More Time or Better Tools? A Large-Scale Retrospective Comparison of Pedagogical Approaches to Teach Programming

Gabriela Silva-Maceda, P. David Arjona-Villicaña, *Member, IEEE*, and F. Edgar Castillo-Barrera, *Member, IEEE*

***Abstract*—Learning to program is a complex task, and the impact of different pedagogical approaches to teach this skill has been hard to measure. This study examined the performance data of seven cohorts of students ($N = 1168$) learning programming under three different pedagogical approaches. These pedagogical approaches varied either in the length of the introductory programming block of courses (two or three semesters) or in the programming tool used in the first semester (C language or the programming support tool Raptor). In addition, gender and initial course selection differences were investigated. Raw pass rates under the three pedagogical approaches were compared; they were also compared, controlling for initial ability levels, using a logistic regression. Results showed that a more extensive duration of the introductory block produced a higher pass rate in students, but changing the programming tool used did not. Raw gender differences were not statistically significant; admission phase differences were initially statistically different, but not once initial ability and pedagogical approach received had been accounted for. Findings are discussed in relation to existing literature.**

***Index Terms*—Admission phase, computer science education, gender differences, learning programming, pedagogical approaches, programming curriculum.**

I. Introduction

LEARNING to program is a difficult task because it requires that students master different higher-order cognitive skills, such as problem solving, developing and applying mental or mathematical models, generating pseudocode and algorithms that could solve a problem, and learning the syntax and semantics necessary to code in a programming language. In general, learning programming is a time-consuming process, which requires effort and dedication from beginners. Therefore, it is not uncommon that many students feel frustrated and abandon their Computer Science (CS) programs [1], [2].

In search of solutions to this complex problem, researchers have looked into the impact of diverse *pedagogical approaches* and into the predictive power of *student factors*. Pedagogical approaches are the different instructional strategies and tools that faculty use to facilitate student learning, while student factors are naturally occurring student characteristics that can be predictive of student performance.

The study described here takes advantage of curricular changes in the teaching of introductory programming in two CS programs comparing student performance differences under three pedagogical approaches. These changes allowed for the possibility of retrospectively evaluating the relative importance of two pedagogical choices: the inclusion of a preprogramming course and the use of a specific type of programming support tool. As secondary analyses, the collected data provided an opportunity to compare two student factors: gender and initial major selection.

This paper is organized as follows: First, the background and work related to the current study are analyzed in Section II, and the curricular structure for the university where this study was carried out is described in Section III. Then, the methodology employed for this study is described in Section IV, and the statistical results are shown in Section V. Finally, these results are discussed in Section VI.

Manuscript received July 22, 2015; revised November 23, 2015 and January 28, 2016; accepted February 13, 2016. Date of publication March 24, 2016; date of current version October 27, 2016. This work was supported in part by different grants from the Professional Development Program for Teachers (PRODEP) and the National Council of Science and Technology (CONACYT) of Mexico.

G. Silva-Maceda is with the School of Psychology, Universidad Autónoma de San Luis Potosí, 78000 San Luis, Mexico (e-mail: gabriela.silva@uaslp.mx).

P. D. Arjona-Villicaña and F. E. Castillo-Barrera are with the School of Engineering, Universidad Autónoma de San Luis Potosí, 78000 San Luis, Mexico.

Digital Object Identifier 10.1109/TE.2016.2535207

II. Background and Related Work

This section describes a brief review of the literature in learning programming, covering pedagogical approaches, student factors, and methodological issues that have a direct impact on what results are obtained and which kind of inferences can be drawn. Special attention has been given to studies using robust research design and analyses.

A. Pedagogical Approaches in Learning Programming

Comprehensive reviews of pedagogical approaches can be found in the literature. While some reviews focus on the programming paradigm and the language used to teach introductory programming [3], [4], others focus on more specific intervention strategies to support learning [5], such as collaborative learning and pair programming.

The literature is mixed, concerning which of the two most common programming paradigms, i.e., procedural or object-oriented, is better suited to teach programming. Using results from actual student performance data, Wiedenbeck *et al.* [6] found that procedural programming allows for an easier learning experience for novice programmers. However, the paradigm is closely related to the programming language each institution selects for instruction, and that evidence is also inconclusive. While some argue that students should benefit from learning a programming language that is common within the industry [7], [8], there is also evidence that languages with a simple syntax, such as Python or Eiffel, facilitate student learning [9], [10]. In the context of this study, the language selected by the institution was C, i.e., a procedural language, and the variations investigated were whether C was introduced in the first or the preprogramming course.

Indeed, the introduction of a preprogramming course, which is sometimes called CS0, has received extensive attention in the literature [5], [11], [12]. Faux [12] showed that students who took a preprogramming course teaching problem-solving, algorithm development, pseudocode generation, and diagramming were able to use pseudocode more consistently than another cohort. Moreover, a review of various pedagogical approaches by Vihavainen *et al.* [5] showed that the addition of a CS0 course, in combination with other strategies, provided one of the best improvements in pass rates when compared to other approaches. The evidence from these studies suggests that adding a preprogramming course may improve the learning programming process, and a recent survey indicated that a CS0 course is currently a common practice among universities [13].

