A Longitudinal Evaluation of a Best Practices CS1

Adrian Salguero, Julian McAuley, Beth Simon, and Leo Porter
University of California San Diego

ABSTRACT

Over a decade ago, the CS1 course for students without prior programming experience at a large research-intensive university was redesigned to incorporate three best practices in teaching programming: Media Computation, Pair Programming, and Peer Instruction. The purpose of this revision was to improve the quality of the course, appeal to a larger student body, and improve retention in the major. An initial analysis of the course indicated an increase in pass rates and 1-yr retention of students in the major. Now that time has passed and those students impacted by the revision have had time to graduate, this longitudinal study revisits and expands on these prior findings through examining student outcomes over a twelve year period (2001 through 2013). The student outcomes examined include failure rates in CS1, retention rates in the major, rates of switching into the major, time to degree, and performance in subsequent major courses. We compare these findings against similar metrics collected for another CS1 course at the same institution that caters to students with prior programming experience and did not make changes during this same time period. Overall, the inclusion of media computation, pair programming, and peer instruction corresponds to a significant improvement in passing rates for CS1 as well as retention of majors from CS1 through graduation. In turn, there is no indication that this larger group of students experienced any harm in terms of lower grades in upper-division courses or their time to degree.

CCS CONCEPTS

• **Social and professional topics → Computing Education**.

KEYWORDS

peer instruction, media computation, pair programming

ACM Reference Format:
Adrian Salguero, Julian McAuley, Beth Simon, and Leo Porter. 2020. A Longitudinal Evaluation of a Best Practices CS1. In *Proceedings of the 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3372782.3406274

1 INTRODUCTION

The primary goal of this study is to determine the long-term impact of a redesign of a CS1 course at UC San Diego in 2008. For context, those in the computer science field are well aware that the number of computer science majors has risen and declined over time. The rise of the dot-coms at the end of the 1990s correlated with a surge in interest in computer science and related fields, with the number of bachelor's degrees in computing peaking in the United States in 2003 [27]. Following that peak, the number of students interested in computer science suffered a precipitous decline. To respond to that decline, computer science programs began to focus on how best to attract and retain students into the computer science discipline. Within this context, our institution began a concerted effort to overhaul our CS1 course for students without prior experience in order to reduce failure rates, improve retention of students in computing majors, and to attract new students to the major [36].

The faculty driving the course revision set forth a mandate to adopt best practices from the computing education and science education communities. After reviewing possible practices, the course was redesigned to include three practices recognized within these communities. The first was to ensure computing was being taught in a context meaningful to students through the use of Media Computation [11]. The second sought to actively engage students in core course concepts during "lecture" through the use of Peer Instruction [7]. The third was to create a community among students by having them engage in Pair Programming [51]. We note that less was known about the efficacy of each of these practices in computing than is known now; Section 3 provides the context for the changes at that time.

Prior work by Porter and Simon reported on the success of this course revision—specifically showing that the course redesign significantly lowered failure rates while also resulting in more students taking courses in the major one year after taking the introductory course [35]. However, because the evaluation of the new course was done only a few years after the course restructuring, there are a number of questions that could not be answered then that are possible to answer now: Did the increase in retention in computing courses 1-year later persist to increase retention at the time of graduation? How well did students from this larger group of majors perform in later courses? Were the changes more beneficial for different demographics of students? And can we explain the improved outcomes simply by the changes over time?

