Race. Black	−0.64(0.33)	−0.29(0.15)	−0.97**(0.33)
Race. Asian	−0.12(0.24)	0.02(0.11)	−0.18(0.26)
Race. Hispanic	−0.01(0.31)	−0.43**(0.13)	−0.08(0.32)
Born in US	−0.78(0.27)	−0.24*(0.13)	0.10(0.29)
Parents' Job Relate to CS	−0.12(0.20)	−0.24(0.10)	−0.11(0.21)
Went to Public High School	−0.24(0.10)	−0.12(0.10)	0.03(0.20)
High School Offered CS	0.10(0.20)	−0.30*(0.14)	−0.22(0.23)
Parents Encourage CS	0.81***(0.09)	0.92***(0.04)	0.86***(0.09)
First Language is English	0.10(0.27)	−0.32(0.13)	0.20(0.29)
Parents Can Help w/Prog	0.42(0.26)	0.24(0.17)	0.05(0.27)
Intercept	−3.05***(0.85)	−1.19**(0.26)	−3.03**(1.02)

*** $p < 0.001$; ** $p < 0.01$;* $p < 0.05$;~ $p < 0.07$, after FDR adjustment.

β = 1.08, s.e. = 0.08, $p < 0.001$) groups scored significantly higher on *positive attitude* than did the *control* group. *Graphical* and *textual* groups did not differ from each other on *positive attitude* (M3: β = −0.06, s.e. = 0.19, $p = 0.36$). As shown in Table 5, the alternative matching methods reached similar conclusions about the differences between groups, although the estimation of the magnitudes of the effects differed somewhat by matching methods.

3.3. RQ2: student achievement in the CS1 course

Using CS1 final grade as the outcome and first programming language as the treatment, we performed regression analyses while controlling for the other covariates for which we had matched. The estimated effects of first programming language (*graphical, textual*, or *control*), using the optimal matching approach, are shown in Table 4.

We found significant differences for *graphical* versus *control* groups (M1: β = 2.84, s.e. = 0.87, $p < 0.01$, effect size = 0.18 sd) and *textual* versus *control* groups (M2: β = 1.54, s.e. = 0.37, $p < 0.001$, effect size = 0.09 sd).

The main treatment effect between the *graphical* and *textual* groups was not statistically significant, when no interaction effect was included (M3.1). However, after inclusion of the interaction effect between treatment (*graphical* vs. *textual*) and the age a student was introduced to the programming language (M3.2), we found both a significant treatment effect (β = 7.28, s.e. = 2.92, $p < 0.05$) and an interaction effect (β = −2.13, s.e. = 0.85, $p < 0.05$).

Table 4. Regression models predicting the final grade after applying optimal matching.

	M1 (Graphic and Control)		M2 (Textual and Control)		M3.1 (Graphic and Textual)		M3.2 (Graphic and Textual)	
	β	s.e.	β	s.e.	β	s.e.	β	s.e.
Treatment (Graphic vs. Control)	2.84**	0.87						
(Textual vs. Control)			1.45***	0.37				
(Graphic vs. Textual)					0.18	0.84	7.28*	2.92
Introduction age					1.05**	0.40	1.76***	0.47
Treatment × introage							−2.13*	0.85
Male	−0.08	0.99	−0.94	0.41	−0.66	0.98	−0.67	0.97
Father Education	1.34*	0.51	1.14***	0.25	0.88	0.52	0.87	0.52
Mother Education	0.20	0.50	0.11	0.21	−0.07	0.48	−0.13	0.48
Access to Computer at Home	−0.73	2.78	1.02	0.93	−6.31	3.92	−5.80	3.92
Race. Black	−5.22***	1.48	−5.51**	0.67	−4.12*	1.40	−4.08*	1.40
Race. Asian	−1.51	1.15	−0.30	0.50	0.17	1.21	0.11	1.20
Race. Hispanic	−1.74	1.38	−1.87**	0.61	−2.22	1.39	−2.25	1.39
Born in US	−0.22	1.37	0.62	0.60	1.86	1.30	1.84	1.30
Parents' Job Relate to Computer Science	0.38	0.87	−0.45	0.45	−0.65	0.86	−0.65	0.86
Went to Public High School	−1.94	0.91	−0.95	0.46	−0.87	0.91	−1.01	0.90
Parents Can Help w/Prog	3.11***	0.43	2.81***	0.19				
High School Offered CS Course(s)	−0.42	0.97	−0.35	0.38	0.59	1.00	0.49	1.00
Parents Encourage the Learning of CS	0.17	0.42	0.25	0.19	1.30**	0.41	1.33**	0.41
First Language is English	−0.39	1.25	−0.79	0.57	−0.54	131	−0.78	1.31
Intercept	87.48***	3.12	85.139***	1.14	90.21***	4.55	84.70***	3.80

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$, ~ $p < 0.08$.