Another pedagogical approach that has been researched is the use of programming support tools. Crews and Butterfield [14] introduced a flowchart simulator software, which was called FLowchart INTerpreter or FLINT, with the aim of developing students' programming skills. Two experiments revealed that: 1) paper flowcharts are useful for novice programmers to abstract problems, as compared to no flowcharts; 2) FLINT was useful for developing programming logic and coding skills, as compared to no software use. Because of the research design of their experiment, this is strong evidence that flowcharts aid in developing programming skills. Nonetheless, the authors did not compare the performance measures between paper flowcharts and FLINT; hence, the question remains whether software simulators are indeed better for improving student learning. More recently, Carlisle *et al.* [15] have evaluated student performance using a software simulator, which is called Raptor, against coding. Nevertheless, the evidence was inconclusive: For some tasks, the use of Raptor produced better performance than MATLAB/Ada, while for other tasks it did not. One of the pedagogical approaches in the present study employs Raptor in the preprogramming course.

It is interesting to note that most implementations of a programming support tool coincide with adding a preprogramming course [5], which makes it difficult to measure the impact of each approach on improving students' learning outcomes. The way the pedagogical approaches were implemented in the study reported here provided the opportunity to disentangle the effect of each of these variables separately.

B. *Student Factors in Learning Programming*

In addition to the pedagogical approaches, student factors have also been studied in relation to performance in CS [16], [17]. Among the student characteristics most examined are gender, mathematics background, previous programming experience, and motivation. The present data set permitted an evaluation of whether gender and a proxy for motivation had an influence on learning to program.

Women are notoriously underrepresented in science fields [2]. Specifically for CS, it has been documented that they are not only underrepresented, but also more likely to leave their CS program than their male peers [18]. Indeed, in a study of attrition [19], women leaving the CS program were more likely than men to feel that they did not belong. Even when underrepresented and more likely to leave, most of the evidence suggests that, when comparing performance using grades, there is no gender gap between women and men in CS courses [17], [20]. Although scarcer, evidence for gender differences in CS courses does exist [16].

Most of the evidence for the lack of discrepancy between female and male CS students comes from studies carried out in industrialized countries, such as the United States, whose Gender Development Index (GDI) places it on the first of five group rankings in terms of parity in life expectancy, education, and standard of living [21]. However, it is reasonable to expect gender differences in countries placed third among the GDI rankings, such as Mexico.

In regard to motivation, this construct could not be derived directly from any variable in the data set, but the records contained a variable that could function as a proxy for motivation: whether the student had chosen CS as their major, therefore being admitted in the first admissions phase, or if it was their second choice and they were admitted in the second phase. Studies examining the relationship between admission phase and achievement are scarce. The faculty's perception, at the institution examined in the present study, was that low motivation in second-phase students led to a lower performance in learning programming. The data set studied here provided an opportunity to examine whether the assumed performance differences existed.

C. *Methodological Issues*

The inferences derived from the studies mentioned earlier depend on the quality of the research design and analyses. Previous reviews [4] have identified the relatively small proportion of studies in the area of learning programming that use quantitative methods and, among those that do, the even smaller proportion where data are analyzed using inferential statistics.

Methodologically sound studies in learning programming mainly fall into one of two categories: experimental or longitudinal. Experimental studies involve the creation of two groups where members have been randomly assigned. The experimental group receives the manipulation of one independent variable to measure its effect on another dependent variable, while the control group does not [22]. It is usually considered that the experimental design produces the best evidence for inferring causality [23]. Although there are some experiments in introductory programming changing specific strategies [14], experiment designs addressing more complex questions are more difficult to carry out in a university context, due to ethical and logistical issues regarding random assignments.

Longitudinal studies follow the same students, for some time, and try to predict performance either from initial student factors [16], [17] or from pedagogical interventions that vary across different cohorts [9], [15], [24]. The second group aims to compare the end-of-term performance of different successive cohorts after changing one or several factors at a time. Although valuable insights can be gained from this method, these particular studies changed not only the pedagogical intervention, but also the programming language or tool used in the course (from Java to Python [9], from MATLAB to Raptor [15] or from C to Python [24]). Therefore, it can be argued that these studies changed the substance of the assessment used as an outcome measure, which makes the inferences derived from these analyses less robust. The present study maintains the same learning objectives in the same language as the outcome measure for all cohorts at the end of three programming courses.

Beyond the changes in outcome measures, a potential threat to the validity of any study is the presence of confounding variables. These variables are those that are not measured in the study but can still have an effect on the outcome [22]. In the education field, one of the main confounding variables is the student's initial ability level, which in the higher education context is generally measured by entrance examinations [25]. For example, the SAT and the ACT in the United States have been found to be significantly correlated to grade point average (GPA) [26].