In this work, we have extracted over 17 years of CS1 student data to evaluate the impact of the course redesign on passing rates, retention of majors to the point of graduation, the number of students switching into computing majors, student performance in required upper-division computing courses, and time-to-degree. Using student demographic data, we are able to evaluate how the course redesign impacted male and female students as well as students from under-represented minority groups (URM).[1] Moreover, a fortunate facet of our computer science curriculum is that the major has long had two alternative paths into the major. The first

ICER '20, August 10–12, 2020, Virtual Event, New Zealand

ACM ISBN 978-1-4503-7092-9/20/08…$15.00
https://doi.org/10.1145/3372782.3406274

[1] Similar to the Special Report on "Women, minorities, and persons with disabilities in science and engineering" [28] we define URM as Chicano, Latino, African-American, American Indian, and Alaska Native.

path, for students *without* prior programming experience, is the path with the course redesign. The second path, for students *with* prior programming experience, experienced no significant changes to its curriculum over the same time. By comparing student results for both paths, we are able to explore whether the benefits experienced after the course redesign are better explained by changes over time or by the redesign itself.

The contributions of this comprehensive longitudinal evaluation of incorporating Peer Instruction, Media Computation, and Pair Programming in an introductory computing course include:

- Student passing rates and retention of majors improved significantly after the course redesign.
- There is no evidence that this larger population of students suffered worse outcomes later in terms of grades in subsequent courses or time-to-degree.
- Outcomes for students from underrepresented groups in computing (women, URMs) improved, as did outcomes for students from represented groups.
- The benefits for students are better explained as a result of the course redesign rather than possible changes over time.
- Instructors who taught both before and after the revision experienced similar benefits for their students.

2 BACKGROUND

In this section we describe prior work related to the relevant best practices and the theoretical underpinning for these practices.

2.1 Peer Instruction

Peer Instruction [7] is an interactive pedagogical approach where students prepare for lecture by reading relevant material, then attend lecture prepared to discuss with peers and the instructor [56]. In class, the instructor poses several multiple choice questions; for each, students individually think about the question, discuss with peers for several minutes, and vote on the correct answer.

Peer Instruction has been studied extensively in computing, finding that Peer Instruction: is valued by students in lower- and upper-division courses at both large research-focused universities and small liberal arts colleges [21, 31, 32]; shifts students classroom engagement from passive to interactive [40]; results in in-class learning, both from peers [34] and from the instructor [57]; reduces failure rates [8, 33]; results in improved final exam scores [42, 55]; and provides data useful to researchers for identifying key concepts and struggling students [23–25, 37].

Recently, Porter and Simon described the factors that led to Peer Instruction being adopted by portions of the computing education community [36]. However, some barriers continue to exist for some faculty wishing to adopt Peer Instruction in their courses [4]. For more detailed summaries of Peer Instruction, please see Simon et al. [41] and Porter and Simon [36].

Peer Instruction is strongly grounded in constructivist learning theory, specifically socioconstructive learning [48]. Peer Instruction replaces lecture from "sage on the stage" with instructor as "guide on the side." Through carefully crafted multiple-choice questions (targeting students' zone of proximal development), Peer Instruction provides students with a scaffolded opportunity to challenge themselves with new concepts and address common misconceptions. The peer discussion phase allows them to develop their own understanding through discussion and explanations to each other. Under the "Interactive, Constructive, Active, and Passive" Framework proposed by Chi and Wylie [6], they connect student behaviors during active learning to cognitive engagement. A study in CS found that students in traditional lectures report lower levels of engagement than those in a Peer Instruction class [40]. While we know of no research that explicitly measures the impact of Peer Instruction on specific social-psychological measures such as growth mindset and social belonging, there seem clear connections. The vote-discuss-vote process of learning in class demonstrates that we can all grow our understanding [3]. Involving students in discussion, becoming acquainted with several other students in the class, and observing other students' thinking could impact students' sense of self-belonging [52]. Peer Instruction also offers a form of cognitive apprenticeship in learning to analyze programs. By bringing analysis and discussion of code, which is often hidden behind the focus on code writing in introductory programming classes, Peer Instruction supports the "enculturation of students into authentic practices through activity and social interaction" [5].

2.2 Media Computation

Media Computation is a contextualized computing curriculum developed to teach programming to a broader range of students than solely those focused on becoming computing professionals [11, 15, 38]. The curriculum teaches standard introductory programming concepts in the context of manipulating images and sounds [11, 14].