Introduction age, father/mother education, and parents encourage the learning of CS have been rescaled to z score

Table 5. Robustness check using different matching methods.

Comparison Groups	Outcome	Before Matching	Pscore Matching	Nearest Matching	Full Matching	Optimal Matching
Graphic vs. Control	Final grade	2.54**	2.02*	2.42*	2.54**	2.68**
	Positive attitude	1.64***	0.95***	1.07***	1.64***	0.86**
Textual vs. Control	Final grade	1.35***	1.62***	1.62***	1.94***	1.67***
	Positive attitude	1.43***	1.11***	1.22***	1.43***	1.22***
Graphic vs. Textual	Final grade	3.23	6.36**	8.08**	4.55*	6.46**
	(Interaction effect with introage)	−0.76	−2.43**	−3.01**	−1.52*	−2.68*
	Positive attitude	0.20	0.07	−0.11	0.20	−0.13

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, ~ $p = 0.07$, all after FDR p-value adjustment

Figure 1 illustrates this interaction effect (with ± 1 s.e.) between the age of introduction event (x-axis) and programming language (color groups). The x-axis in Figure 1 indicates the age at which the first programming language was introduced, which was a one-time event and does not imply existence or non-existence of prolonged learning or language switching (information we do not know) after the event. To interpret the figure, for example, if a child learned a *graphical* language as the first language and was introduced to the *graphical* language at the age of 6, this child would be expected to score 6 points (effect size = 0.43 s.d.) higher than a child under the scenario that he/she learned a *textual* language as the first language and first learned it at the age of 6 (our matching method allowed us to make a reasonable argument that we were comparing a participant against his/her own counterfactual, in

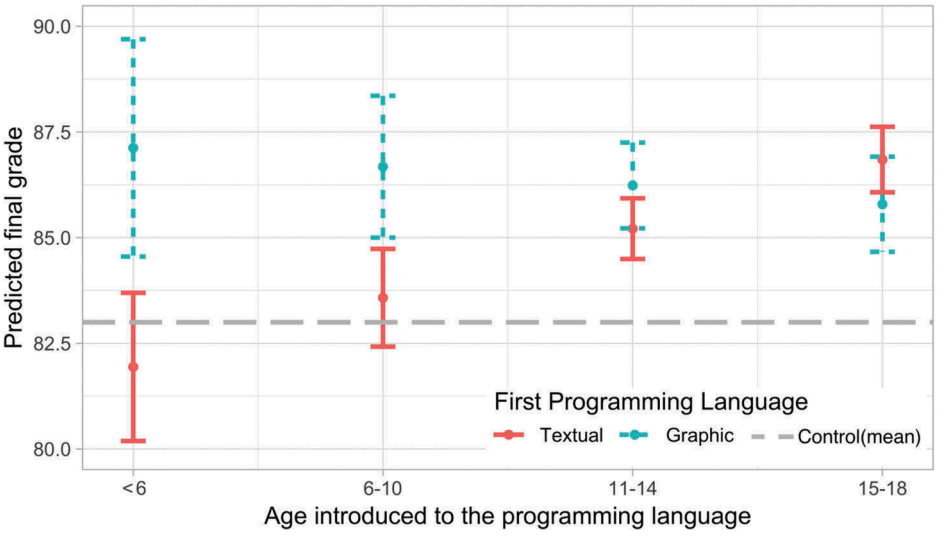

Figure 1. Interaction effect between introduction age and first language (*Graphic* vs. *Textual*) in predicting the final grade.

other words, to compare that participant with someone he/she was nearly identical to in terms of background information). Figure 1 should not be read as the progressive trajectory of participants who started with a *graphical* versus *textual* language, which would mislead the readers to imply that if one started with *textual* language, he/she would improve steadily, and if one started with *graphical* language, he/she would decline steadily. Although such a hypothesis was theoretically interesting and important, we could not test it given that (1) to reiterate, we did not have information about students' prolonged and posttreatment learning, and (2) we had a cross-sectional data structure, not a growth or time series data.

According to a post-GLM test, the *graphical* group scored higher than the *textual* group when the languages were introduced before 6 years old ($p = 0.02$, effect size = 0.43 s.d.) or between 6 and 10 years old ($p = 0.04$, effect size = 0.29 s.d.). The two groups did not differ from each other between 11 and 14, or 15 and 18 years old. Within the *textual* group, those who learned textual languages between 11 and 14, or between 15 and 18, scored higher than those who learned the textual languages before they were 6 years old ($p < 0.001$, effect size = 0.12 s.d.; $p < 0.001$, effect size = 0.25 s.d., respectively). In Figure 1, for illustration, we superimposed a reference line (dashed grey line) for the raw (unmatched) mean score of the *control* group. It appeared that the *graphical* group was consistently outperforming the mean of the *control* group, whereas the younger end of the *textual* group did not differ from the *control* group. Yet, note that this graph was plotted for the subsample of the matched data between *textual* and *graphical* groups, disregarding the *control*

group. When the *control* group was matched with the *graphical* or *textual* group, we only detected a main treatment effect. It is noteworthy that we could not estimate exactly the year of age at which equalization occurs because the introduction age was an ordinal variable, rather than a continuous variable.