An important consideration when analyzing performance data, particularly when different cohorts were taught with different approaches, is that it is quite possible for these generations to have differing initial ability; this makes any inference about the causes of better performance inconclusive because it is not possible to ensure that variations are coming from the pedagogical strategies. The CS studies reviewed here either assume that students in different cohorts are similar [9] or do not consider this variable [15], [24]. Although variation in student initial ability levels cannot be changed, a way to address this issue is to statistically control for this factor. The inclusion of initial ability as a means to control the cohort variability is at the core of the present study.

The curricular changes undertaken for the purpose of increasing student performance in this study are described in Section III.

III. Curricular Modifications

In order to improve student pass rates, curricular modifications were carried out in a large public university in Central Mexico. Students enrolled with the Computer Science Department at the School of Engineering majored in either computer engineering or a degree that is a mixture of computer science and information systems (informatics engineering). Although students are required to learn to program in more than one language and use both programming paradigms throughout their programs, the curricular changes described here only applied to the introductory programming courses.

The grading system at this university is based on a scale from 0 to 100. A passing grade for each course is equal to 60 or above. The final grade is usually an average of the marks obtained in tests given regularly during the course. Students who cannot obtain a final passing grade in a regular course can take two extraordinary written tests in a further attempt to pass. In the context of this study, passing means that a student was able to obtain 60 or more, regardless of when this grade was obtained, whether during a regular course examination or any of the two extraordinary tests.

TABLE I
Curricular Modifications

Approach	2&C	3&C	3&Raptor
Cohort(s)	2005 to 2007	2008 to 2010	2011
1st semester	–	IP with C	IP w/ Raptor
2nd semester	DSA	DSA	DSA
3rd semester	DSB	DSB	DSB

Students at this institution are required to master procedural programming and data structures. Two courses have been employed to teach both aspects of programming: *Data Structures and Algorithms A* (DSA) and *Data Structures and Algorithms B* (DSB). Throughout the curricular changes, DSB has remained mostly unmodified. Moreover, faculty members teaching this course have expressed that the learning objectives and course content have remained the same. It must be noted that, although an ideal assessment would have consisted of having the same final exam for all groups and for all cohorts, in practice, each faculty member created his own exams. DSB covers the following topics using the C programming language: pointers, linked lists, basic graph representation, and trees. Hence, students are considered to have mastered basic programming skills once they have passed this course. Therefore, all programming courses leading to and including DSB will be referred to here as the *programming block*.

Three different pedagogical approaches were used that varied either the length or the support tool for teaching programming in the first course. Section III-A–C describe the curricular changes taken to implement these approaches in the programming block.

A. Two Courses With C

The original curriculum taught data structures and C language at the same time in two semesters (DSA and DSB). Students who passed DSB were considered ready to move to the next level, i.e., object-oriented programming with C++. This strategy is labeled as 2&C, and three cohorts are included under this approach for this study (see Table I).

The learning objective of the DSA course, with this strategy, is to be an introduction to C and basic data structures. Therefore, the following topics were included: introduction to the C language and its control commands (if, for, while, etc.), arrays, *Structs*, stacks and queues, recursion, basic sorting and searching algorithms, and types of files.

B. Three Courses With C

Since the first pedagogical approach was not considered successful by the faculty, the next approach tried to help students' learning process by adding a preprogramming course whose learning objective was only to use the C language and its control commands; this course was named *Introduction to Programming* (IP). Data structures and pointers were not included in this course.

DSA's learning objective changed to teaching how to employ static data structures and pointers, while DSB was left almost unchanged. This second approach is labeled as 3&C and was implemented in the Fall of 2008. Three cohorts were included under this approach.

Although many students passed the new IP course, the overall faculty impression was that this strategy was not working as expected; hence, a third approach was developed.

C. Raptor and Two Courses With C

This approach stemmed from the idea that students would find it easier to learn programming skills from a graphic support tool than they would by being taught a programming language. After some research [27], Raptor was selected as the most appropriate tool for teaching algorithmic thinking. Raptor [15] is a programming support tool, which allows students to build their own graphical algorithms, test them, and verify if they work as intended. This tool does not use a programming language; instead, it has a user-friendly interface, which displays algorithm simulations.

Under this approach, the learning objective of IP changed to introduce the algorithmic thinking needed to learn programming. The DSA course would again introduce C language and its control commands; the expectation was that this would facilitate students' comprehension of the topics after taking the new IP course. This new approach is labeled 3&Raptor, and it was implemented in Fall 2011.

IV. Methodology

This study compared three pedagogical approaches, by applying statistical analyses to retrospective data of several CS student cohorts followed longitudinally, in order to address the following research questions:

1) Is student performance improved by having a preprogramming course?
2) Is student performance improved by using a programming support tool instead of a programming language?
3) Are two student characteristics, i.e., gender and initial major selection, predictive of student performance in learning programming?
4) Do any of the differences in pedagogical approaches or student characteristics remain once accounting for differences in initial ability levels?