Adoption of media computation has been connected to reduced failure rates [12, 35] as well as improved retention of students in computing [35, 43, 45]. A goal of the media computation curriculum was to improve outcomes for women in computing [12] and researchers found that women find the course more motivating than prior offerings [9], appreciate the media context [38], and may appreciate the opportunities for creative expression [2]. Passing rates were found to be balanced between men and women in the media computation version of the course [12].

Media Computation as a context for introducing computer programming is also grounded in the theory of situated cognition [20]. It recognizes that (as of the early 2000's) computers had, in general society, moved from being considered a computational tool to being a generally used communication tool. With Media Computation, students experience programming as "situated in activity bound to social, cultural and physical contexts" [10]. Additionally, as described in Forte and Guzdial [9], media computation-based assignments allow for much more creativity than traditional computational-focused assignments. These may improve learning as "the constructionist approach to learning asserts that people learn particularly well when they are engaged in constructing a public artifact that is personally meaningful" [9]. For extended summaries of the work on media computation, please see Guzdial [12, 13].

2.3 Pair Programming

Pair Programming is a cooperative learning approach that involves students working closely together while programming by alternating roles as "driver" (the person controlling the keyboard and mouse) and "navigator" (the person providing guidance and suggestions) [51]. Pair programming has been extensively studied in computing with three meta reviews appearing between 2011 and

2019 [17, 39, 47]. These meta-reviews find that pair programming is broadly associated with improved outcomes in computer science, including increased enjoyment and satisfaction [17] as well as improved grades on programming assignments, overall grades, and pass rates [47]. Pair programming has also been shown to improve retention of women in computing [26].

Not all pairings of students are equally effective as students with similar programming skills appear to be more successful [17]. Moreover, a recent qualitative study explored the components of inequitable pairings [22]. A longstanding concern with pair programming is that some students may not contribute enough to the pair to learn effectively [39, 54], however findings remain mixed.

Pair programming, and possible benefits from its use, can be explored through the lens of several theories of learning also seen in the two previous best practices. Pair programming is not just a constructivist learning approach but specifically a socioconstructivist one [48]. Pair programming, if explained to students as an industry-practice, can also contribute to students' sense of social belonging, although it is not really a form of legitimate peripheral participation as their community of practice only contains other novices [20]. Please see Simon et al. [41] for a detailed summary of Pair Programming.

3 COURSE CONTEXT

The quantity and quality of empirical computer science education research in 2008 was significantly less than it is today—even for research on introductory programming. In 2008, the redesign was centered around practices with quantified impacts on learning and/or outcomes with a focus on improving student retention, particularly for women and underrepresented groups.

The easiest choice was in adopting pair programming. The lead faculty member on the redesign was well aware of the research on pair programming and was heavily swayed by the 2004–2006 studies by Werner et al. [50], Werner [49], and McDowell et al. [26] on the value of pair programming for improving student retention and confidence, with a particular benefit for retention of women in computing. Additionally, because these studies had been performed by an institution with comparably large class sizes, we felt we could successfully implement similar processes.

In our CS1 course, we implemented pair programming in both a closed-lab setting (with TAs present) and for programming assignments completed outside class. To emphasize the importance of both students understanding what they produced, an additional requirement was added to each assignment for each student to complete a brief in-person comprehension check with the course staff (2 simple questions worth 10% of the assignment grade). In practice, these checks allowed the course staff to encourage students to participate more when pair programming and as an excuse for students to ask their partner to engage more.

Similarly, the choice to adopt Peer Instruction was bolstered by a relatively large pool of research—most in large, research-intensive university settings similar to our university. The difference for Peer Instruction was that research existed outside of the discipline of computer science, with many studies in introductory physics and other STEM disciplines [7, 16, 19, 44]. In this case, the faculty member leading the redesign (Author Simon) had extensive experience working with science courses that had adopted Peer Instruction during her sabbatical with the Carl Wieman Science Education Initiative at the University of British Columbia. By engaging with instructors using Peer Instruction in their classes, she gained a deep understanding of how to create quality Peer Instruction questions along with successful tips for integrating Peer Instruction into a course (how to motivate it, how to give students credit, etc.).