The effects of the covariates were not the focus of this article. Therefore, we did not further interpret their coefficients. Nevertheless, once we included more covariates, we made more multiple comparisons, thus we arguably encountered a higher risk of type I error. We therefore reported in Table 4 all *p* values adjusted using the false discovery rate (FDR) method. If we had used Bonferroni adjustment, the most conservative *p* value adjustment method, the treatment, and interaction effects in M3 would not be significant; however, we believe such a method would too heavily penalize our effort to control all covariates (theoretically, controlling for covariates that have been matched for is not necessary because the covariates no longer affect the treatment assignment) and increase the chances for type II error.

The matching method literature (Montgomery et al., 2018), as mentioned above, advises not to control for clustering if the clustering happens after the treatment, because it has been mathematically proven (see Gelman & Hill, 2006, p. 188–190) that to match or to adjust using post-treatment variables would reintroduce nonrandomness to the model estimation, which would increase bias in our estimate of the treatment effect and counteract the effort of matching. This was the reason we did not apply multilevel analysis in the first place. However, considering that (1) students were clustered in schools, (2) each school might have different criteria grading students, and (3) when we converted letter grades to 100-point scale, the comparison might not be fair between institutes that graded on a 100-point scale and institutes that graded on a letter scale, we decided it was important to run a multilevel model to diagnose the magnitude of the clustering issue. Such a solution might introduce bias to our estimation of the effect size; however, it would at least provide a naïve estimation of the average effect among students graded by the same criteria within the same institution. A multilevel model showed the intraclass correlation rho = $3.05^2/(3.05^2 + 11.18^2) = 0.07$, which indicated that only an insubstantial proportion of the total variance in grade was accounted for by the clustering. In addition, the estimated effect of our key predictors in multilevel models remained nearly the same as those in the flat models (M1: $\beta_{treatment} = 2.62$, $p < 0.01$; M2: $\beta_{treatment} = 1.75$, $p < 0.001$; M3.2: $\beta_{treatment} = 4.86$, $p < 0.05$, $\beta_{interaction} = -2.04$, $p < 0.05$).

3.4. Robustness check

Different matching methods reached approximately the same conclusions, as shown in Table 5, despite slight differences in the magnitude of the effects.

The models in Table 5 did not control for the covariates that had been matched for. Once matching has resolved the confounding issue of the covariates (the covariates do not affect the treatment assignment after matching), adjusting for those covariates (as in Table 4) in regression models should not change the treatment effect in theory and should have minimal impact empirically. As argued above, there is no perfect matching method; therefore, our best estimation of the effects should range between the maximum and minimum effects shown in Table 5.

4. Discussion

In summary, our findings suggest that previous programming experiences, independent of language type (in terms of graphical/textual syntax), have positive effects on students' attitude and achievement in college introductory computer science coursework. Furthermore, our findings undermine the common assumption that having a first programming experience with a textual language (vs. a graphical language) generally provides a compelling advantage in subsequent courses that employ textual languages. On the other hand, these results do not support the conjecture that a graphical language is always superior to a textual language in offering a better foundation for the further development of programming skills. This was found to be true only for younger children in the age ranges of 6–10 years or younger. *Graphical* and *textual* languages have equivalently positive effects, if the language is acquired after the age of 10, compared with the group that has not learned any programming language. We concluded that the effects of first programming language type were age sensitive. It appears advantageous to introduce computing through *graphical* languages for young students, as many graphical languages were designed for young learning incorporating fun, social, storytelling, and visually appealing features (Brennan & Resnick, 2013; Dehouck, 2016; Mishra, Balan, Iyer, & Murthy, 2014). We did not find supportive evidence for the recent trend to use graphical languages as a starter for older students in high school (before college) from a programming performance perspective (Malan & Leitner, 2007; Rizvi & Humphries, 2012; Rizvi et al., 2011; Uludag et al., 2011; Wolz & Leitner, 2009), whereas students in their later teens become more adaptive to, and appreciative of, the higher cognitive load of *textual* languages. Previous studies have also shown that *graphical* languages were less appealing to some students in their later teens because such languages tended to be perceived as overprotective and inflexible for advanced and creative work (Cheung, Ngai & Chan, 2009; DiSalvo, 2014; Weintrop & Wilensky, 2015a). Because the questionnaire did not ask for specific ages, but for age ranges, our estimates cannot pinpoint a specific "critical age of 11" at which the effect of languages would change. Future studies should examine age effects with greater nuance.