A. Database

The database contained information for seven cohorts of students admitted in each successive year from 2005 to 2011. Every cohort was followed for up to three and a half years to complete the programming block. The original database included students who did not attend any of the first-semester courses that they registered for. Only students attending at least half of their courses in their first semester were included in the study. The final record contained full information for 1168 registered students (22.3% female and 77.7% male). Each cohort received one of the three pedagogical approaches shown in Table I, which shows that each subsequent approach only implemented one change: either it varied the length of the programming block by adding a preprogramming course, or it varied the programming tool used in the first course.

The admission process at this institution asks students to list the programs to which they want to be admitted to, in order of preference. Those who did not get into their preferred program in the first admission round have the choice of entering an alternative one in a second round if there are still places available. More than a tenth of the students in the final record were admitted in the second round (13.6%).

The database contained information about entrance examination scores, attendance, gender, grades, admission phase (first or second), and passing or not passing DSB.

B. Predictor Variables

Four variables were selected as predictors of student performance. The first was the *pedagogical approach* used, which varied the duration of the programming block or the initial programming tool used. Two variables, i.e., *gender* and *admission phase*, were also used as predictors.

In addition, one of the examination scores was selected using a statistical analysis as the *initial ability* indicator, to be used as a control variable. The statistical analysis in Section V-A describes how this fourth predictor variable was obtained.

C. Outcome Variables

The main outcome variable was a categorical variable indicating *passing* or *not passing* the final course of the programming block (DSB). It must be noted that *not passing* included those failing as well as those who dropped out before completing the programming block, either because they moved to a different department within the university or because they dropped out of the university altogether. Unfortunately, the distinction between students who moved and those who actually left could not be made due to how the information from the registry was recorded since different schools have independent registry departments.

For the preliminary analysis to select an initial ability level variable, the first-semester GPA was the outcome variable. Only the first-semester GPA was considered for this analysis because this is the only semester when all students of the same cohort are evaluated on the exact same courses. After the first semester, student variation on the courses that they registered for depended on how many courses they failed in their first semester (i.e., how many repeat courses they subsequently had to take); hence, the following semesters' GPA would not be equivalent within the same cohort. Furthermore, a three-semester average GPA was not considered since some students took longer than others to complete the introductory programming block.

V. Results

This section is divided into three parts. The first describes the preliminary analysis carried out to select the best indicator for

TABLE II
PEARSON CORRELATIONS FOR ENTRANCE EXAMINATIONS AND FIRST-SEMESTER GPA

	1	2	3	4	5
1. First sem. GPA	–				
2. EXANI-II	.418*	–			
3. Sch. of Eng. exam	.463*	.580*	–		
4. Reasoning skills	.226*	.550*	.366*	–	
5. Global score	.482*	.921*	.834*	.633*	–

$*p < .001$

initial ability levels, which was then used as a control variable in the main analyses. The second describes the main analyses evaluating the differences among the three pedagogical approaches and offers a statistical analysis of these differences controlling for initial ability levels. The third presents the results of the secondary analyses comparing student performance by gender and admission phase.

A. Preliminary Analysis: Selection of the Initial Ability Variable

As described in Section IV, an initial ability indicator had to be selected. A correlation analysis was undertaken to examine how four entrance examination scores (three tests and a composite) were associated with the students' first-semester performance in all courses.

Table II shows the Pearson correlation values linking the entrance examinations to first-semester GPA.

The four entrance examination scores represent three exams and the composite score: 1) an SAT-type test called EXANI-II; 2) a knowledge test designed by the School of Engineering; 3) a reasoning skills exam; and 4) a global score encompassing the previous three exams with the weights of 40%, 45%, and 15%, respectively. All correlations were positive, indicating that a higher score on any of the tests or the composite was associated with a higher first-semester GPA. The highest correlation was for the global score; therefore, this was chosen as the initial ability variable.

B. Comparison of Pedagogical Approaches

After selecting the control variable, the three pedagogical approaches were compared in terms of passing rate percentages: First, the comparison was performed without controls and then controlling for initial ability. Fig. 1 illustrates the raw pass rates for each approach. The highest performance was for 3&Raptor with a 28.8% pass rate, which was followed closely by 3&C with 26.4% and then by 2&C with 12.1%.

These differences were found to be statistically significant ($\chi^2 = 40.4$, $df = 2$, $p < 0.001$). According to this analysis, there was a statistically significant association between the kind of pedagogical approach received and the passing or not-passing rate. To examine individual effects, χ^2 standardized residuals were examined [28]. These residuals indicated that the greater rate of not passing 2&C was statistically different ($z = 2.1$, $p < 0.05$) from the rest. Also significant were the greater rates of passing for both 3&C and 3&Raptor ($z = 3.1$, $p < 0.001$ and $z = 2.4$, $p < 0.01$, respectively). Finally, the lower rate of passing under 2&C was highly significant ($z = -4.1, p < 0.001$). Taken together, these individual effects show that the approaches spread into three courses were significantly better for increasing the pass rate than the two-course approach.