The media computation approach for introducing CS1 Java programming concepts and skills was relatively lacking in formal documentation of success at the time of adoption. The 2003 article by Guzdial [11] described a semester long implementation at Georgia Institute of Technology and that the targeted audience of the course was non-computing and engineering majors required to take a computing course. This focus on a non-traditional "computer science major" was in alignment with the goals of our redesign. The fact that the course in that study was taken as a requirement rather than elective made the limited assessment at the time more impressive than it would have been otherwise: in a 120 student course, 2/3 of whom were female, only 2 students dropped the course. Student comments, such as "very applicable to everyday life" and "programming is fun and ANYONE can do it", were quite appealing. Lastly, the addition of a major at the intersection of visual arts and computing at UC San Diego, and the likelihood these students would take the revised course, made the approach a good fit for the course.

4 METHODS

Our study focuses on the following research questions:

- **RQ1**—How do student outcomes compare between the version of the course before and after the redesign to include best practices?
- **RQ2**—Can changes in student outcomes be attributed to possible changes over time?
- **RQ3**—Which groups of students benefited from the introduction of these best practices?

The first question is used to gain an overview of how the introduction of best practices changed student performance and retention in the major. Our second research question was motivated by the possibility that any significant difference in performance and retention could be explained away by changes over time. The third question addresses how the course revision impacts students belonging to underrepresented groups in computing. Our analysis focuses on the two introductory programming courses which are the starts of two separate paths into the CS major at our institution: **CS1-NPE (No Prior Experience)** and **CS1-PE (Prior Experience)**, both of which act as an introduction to fundamental topics and techniques of programming. CS1-NPE is the first in a two-term course that serves as this introduction. The course is designed for students with little to no programming experience and is the course that experienced the redesign to include the three best practices. CS1-PE is designed for students with prior programming experience and leverages that prior experience to teach the same learning goals in a single term, rather than two terms.

We focus on two time periods: academic years 2001–2007 and 2008–2012. Academic years will be referred to by the year in which they begin, as all academic years span across two different calendar years. The first-time period, 2001–2007 (Fall 2001–Spring 2008), refers to the time period before best practices were implemented

Table 1: Metrics evaluated in this study

Metric	Description
Fail Rate	The percentage of students who received a D, F, or Withdrawal among students enrolled in the course.
Retained	The percentage of students who graduated with a degree in computing after entering CS1-NPE or CS1-PE already majoring in computing.
Switched	The percentage of students who graduated with a degree in computing after entering CS1-NPE or CS1-PE majoring in another discipline than computing (or as undeclared).
Upper-Division GPA	Average grades received (GPA), including counting withdrawals as failures, for students who attempted at least 5 upper-division computer science courses required for the major.
Time-to-Degree	Number of years from starting at our institution until graduation for students who receive a bachelor's in a computing major.

in CS1-NPE. The second time period, 2008–2012 (Fall 2008–Spring 2013), refers to the time period when best practices were implemented in CS1-NPE. CS1-PE had no significant change in course delivery over the time period between 2001—2012 and, as such, is useful for comparison. We gathered student data from those who enrolled in CS1-NPE and CS1-PE across the twelve-year time period (including their grades and graduation outcomes after the end of that period).

The analysis ends with the Spring of 2013 as our institution began restricting students' ability to major in computing at that time. The restrictions were significant, creating a highly competitive environment for those hoping to major and still turning away many students who were interested in the major. As many of the metrics we evaluate would likely be impacted by this change, particularly for underrepresented students based on recent work by Nguyen and Lewis [30], we end our analysis at the point those changes were made. In addition, we need to allow students time to graduate after they take the CS1 course for a number of our metrics to be accurate (e.g., retention and time-to-degree).