It may seem counterintuitive that early learners of a *textual* language did worse than those who learned it in their late teens, but the early contact with a textual language may be somewhat overwhelming for a young learner's cognitive capacity and perhaps even counterproductive for later learning. For example, previous studies have shown that the teaching and learning of a textual language often focus on syntactic coding and overlook social interactions and problem-solving skills, which hampers young learners' interests and persistence in computer science (Kelleher & Pausch, 2005; Schollmeyer, 1996; Weintrop & Wilensky, 2015a).

The above findings have several direct implications: (a) learning any computer language is better than learning none: students should not wait till college to learn programming formally. Early exposure can pave the way to positive attitudes, with moderately large effects, and better grades, with moderately small effects. (b) If we wish to teach a student who is 10 years old or younger about programming, it is advised to teach him/her through using a *graphical* language rather than a *textual* language as a starter, which is expected to be more advantageous to the student's programming performance when he/she enters college, with moderately small effects. If we only care to cultivate a child's interest in, and positive attitudes toward, CS, either type of programming language can be adopted, with moderately large effects. (c) If we wish to teach an adolescent who is 11 years and older and who has never been exposed to any programming language, we can choose either a graphical language or a textual language. Both are expected to help the student to achieve equivalent outcomes in terms of CS attitude, with moderately large effects, and performance, with moderately small effect sizes. We note that our study cannot make suggestions about the environment and pedagogy of an introduction language, nor can our study make suggestions about the transition from an introduction language to other languages, because it was limited by the available data.

4.1. Limitations

The following are the three most pressing limitations of this work, which invite further investigation.

4.1.1. What happened after the introduction of the first programming language?

The survey in this study asked participants what their first programming language was and at what age this first experience occurred. While this gives insight into the starting point, it says nothing about the trajectory after the introductory event. If a student responds that he/she learned Scratch at age 7, we do not know if he/she extended learning or using Scratch for more years or if he/she learned Java afterwards. In other words, as noted above, we did not have information about students' posttreatment experience. The lack of this information is a major

limitation of the study if we intend to explain the mechanism behind the introductory effect. Based on the available information and analysis, the mechanism remained a black-box. Yet, recall that the aim of this study was to investigate the effect of introduction, a one-time treatment. The intricate trajectories and their effects remain to be investigated in future studies. Nonetheless, it is important to note that, statistically speaking, not controlling for the events that happened after the treatment does not bias our estimation of the treatment effect. The confounding variables were the pretreatment variables that could affect the probability of one's first programming language status, and we have matched such variables as much as we could. To further elaborate, each time we picked a student who learned Scratch at 7 years old, we matched this student with another one who learned Java at 7 years old as well, and both students had nearly identical pretreatment information. By forcing the pretreatment variables to be identical between the two students, such potentially confounding variables would not correlate with the group assignment, thus they were not confounding variables and would not bias our comparison between the two students anymore. There was indeed the possibility that one group learned in more depth, or longer duration, or more types of language, than the other groups, but such posttreatment experiences are endogenous within the effect of first language. Without such information and without a tailored design for the investigation of casual mediation effects (see Imai, Keele, Tingley, & Yamamoto, 2011), we could not parse the effect of posttreatment experience from the effect of the treatment itself. Nevertheless, the transition between programming languages remains an important question to be investigated in future studies, especially because graphical tools are often used as means to introduce textual programming languages.

4.1.2. Who is not represented – or is not represented sufficiently?

Selection bias is a concern; we know that dropout rates from CS1 courses are high, and students who dropped out immediately (i.e. even before the survey was administered) and students who withdrew from the course (i.e. did not receive a final grade) are not included in the sample. In addition to students who dropped out of the course, one also needs to consider (a) students who have "dropped out" of CS participation entirely – by not enrolling in a CS1 course, perhaps due to negative prior experiences with programming; and (b) students who have "skipped" CS1 and have immediately gone to an advanced CS course, perhaps due to advanced programming background. In other words, the effects found in this study hold for students who went on to take introductory CS in college, and we do not know if the effects are the same for other students.

Another important question to explore in this context would be to what extent prior exposure to computer languages (of different kinds) might boost participation in CS1 classes. Of those who did participate in the survey, the relatively small sample size in the graphical group may have contributed to not

detecting more differences between the graphical and textual groups. As interest in CS continues to increase at the K–12 level, and as graphical programming languages for K–12 learners become increasingly available, the number of students entering CS1 courses with graphical first experiences in programming may grow. In addition, although this study benefited from a large sample size, the analytic sample was only of moderate size due to matching trimming. Moreover, the samples of the *graphical* group and of students who were introduced to programming at a young age were relatively small. Future studies should consider boosting the sample size by strategically oversampling the *graphical* group and those who started programming at a young age.