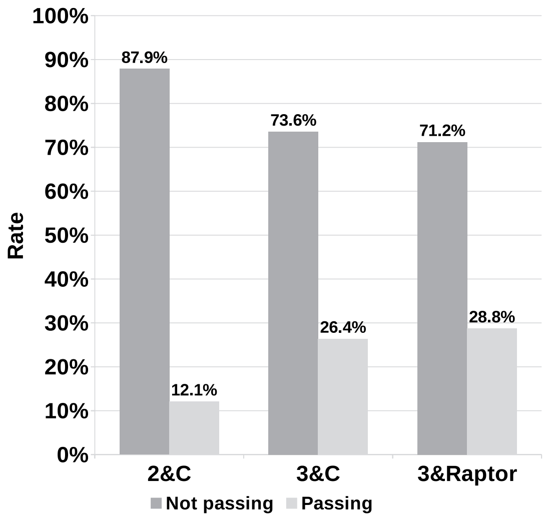

Fig. 1. Rates of passing and not passing for each approach.

TABLE III
LOGISTIC REGRESSION TO PREDICT PASSING FROM GLOBAL SCORE

	B (Standard error)	95% CI for odds ratio Lower limit	Odds ratio	Upper limit
Constant	-8.28 (.60)			
Global admission score	.14** (.01)	1.13	1.15	1.18

CI: Confidence Interval; $R^2 = .26 (Nagelkerke)$; Model $\chi^2 = 208.76$, $df = 1$, $p < .001$; $^{**}p < .001$

Having already selected the global score as a measure of initial ability levels, a logistic regression was carried out to examine how much variability was predicted by initial ability on its own for all 1168 students, regardless of the pedagogical approach received. Table III shows that the global score coefficient ($b = 0.14$, $p < 0.001$) was significant, which confirms that initial ability is a significant predictor for passing. In addition, this analysis found that this model accounts for a 26% variability in the pass rate.

Then, a model was performed, predicting the probability of passing from the three pedagogical approaches, this time accounting for initial ability. This process compares two of the variables to one reference; in this case, the first approach 2&C was chosen as the reference. Results from this analysis, as shown in Table IV, show that, even after controlling for initial ability, being a student under the approach 3&C was statistically different compared to the reference ($b = 0.95, p < 0.001$). It also shows that being a student under the approach 3&Raptor was statistically different, in terms of passing, when compared to 2&C ($b = 0.79$, $p < 0.01$). In other words, both approaches of the introductory programming block in three courses were statistically better at predicting pass rates than the two-course approach.

TABLE IV
LOGISTIC REGRESSION TO PREDICT PASSING FROM PEDAGOGICAL APPROACH, CONTROLLING FOR GLOBAL SCORE

	B (Standard error)	95% CI for odds ratio: Lower limit	Odds ratio	Upper limit
Constant	-8.95 (.62)			
Global admission score	.14** (.01)	1.13	1.15	1.18
2&C	(reference)			
3&C	.95** (.19)	1.78	2.58	3.74
3&Raptor	.79* (.25)	1.34	2.21	3.63

CI: Confidence Interval; $R^2 = .29(Nagelkerke)$;
Model $\chi^2 = 236.59$, $df = 3$, $p < .001$; $^*p < .01$, $^{**}p < .001$

TABLE V
LOGISTIC REGRESSION TO PREDICT PASSING 3&C VERSUS 3&RAPTOR, CONTROLLING FOR GLOBAL SCORE ($N = 618$)

	B (Standard error)	95% CI for odds ratio: Lower limit	Odds ratio	Upper limit
Constant	-8.16 (.77)			
Global admission score	.14** (.02)	1.13	1.15	1.18
3&C	(reference)			
3&Raptor	−.16 n.s. (.24)	0.54	0.85	1.36

CI: Confidence Interval; $R^2 = .25(Nagelkerke)$;
Model $\chi^2 = 114.34$, $df = 2$, $p < .001$; $^{**}p < .001$

Table IV also shows odds ratios of 2.58 and 2.21 for 3&C and 3&Raptor, respectively, compared to the reference 2&C. This means that, with initial ability levels being equal, a student who was taught in 3&C was 2.58 times more likely to pass than a student taught in 2&C. Likewise, with initial ability levels being equal, a student under 3&Raptor was 2.21 times more likely to pass than a student in 2&C. This new model with both initial ability and pedagogical approaches accounts for 29% of variance.

The previous model was only able to make two comparisons against the chosen reference (2&C). It remained to be seen how 3&C and 3&Raptor would compare against each other. For that purpose, a new logistic regression model was conducted exclusively with the 3&C and 3&Raptor cohorts (new $N = 618$). In this analysis, 3&C functions as a reference and 3&Raptor is compared against it. This new model, as shown in Table V, shows that being a student under the 3&Raptor approach was not statistically different compared to the reference ($b = -0.16$, $p > 0.05$) when controlling for initial ability levels.