The primary metrics used in our evaluation appear in Table 1. Students included in the analysis are those who took CS1-NPE or CS1-PE during the regular school year as the courses are infrequently taught in the summer. Summer terms are accelerated and are typically taught by outside instructors. In addition, students needed to have earned a letter grade or withdrew from the CS1-NPE or CS1-PE course. Students who enrolled initially but dropped before the 4th week drop deadline were not present in the data provided and are hence not part of our analysis.

The data was provided from UC San Diego's educational services office in accordance with Human Subjects approval. The characteristics of the courses and students appear in Table 2. One special challenge in the data was how to handle students who failed CS1-PE or CS1-NPE but then later retook either CS1-PE or CS1-NPE. For Fail Rate, the student is counted in each course they received a grade. But all other metrics are tracked based on the final attempt outcomes of each student. As such, we grouped students based on their final attempt of a course. For example, if a student took CS1-NPE and received an "F" in 2007 and then took CS1-PE in 2009 and received a "C", they are considered a CS1-PE student in the 2008-2012 time period for metrics other than Fail Rate.

Once the student data was filtered, we ensured that the data matched with results in the prior evaluation of this course [35].

Table 2: Overall group breakdown of the classes. Unique students are students who took their *last* attempt in the given time period. Percentages are based on unique students.

	CS1-NPE		CS1-PE	
	2001–2007	2008–2012	2001–2007	2008–2012
# Enrolled	1732	2046	1513	1265
# Unique	1510	1868	1365	1179
% Female	25.4%	31.6%	15.4%	18.9%
% URM	10.5%	14.5%	5.3%	9.0%
% Comp Major	25.6%	41.7%	39.2%	53.7%

Specifically the Fail Rates in CS1-NPE and CS1-PE reported in the previous study matched the rates in our data.[2]

4.1 Data Analysis

Throughout our analysis, we note that different subsets of students are considered for different calculations, dependent on a pre-selected criteria. For example, a student who took CS1-NPE but earned a non-CS degree would not be considered in our calculation for Time-to-Degree. Rather than reporting the number of students for each metric throughout, we provide these numbers in Tables 3 and 4.

To answer our research questions, we use a combination of descriptive statistics, statistical tests for significance with significance set at $p = 0.05$, and effect sizes. To determine whether particular factors impacted outcomes for students when compared against other factors, we used a likelihood ratio test on regression models [29]. For clarity, more details regarding the particular tests performed are included along with the results.

5 RESULTS

5.1 RQ1: All Students

Tables 5 and 6 summarize the average metrics across both time periods for both courses. The overall average for each metric is followed by the significance and effect size test used to analyze the results. For binary metrics, such as Fail Rates, Retained, and Switched rates, we used a Z-test to test for significance. Relative risk

[2]Numbers matched or were within 1% of those previously reported. Conversations with the office providing the data explained that different databases handle some students differently (e.g., students who withdrew for medical leave, etc.).

Table 3: CS1-NPE Sample Sizes

	All		Male		Female		Non-URM		URM	
	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012
# Enrollment	1732	2046	1289	1398	443	647	1517	1749	215	297
# Unique	1510	1868	1126	1276	384	591	1351	1597	159	271
# CS Majors Entered	739	1037	599	807	140	229	642	875	97	162
# Non-CS Majors Entered	771	831	527	469	244	362	709	722	62	109
# >= 5 Upper Div Courses	405	763	321	618	84	174	366	699	39	94
# CS Graduates	387	779	303	600	84	178	355	695	32	84

Table 4: CS1-PE Sample Sizes

	All		Male		Female		Non-URM		URM	
	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012
# Enrollment	1513	1265	1261	1023	252	241	1431	1140	82	125
# Unique	1365	1179	1155	955	210	224	1292	1073	73	106
# CS Majors Entered	669	730	588	615	81	115	627	664	42	66
# Non-CS Majors Entered	696	449	567	340	129	109	665	409	31	40
# >= 5 Upper Div Courses	567	664	494	562	73	102	540	613	27	51
# CS Graduates	535	633	465	532	70	101	512	590	23	43

Table 5: An overall summary of average values for binary metrics, including significance test (* for p<0.05) and effect size.