4.1.3. What is the variation in context of the "first programming experience"?

The response to the survey item about prior programming experience (see Section 2.2) relies on a student's memory to retrospectively report information and, furthermore, offers no nuance about the nature of that first programming experience. Thus, respondents with the same "first programming experience" by the survey standard, may vary enormously in their total contact time with programming (and specific programming languages), sense of initiative and engagement with programming, and access to systems of support. Most importantly, we did not have any information about the classroom environment and pedagogical context in which the first programming experiences took place. Numerous studies have found these factors to be important from both theoretical and practical perspectives, as summarized below.

4.1.3.1. Possibility. What are learners invited to make? Are their creations meaningful, either personally or socially? Teaching with the languages more commonly used in industry and academia (primarily textual) better facilitates the inclusion of real-world examples that highlight the usefulness and breadth of programming. These languages are typically mature, full-featured, and, as a result, quite powerful, but because they were not designed for learners, they might not necessarily highlight and/or scaffold concepts for novices (Kölling, 2010). In contrast, educational programming languages, designed to be accessible to new learners, are often more limited in scope and functionality. Some of these languages, particularly the ones designed for younger learners (ages 8–12), also incorporate additional elements of engagement, fun, and expression, such as colorful visuals or a variety of sounds (Dehouck, 2016). Graphical programming languages like Alice, Storytelling Alice, and Scratch, which enable novices to easily tell stories, create games, or produce music videos (depending on their interests), have been shown to increase interest and enthusiasm (Kelleher, Pausch, & Kiesler, 2007; Mishra et al., 2014; Moskal, Lurie, & Cooper, 2004). But these educational learning environments may look very different from traditional ("real") languages, and learners (particularly

older students working with a language designed for younger audiences) might be skeptical of the environment and not fully understand how their mastery in educational programming languages applies to further study of programming (Abelson et al., 2014; Bau et al., 2017; Cheung, Ngai, & Lau, 2009; Powers et al., 2007).

4.1.3.2. Sociality. Software developers and other computer science experts in academia and industry often continue to learn from one another and collaborate in person (e.g. through meetups and hackathons) and online (e.g. through sites like GitHub and Stack Overflow) (Badashian, Esteki, Gholipour, Hindle, & Stroulia, 2014; Wang, Lo, & Jiang, 2013). To prepare students for the field of computer science, learning experiences should ideally enable novices to practice communication and collaboration skills, both on- and offline. Kafai and Burke (2013) describe this learning goal as "computational participation," pointing out that in-person experiences often incorporate pair programming, which has been demonstrated to be effective for novices (Denner & Werner, 2007). Novices learning Alice, Gamestar Mechanic, Greenfoot, Hopscotch, Kodu, or Scratch can participate in a language's individual community and forums. For young programmers, these online learning communities may be their first exposure to social media, and the communities provide developmentally appropriate spaces for young people to learn how to participate, critique constructively, and collaborate (Brennan & Resnick, 2013). In contrast, if novices start with languages used in industry, although a wealth of information and resources may be available, finding information that is aligned with the developmental level of the learner may be more difficult, and younger programmers may not be ready to engage with communities frequented by adult programmers with high levels of expertise.

4.1.3.3. Pedagogy. Teachers can play a critical role in broadening access to computing education in K–12 and beyond – creating classroom cultures that support computational literacy and that offer all students opportunities for computational creation (Brennan, 2013; Cooper, Grover, Guzdial, & Simon, 2014; Cuny, Baxter, Garcia, Gray, & Morelli, 2014; Goode, 2008; Goode, Chapman, & Margolis, 2012; Kafai & Burke, 2014; Margolis & Kafai, 2014; Margolis et al., 2012). When considering which programming language to teach with, it may be important to consider not only the students' needs and interests, but also the educator's needs. How is the teacher supported in using the programming language in the classroom? Some languages may have more robust communities of support for educators than others, particularly if those languages are tied to curricular standards (e.g. NSF's support of Java and the CS AP exam). Some languages have been used by educators more frequently, and there may be more curricular content developed for those languages. Some language developers have explicitly invested in educator

support, such as the Greenroom for Greenfoot and ScratchEd for Scratch (Brennan, 2013; Brown, Stevens, & Kölling, 2010).

In sum, keeping in mind the limitations of the study and the above generic considerations about the goal of teaching computer science in K–12, future studies should (1) strategically oversample the students who learned a graphical language as their first language; (2) conduct longitudinal observations of students' learning trajectories; (3) collect information about the variation in the CS learning experience, such as curriculum, environment, and pedagogy; and (4) as discussed in the methods section, be cognizant of the limits of causal claims. Although matching methods are commonly used in causal inference studies, the methods themselves do not guarantee the valid detection of causality. We should only accept the existence of causality if all confounding factors have been matched for, which was not the case in our study. Therefore, we want to explicitly caution against firm causal conclusions. Our key contribution was to reveal treatment and interaction effects, while excluding the possibility that these effects could be explained by a list of commonly suspected confounding factors. Thus, we made a strong argument for the plausibility of a causal link. However, because our exclusion of confounding factors was not exhaustive, future studies should either match for more factors or adopt other methods, such as experimentation. The abovementioned efforts should help CS education researchers to arrive at a more accurate estimation of the effect of different introductory languages and their interplay with the students' background, learning experience and context.