C. Gender and Admission Phase Differences

Raw gender differences are shown in Fig. 2. These percentages show that men have a higher pass rate than women. However, these differences were not statistically significant ($\chi^2 = 2.11$, $df = 1$, $p = 0.14$). Given the lack of statistical significance, it was not necessary to examine these differences further with controls.

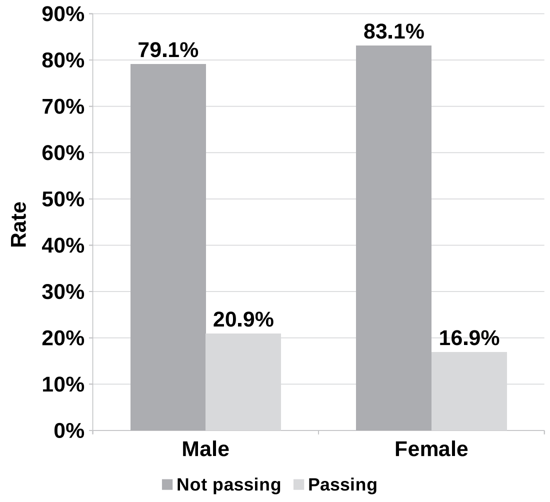

Fig. 2. Rates of passing and not passing by gender.

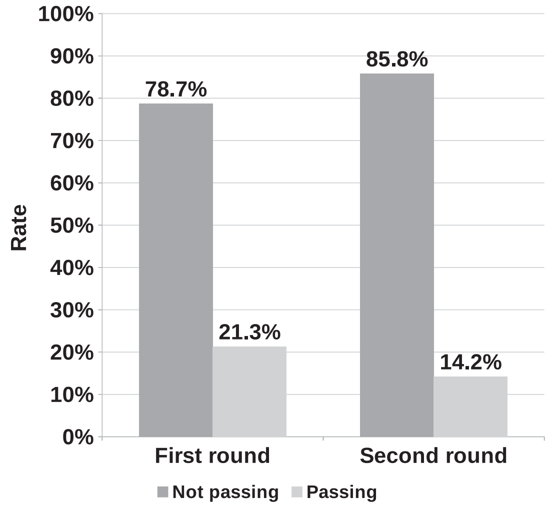

Fig. 3. Rates of passing and not passing per admission round.

Of the students in the sample, 937 had chosen one of the two CS programs as their first choice and were admitted in the first admission round, while 148 got into computing in the second admission round. Raw differences in performance were examined between *first* and *second rounds* (new $N = 1085$). The new N value excluded some students who transferred to the program from another school within the university or another university.

Fig. 3 shows that the pass rate for first-round students was higher (21.3%) than for second round (14.2%). These differences were statistically significant ($\chi^2 = 4.03$, $df = 1$, $p < 0.05$). However, an examination of standardized residuals revealed that none of the individual effects were significant (all $p > 0.05$). The discrepancy between the χ^2 results and the individual effects suggests generalized small differences, rather than a single effect.

Nonetheless, a logistic regression analysis was conducted to see whether these differences remained after controlling

TABLE VI
LOGISTIC REGRESSION TO PREDICT PASSING FROM ADMISSION ROUND, CONTROLLING FOR GLOBAL SCORE AND PEDAGOGICAL APPROACH ($N = 1085$)

	B (Standard error)	95% CI for odds ratio Lower limit	Odds ratio	Upper limit
Constant	-8.53 (.64)			
Global admission score	.13** (.01)	1.12	1.14	1.17
2&C	(reference)			
3&C	.85** (.20)	1.59	2.33	3.42
3&Raptor	.73* (.26)	1.25	2.08	3.46
1st round	(reference)			
2nd round	$-.07$ n.s. (.27)	.54	.93	1.58

CI: Confidence Interval; $R^2 = .27(Nagelkerke)$;
Model $\chi^2 = 203.08$, $df = 4$, $p < .001$; $^*p < .01$, $^{**}p < .001$

for both initial ability and pedagogical approach since they influence the rate of passing the programming block. This new model, as shown in Table VI, reveals that being admitted in the second round was not statistically different from being admitted in the first round ($b = -0.07, p > 0.05$).

VI. DISCUSSION

The longitudinal retrospective study described here includes some of the desirable research characteristics proposed by Pears *et al.* [4] in the field of teaching introductory programming: 1) it is a large-scale study, based on empirical results; 2) it is systematic in the sense that the one single change introduced in each variation of a pedagogical approach made it possible to disentangle the effects of extending the course duration and of changing the initial programming tool used; and 3) it is systematic in how the data were analyzed, using inferential statistics. In addition, a contribution of this study has been to control for one of the main confounding variables in any teaching and learning study, i.e., initial ability levels.