	CS1-NPE				CS1-PE			
	2001–2007	2008–2012	p (Z-test)	Relative Risk	2001–2007	2008–2012	p (Z-test)	Relative Risk
Fail Rate	0.236	0.099	**2.909e-31***	0.418	0.200	0.179	0.161	0.895
Retained	0.471	0.624	**9.482e-11***	0.711	0.714	0.767	**0.024***	0.816
Switched	0.051	0.159	**1.101e-12***	0.886	0.083	0.163	**3.469e-05***	0.914

Table 6: An overall summary of average values for continuous metrics, including significance test (* for p<0.05) and effect size.

	CS1-NPE				CS1-PE			
	2001–2007	2008–2012	p (T-test)	Cohen's D	2001–2007	2008–2012	p (T-test)	Cohen's D
Upper-Division GPA	2.708	2.842	**0.002***	0.187	2.875	2.921	0.290	0.061
Time-to-Degree	4.490	4.490	0.937	-0.002	4.070	4.196	**0.042***	0.063

was used in order to analyze the effect size of the intervention in our data [1]. Relative risk conveys the risk of a negative outcome (failure, leaving the major, not joining the major) after an intervention relative to the risk before. Values less than 1 express a reduction in risk (e.g., the relative risk for Fail Rates for CS1-NPE of 0.418 means the risk of failing after the best practices intervention is 41.8% of the average risk before the intervention).

For continuous metrics, such as Upper-Division GPA and Time-to-Degree, we used a T-test for significance and Cohen's D for effect size. Since standard Cohen's D uses high variance in the data to calculate an accurate effect size, standard Cohen's D does not work well with GPA data. Hence, we used a variation of the Cohen's D appropriate for GPA analysis [53].

Looking at students in CS1-NPE in Tables 5 and 6 we see substantial positive impacts occurring in Fail Rates, Retention rates in the major, Switch rates into the major, and upper-division course performance. The reduced failure rates for CS1-NPE also led to fewer students retaking CS1 (from 8.6% of all students in the course failing and retaking it to 4.1%) whereas CS1-PE saw an rise in students retaking CS1 between the time periods (from 11% to 14.8%).

The relative risk indicates that students who took CS1-NPE with the intervention were at around 42% of the total average risk of failing the course, 71% of the total average risk of not being retained in the major, and at around 89% of the total average risk of not switching into the major. The positive effect, per Cohen's D, on Upper-Division GPA is small (where a small effect size is considered at 0.2) [53]. For all these metrics, students in the best practices

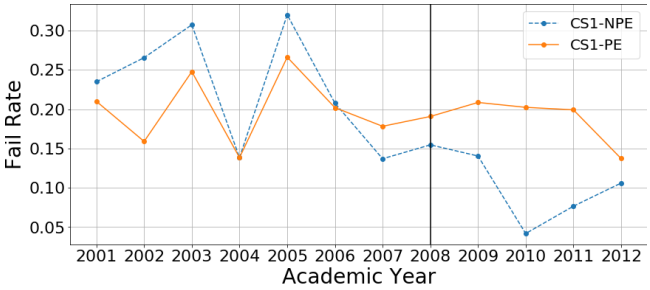


Figure 1: Average annual Fail Rates for CS1-NPE and CS1-PE.