5. Conclusion

Efforts like CS for All and analogous out-of-school-time initiatives have increased the urgency of understanding how initial contact with computing at a relatively early age influences later success and participation in computing, whether as a CS professional or otherwise. Our study shows that prior programming experience is beneficial for both student attitude and achievement in computer science at the college level.

In terms of attitudes towards computer science, having some prior experience with programming is better than no prior experience, an outcome which is also reflected in the literature. In terms of achievement, we found age-sensitive effects of the type of first programming language. Graphical languages are particularly beneficial if introduced in the age ranges of 6–10 years or younger. They appear to be a suitable means to teach programming skills to children at a young age (early adolescent or earlier). Our findings also undermine the common myth that having a first programming experience with a textual language generally provides a compelling performance advantage in subsequent courses that employ textual languages. This was found to be true only for older children. To forestall

misinterpretation, it is worth reiterating that we did not find that learning graphical-based programming languages was in all respects better than learning textual-based languages. We only found this to be true with respect to grades in introductory CS in college, but not to attitudes towards CS, and only when learners were introduced to graphic languages at or before the age of 10.

Future research on novices' attitudes and outcomes, particularly as CS concepts continue to be introduced in K–12, may need to focus on more than just the language itself, considering aspects of the environment as a whole, such as support for educators, the peer community of learners, and the goals and contexts for these early programming experiences.

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 excellent 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 on this project as members of our Advisory Board: Hal Abelson; Lecia Barker, Chair of Advisory Board; Randy Battat; Joanne Cohoon; Maria Litvin; Clayton Lewis; Irene Porro; Kelly Powers; Lucy Sanders; Susanne Steiger–Escobar; and Jane Stout. 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.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the National Science Foundation [1339200]; Scratch Foundation.

Notes on contributor

Chen Chen is a postdoctoral fellow at the Science Education Department of the Harvard-Smithsonian Center for Astrophysics.

ORCID

Chen Chen http://orcid.org/0000-0002-6065-8889

References

Abelson, H., Baldwin, L., Friedman, M., Lipman, D., Sheldon, J., Sherman, M., & Wolber, D. (2014). Real programmers use blocks – A new definition of who is a programmer [Blog post]. Retrieved from http://blog.csta.acm.org/2014/10/28/real-programmers-use-blocks-a-new-definition-of-who-is-a-programmer/

Alexandron, G., Armoni, M., Gordon, M., & Harel, D. (2012). The effect of previous programming experience on the learning of scenario-based programming. *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli, Finland.

Anderson, N., Lankshear, C., Timms, C., & Courtney, L. (2008). 'Because it's boring, irrelevant and I don't like computers': Why high school girls avoid professionally-oriented ICT subjects. *Computers & Education*, *50*(4), 1304–1318.

Badashian, A. S., Esteki, A., Gholipour, A., Hindle, A., & Stroulia, E. (2014). Involvement, contribution and influence in GitHub and stack overflow. *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, Markham, Ontario, Canada.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM*, *60*(6), 72–80.

Ben Arfa Rabai, L., Cohen, B., & Mili, A. (2015). Programming language use in US academia and industry. *Informatics in Education*, *14*(2), 143–160.

Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, St. Louis, Missouri, USA.

Bers, M. U. (2010). The TangibleK Robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice*, *12*(2), n2.

Brennan, K., & Resnick, M. (2013). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In N. Lavigne & C. Mouza (Eds.), *Emerging Technologies for the Classroom: A Learning Sciences Perspective* (pp. 253–268). New York, NY: Springer.

Brennan, K. (2013). Learning computing through creating and connecting. *Computer*, *46*(9), 52–59.

Brennan, K., Valverde, A., Prempeh, J., Roque, R., & Chung, M. (2011). More than code: The significance of social interactions in young people's development as interactive media creators. In *EdMedia: World Conference on Educational Media and Technology* (pp. 2147–2156). Association for the Advancement of Computing in Education (AACE), Lisbon, Portugal .

Brown, N., Stevens, P., & Kölling, K. M. (2010). Greenroom: A teacher community for collaborative resource development. *Proceedings of the Fifteenth Annual Conference on Innovation and technology in computer science education*, Bilkent, Ankara, Turkey.

Charlton, J. P., & Birkett, P. E. (1999). An integrative model of factors related to computing course performance. *Journal of Educational Computing Research*, *20*(3), 237–257.

Cheung, J. C. Y., Ngai, G., Chan, S. C. F., & Lau, W. W. Y. (2009). Filling the gap in programming instruction: A text-enhanced graphical programming environment for junior high students. *SIGCSE Bulletin*, *41*(1), 276–280.

Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A future for computing education research. *Communications of the ACM*, *57*(11), 34–36.

Cuny, J., Baxter, D. A., Garcia, D. D., Gray, J., & Morelli, R. (2014). CS principles professional development: Only 9,500 to go!. *Proceedings of the 45th ACM technical symposium on Computer science education*, Atlanta, GA.

Dehouck, R. (2016). The maturity of visual programming. Retrieved from http://www.craft.ai/blog/the-maturity-of-visual-programming/

Denner, J., & Werner, L. (2007). Computer programming in middle school: How pairs respond to challenges. *Journal of Educational Computing Research*, *37*(2), 131–150.

Dingle, A., & Zander, C. (2000). Assessing the ripple effect of CS1 language choice. *Journal of Computing Sciences in Colleges*, *16*(2), 85–93.

DiSalvo, B. (2014). Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE Computer Graphics and Applications*, *34*(6), 12–15.

Du, J., Wimmer, H., & Rada, R. (2016). "Hour of Code": can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice*, *15*, 52–73.

Enbody, R. J., & Punch, W. F. (2010). Performance of python CS1 students in mid-level non-python CS courses. *Proceedings of the 41st ACM technical symposium on Computer science education*, Milwaukee, WI.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97.

Gal-Ezer, J., & Harel, D. (1998). What (else) should CS educators know? *Communications in the ACM*, *41*(9), 77–84.

Gelman, A., & Hill, J. (2006). *Data analysis using regression and multilevel/hierarchical models*. New York, NY: Cambridge university press.

Gilmore, D. J., & Green, T. R. G. (1984). Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, *21*, 31–48.

Gomes, A. J., Santos, Á. N., & Mendes, A. J. (2012). A study on students' behaviours and attitudes towards learning to program. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, Haifa, Israel.

Goode, J. (2008). Increasing Diversity in K-12 computer science: Strategies from the field. *ACM SIGCSE Bulletin*, *40*(1), 362–366.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47–53.

Goosen, L. (2008). A brief history of choosing first programming languages. *History of Computing and Education 3 (HCE3)*, *269*, 167–170.

Grover, S., Pea, R., & Cooper, S. (2014). Remedying misperceptions of computer science among middle school students. *Proceedings of the 45th ACM technical symposium on Computer science education*, Atlanta, Georgia, USA.

Gu, X. S., & Rosenbaum, P. R. (1993). Comparison of multivariate matching methods: Structures, distances, and algorithms. *Journal of Computational and Graphical Statistics*, *2*(4), 405–420.

Harvey, B., & Mönig, J. (2010). Bringing "no ceiling" to scratch: Can one language serve kids and computer scientists. *Constructionism*, *2010*, 1–10.

Hewner, M., & Guzdial, M. (2008). Attitudes about computing in postsecondary graduates. *Proceedings of the Fourth International Workshop on Computing Education Research*, Sydney, Australia.

Ho, D. E., Imai, K., King, G., & Stuart, E. A. (2007). Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference. *Political Analysis*, *15*(3), 199–236.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417.

Imai, K., Keele, L., Tingley, D., & Yamamoto, T. (2011). Unpacking the black box of causality: Learning about causal mechanisms from experimental and observational studies. *American Political Science Review*, *105*(4), 765–789.

Kafai, Y. B., & Burke, Q. (2013). The social turn in K–12 programming: Moving from computational thinking to computational participation. *Proceedings of the 44th ACM technical symposium on Computer science education*, Denver, Colorado, USA. http://dl.acm.org/citation.cfm?doid=2445196.2445373.

Kafai, Y. B., & Burke, Q. (2014). *Connected code: Why children need to learn programming*. Cambridge, MA: MIT Press.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37*(2), 83–137.

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling Alice motivates middle school girls to learn computer programming. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose, California, USA.

King, G., & Nielsen, R. (2016). Why propensity scores should not be used for matching. *http://j.mp/1sexgVw*. *Download Citation BibTex Tagged XML Download Paper, 378*.

Kölling, M. (2010). The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 14.

Kunkle, W. M., & Allen, R. B. (2016). The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Transactions on Computing Education (TOCE)*, *16*(1), 3.

Lau, W. W. F., & Yuen, A. H. K. (2011). Modelling programming performance: Beyond the influence of learner characteristics. *Computers & Education*, *57*(1), 1202–1213.

Lee, M. J., Bahmani, F., Kwan, I., LaFerte, J., Charters, P., Horvath, A., & Beswetherick, M. (2014). Principles of a debugging-first puzzle game for computing education. *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Melbourne, Australia.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K–12? *Computers in Human Behavior*, *41*, 51–61.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin 39*(1), 223–227.

Margolis, J., & Kafai, Y. (2014). Why the 'coding for all' movement is more than a boutique reform. *Washington Post*. Retrieved from https://www.washingtonpost.com/news/answer-sheet/wp/2014/10/17/why-the-coding-for-all-movement-is-more-than-a-boutique-reform.