The curricular changes undertaken at this university allowed for a comparison of pedagogical approaches that varied either in the length of the introductory programming block or in the tool used. Pass rates demonstrated that both pedagogical approaches extended over three courses were more effective than the condensed two-course approach and, when considering the initial ability levels, these results were statistically significant. This same analysis showed that the 3&Raptor pedagogical approach was slightly more successful in increasing the passing rate than the 3&C approach. However, once taking into account the initial ability levels, the differences between these approaches were not significant.

Taken together, both sets of results seem to suggest that, rather than the specific tool to introduce programming, the key variable increasing the pass rate is the extended duration of the introductory programming block. This finding is consistent with existing evidence [5], [12] that a CS0 course increases performance. The results also quantified the benefit of adding a precourse, which can substantially increase the pass rate (see Table IV).

However, the nonsignificant findings reported here on the use of flowchart simulators were inconsistent with a previous study suggesting a statistically significant improvement with the use of flowchart software simulators for both Raptor [15] and FLINT [14]. Still, it must be noted that the Raptor cohort had a slight increase in the pass rate, but this was not statistically significant, even less so after controlling for initial ability; with these statistical controls, it could be argued that the present analysis shows the most robust findings.

Another finding of this study was that it was able to quantify the contribution of initial ability, as measured by entrance exams, to performance in learning programming. Indeed, initial ability was able to predict 25% of the variability in the pass rate for the introductory programming block at this institution. In other words, previous education matters, but it is not a determining factor since there is still 75% of variance unexplained. The finding that the pedagogical approach used can explain an additional 3% of variability might seem minimal in the context of all factors influencing learning, but is still encouraging, particularly if only pedagogical approaches are considered, given that it can substantially increase the passing rate, as described earlier.

In regard to gender, the analysis revealed that, even if female students in these CS majors were underrepresented (22%) and studying in a country with wider gender gaps, they were as successful as their male counterparts. This evidence converges with findings in other countries, which have found no statistical differences in success by gender [17], [20], and contrasts to others that have found them [16].

Finally, the proxy measure for motivation used in the present study, being admitted in the first or second round, was not significantly different once accounting for initial ability and pedagogical approach received. This suggests that initial major selection is not important for acquiring programming skills, a finding that stands in contrast to faculty expectations.

Although the large scale of the data set and the gradual curricular modifications allowed for an optimal comparison of pedagogical approaches, these were applied to different cohorts of students, and the analysis was done retrospectively. Although care was taken to control for initial ability levels, it is still possible that other differences in cohorts not considered here might be responsible for the variations in performance. For example, changes in faculty members, variations in the difficulty level of evaluations, and cohort differences in prior exposure to programming could all have played a role in the changes in performance observed here. Among these possible confounding factors, variations among teachers, in particular, could have a substantial impact on students' learning [29].

Although faculty attempted to ensure consistency of teaching and assessment methods, these could not be controlled for in the statistical analyses since they were not documented. Such is the drawback of a retrospective study, and any future study designed prospectively should ensure that these are documented for inclusion in further analyses.

VII. Conclusion

The implication for curriculum design from the comparison of pedagogical approaches is that extending the instructional time allocated to learn programming, rather than changing the initial programming tool used, will result in higher pass rates for learning introductory programming skills. Evidently, this finding needs to be replicated at other institutions in other countries before a definitive conclusion can be made.

The implications of this study for performance in regard to student factors are that initial ability levels are important, while being female or being admitted in a second round of admission is not. Since differences existed when looking at raw data, but these were not significant once they were evaluated using inferential statistics, it is necessary that studies comparing student performance go beyond stating differences in descriptive terms. In particular, given that initial ability is such an important performance predictor, it becomes necessary to take this variable into account in any analysis comparing different populations.

Finally, for CS faculty members and administrators, the results from this study highlight the need to measure the impact of curricular changes in order to support an evidence-based decision-making process conducive to substantive student performance improvements.

Acknowledgment

The authors would like to thank S. Nava-Muñoz and A. Ramos-Blanco, for kindly providing records from the registry; Robert Bryce, for editing suggestions; and the anonymous reviewers, for their valuable comments.

References

[1] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in *Proc. Conf. ITiCSE*, 2004, pp. 39–44.

[2] "Science and engineering indicators 2012," Nat. Sci. Found. (NSF), Arlington, VA, USA. [Online]. Available: http://www.nsf.gov/statistics/seind12/start.htm

[3] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Comput. Sci. Educ.*, vol. 13, no. 2, pp. 137–172, 2003.

[4] A. Pears *et al.*, "A survey of literature on the teaching of introductory programming," *SIGCSE Bull.*, vol. 39, no. 4, pp. 204–223, Dec. 2007.

[5] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review of approaches for teaching introductory programming and their influence on success," in *Proc. ICER*, 2014, pp. 19–26.

[6] S. Wiedenbeck, V. Ramalingam, S. Sarasamma, and C. L. Corritore, "A comparison of the comprehension of object-oriented and procedural programs by novice programmers," *Interacting Comput.*, vol. 11, no. 3, pp. 255–282, Jan. 1999.