Margolis, J., Ryoo, J. J., Sandoval, C. D., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads*, *3*(4), 72–78.

McIver, L. (2000). The effect of programming language on error rates of novice programmers. *12th Annual Workshop of the Psychology of Programming Interest Group*, Cozenza, Italy (pp. 181–192).

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, Darmstadt, Germany (pp. 168–172).

Mishra, S., Balan, S., Iyer, S., & Murthy, S. (2014). Effect of a 2-week scratch intervention in CS1 on learners with varying prior knowledge. *Proceedings of the 2014 conference on Innovation & technology in computer science education*, Uppsala, Sweden.

Montgomery, J. M., Nyhan, B., & Torres, M. (2018). How conditioning on posttreatment variables can ruin your experiment and what to do about it. *American Journal of Political Science*, *62*(3), 760-775.

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

Morrison, M., & Newman, T. S. (2001). *A study of the impact of student background and preparedness on outcomes in CS I*. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (SIGCSE '01)* (pp. 179-183). New York, NY: ACM. doi:10.1145/364447.364580.

Moskal, B., Lurie, D., Cooper, S., Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, Norfolk, VA.

Mounfield, L. C., & Taylor, H. G. (1994). Exploration of the relationship between prior computing experience and gender on success in college computer science. *Journal of Educational Computing Research*, *11*(4), 291–306.

Nickerson, J. V. (1995). *Visual programming*. New York, NY: New York University.

Papadakis, S., Kalogiannakis, M., Orfanakis, V., & Zaranis, N. (2014). Novice programming environments. Scratch & app inventor: A first comparison. *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*, Albacete, Spain (p. 1).

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: Teaching CS0 with Alice. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE '07)* (pp. 213-217). New York, NY: ACM. doi:10.1145/1227310.1227386.

Resnick, M., Maloney, J., Monroy-Hernandéz, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Rizvi, M., & Humphries, T. (2012). A Scratch-based CS0 course for at-risk computer science majors. In *Frontiers in Education* Conference *(FIE), 2012*, Seattle, WA (pp. 1–5).

Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CS0 course using scratch. *Journal of Computing Sciences in Colleges*, *26*(3), 19–27.

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

Rosenbaum, P. R. (1984). The consequences of adjustment for a concomitant variable that has been affected by the treatment. *Journal of the Royal Statistical Society. Series A (General)*, *147*(5), 656–666.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129–141.

Schollmeyer, M. (1996). Computer programming in high school vs. college. *ACM SIGCSE Bulletin*, *28*(1), 378–382.

Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. *Proceedings of the third international workshop on Computing education research*, Atlanta, GA.

Statter, D., & Armoni, M. (2017). Learning abstraction in computer science: A gender perspective. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, Nijmegen, Netherlands (pp. 5–14).

Stuart, E. A. (2010). Matching methods for causal inference: A review and a look forward. *Statistical Science: A Review Journal of the Institute of Mathematical Statistics*, *25*(1), 1.

Tafliovich, A., Campbell, J., & Petersen, A. (2013). A student perspective on prior experience in CS1. *Proceeding of the 44th ACM technical symposium on Computer science education*, Denver, CO.

Techapalokul, P. (2017). Sniffing through millions of blocks for bad smells. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Seattle, WA (pp. 781–782).

Thoemmes, F. J., & Kim, E. S. (2011). A systematic review of propensity score methods in the social sciences. *Multivariate Behavioral Research*, *46*(1), 90–118.

TIOBE. (2016). TIOBE Index for August 2016. Retrieved from https://perma.cc/93D3-JJ5Q.

Uludag, S., Karakus, M., & Turner, S. W. (2011). Implementing IT0/CS0 with scratch, app inventor forandroid, and lego mindstorms. *Proceedings of the 2011 conference on Information technology education*, West Point, NY (pp. 183–190).

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

Wang, S., Lo, D., & Jiang, L. (2013). An empirical study on developer interactions in StackOverflow. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, Coimbra, Portugal.

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on Innovation & technology in computer science education*, Uppsala, Sweden.

Weintrop, D., & Wilensky, U. (2015a). To block or not to block, that is the question: Students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children*, Boston, MA (pp. 199–208).

Weintrop, D., & Wilensky, U. (2015b). Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. *Proceedings of the 11th Annual International Computing Education Research (ICER) conference*. New York, NY: ACM.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 3.

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. *Proceedings of the first international workshop on Computing education research*, Seattle, WA: USA.

Wilson, A., & Moffat, D. C. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. *Proceedings of the 22nd Annual Psychology of Programming Interest Group*. Leganés, Spain: Universidad Carlos III de Madrid

Wilson, B. C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, *12*(1–2), 141–164.

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, Charlotte, North Carolina, USA.

Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with scratch in CS 1. *ACM SIGCSE Bulletin*, *41*(1), 2–3.