[7] M. de Raadt, R. Watson, and M. Toleman, "Introductory programming: What's happening today and will there be any students to teach tomorrow?" in *Proc. Aust. Conf. Comput. Educ.*, 2004, pp. 277–282.

[8] A. Dingle and C. Zander, "Assessing the ripple effect of CS1 language choice," *J. Comput. Sci. Colleges*, vol. 16, no. 2, pp. 85–93, Jan. 2000.

[9] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching introductory programming: A quantitative evaluation of different approaches," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, pp. 1–28, Feb. 2014.

[10] L. Mannila and M. de Raadt, "An objective comparison of languages for teaching introductory programming," in *Proc. Baltic Sea Conf. Comput. Educ. Res.*, 2006, pp. 32–37.

[11] M. Rizvi, T. Humphries, D. Major, M. Jones, and H. Lauzun, "A CS0 course using Scratch," *J. Comput. Sci. Colleges*, vol. 26, no. 3, pp. 19–27, Jan. 2011.

[12] R. Faux, "Impact of preprogramming course curriculum on learning in the first programming course," *IEEE Trans. Educ.*, vol. 49, no. 1, pp. 11–15, Feb. 2006.

[13] S. Davies, J. A. Polack-Wahl, and K. Anewalt, "A snapshot of current practices in teaching the introductory programming sequence," in *Proc. ACM Tech. Symp. Comput. Sci. Educ.*, 2011, pp. 625–630.

[14] T. Crews and J. Butterfield, "Using technology to bring abstract concepts into focus: A programming case study," *J. Comput. Higher Educ.*, vol. 13, no. 2, pp. 25–50, Mar. 2002.

[15] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, "Raptor: A visual programming environment for teaching algorithmic problem solving," *SIGCSE Bull.*, vol. 37, no. 1, pp. 176–180, Feb. 2005.

[16] S. Bergin and R. Reilly, "Programming: Factors that influence success," *SIGCSE Bull.*, vol. 37, no. 1, pp. 411–415, Feb. 2005.

[17] B. C. Wilson and S. Shrock, "Contributing to success in an introductory computer science course: A study of twelve factors," *SIGCSE Bull.*, vol. 33, no. 1, pp. 184–188, Feb. 2001.

[18] J. M. Cohoon, "Toward improving female retention in the computer science major," *Commun. ACM*, vol. 44, no. 5, pp. 108–114, May 2001.

[19] M. Biggers, A. Brauer, and T. Yilmaz, "Student perceptions of computer science: A retention study comparing graduating seniors v.s. CS leavers," *SIGCSE Bull.*, vol. 40, no. 1, pp. 402–406, Mar. 2008.

[20] S. Beyer, "Predictors of female and male computer science students' grades," *J. Women Minorities Sci. Eng.*, vol. 14, no. 4, pp. 377–409, 2008.

[21] "Human development report," United Nations Develop. Programme, New York, NY, USA, 2015. [Online]. Available: http://hdr.undp.org/en/composite/GDI

[22] G. M. Breakwell, J. A. Smith, and D. B. Wright, *Research Methods in Psychology*, 4th ed. London, U.K.: SAGE, 2012.

[23] B. G. Tabachnick and L. S. Fidell, *Using Multivariate Statistics*, 5th ed. London, U.K.: Pearson, 2007.

[24] U. Nikula, O. Gotel, and J. Kasurinen "A motivation guided holistic rehabilitation of the first programming course," *ACM Trans. Comput. Educ.*, vol. 11, no. 4, pp. 24:1–24:38, Nov. 2011.

[25] T. R. Coyle and D. R. Pillow, "SAT and ACT predict college GPA after removing *g*," *Intelligence*, vol. 36, no. 6, pp. 719–729, Nov. 2008.

[26] J. L. Kobrin, B. F. Patterson, E. J. Shaw, K. D. Mattern, and S. M. Barbuti, "Validity of the SAT for predicting first-year college grade point average," The College Board, New York, NY, USA, Rep. 2008-5, 2008.

[27] F. E. Castillo-Barrera, P. D. Arjona-Villicaña, C. A. Ramírez-Gámez, F. E. Hernández-Castro, and M. Sadjadi, "Turtles, robots, sheep, cats, languages, what is the next to teach programming: A future developer's crisis," in *Proc. Int. Conf. FECS*, 2013, pp. 248–253.

[28] A. Field, *Discovering Statistics Using IBM SPSS Statistics*, 4th ed. London, U.K.: SAGE, 2013.

[29] H. McBer, "Research into teacher effectiveness: A model of teacher effectiveness," Dept. Educ. Employment, Nottingham, U.K., Res. Rep. 216, 2000. [Online]. Available: http://www.education.gov.uk/publications/eorderingdownload/rr216.pdf

Gabriela Silva-Maceda, biography not available at the time of publication.

P. David Arjona-Villicaña (M'15), biography not available at the time of publication.

F. Edgar Castillo-Barrera (M'08), biography not available at the time of publication.