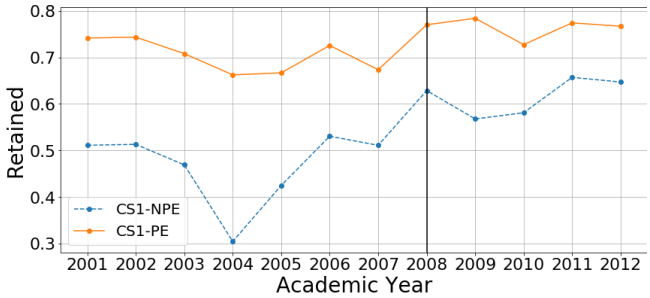


Figure 2: Average annual Retained rates for students who entered CS1-NPE and CS1-PE as computer science majors and graduated with a computer science degree. Y-axis begins at 0.3 to help show differences over time.

time period saw statistically significantly improved outcomes with varying effect sizes. There was no perceivable impact to Time-to-Degree as the average is the same for both time periods. We note that a myriad of factors impact Time-to-Degree but this is consistent with the other findings that there is no evidence students were harmed by the best-practices course.

5.2 RQ2: Effect of Time Periods

5.2.1 Comparing CS1-NPE and CS1-PE statistics. Recall that there was no intervention implemented in CS1-PE during the time period of the intervention in CS1-NPE. Hence, our first step in answering RQ2 is to explore how students performed when taking CS1-PE in each time period. Tables 5 and 6 provide the results for CS1-PE. Here we see that CS1-PE also experienced statistically significant improvements to Retained and Switched. Unfortunately, it also saw a statistically significant increase in Time-to-Degree. For Fail Rates and Upper-Division GPA, it appears CS1-NPE uniquely benefited in a statistically significant way during the best practices time period. For Retained and Switched, both courses benefited which might be expected given the increased interest in computing nationally during the same time period. Examining the effect size for Retained and Switched, we see the impact on CS1-NPE appears larger than for CS1-PE as the relative risk is higher for CS1-PE. We hesitate to draw too large a conclusion from this, however, as CS1-PE started with better values for each metric. Lastly, although Time-to-Degree remained roughly constant between time periods for CS1-NPE, it worsened significantly for CS1-PE.

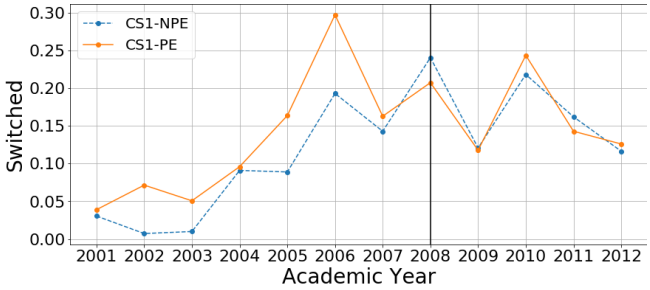


Figure 3: Average annual Switched rates for students who entered CS1-NPE and CS1-PE as non-computer science majors and graduated with a computer science degree.

5.2.2 Outcomes per Year. One possible reason for the improvements for the two averages between time periods could simply be a steady improvement over time (e.g., a steady positive slope from 2001–2012 would result in lower averages in 2001-2007 than 2008–2012). To explore this possible explanation, Figures 1–5 contain the average for each metric over each academic year. The black vertical bar indicates the year when best practices were introduced into CS1-NPE.

The Fail Rates in Figure 1 varied year to year with no general trend (although there might have been a negative trend for CS1-NPE starting in 2005). The decline in Fail Rates for CS1-NPE stands out relative to CS1-PE. Retained rates for CS1-PE in Figure 2 remain relatively constant with a slight increase in the later time period whereas CS1-NPE sees a marked increase over time (2004 is a particularly poor year for CS1-NPE). CS1-PE and CS1-NPE appear rather comparable for the rate of students switching into the major over time, in Figure 3, with perhaps CS1-NPE catching up to CS1-PE during the best practices time period.

Upper-Division GPA over time, found in Figure 4, shows that students in CS1-NPE during the earlier time frame under-performed in later courses relative to those who took CS1-PE. Encouragingly, after the change to best practices, CS1-NPE appears to close that gap. Figure 5 shows that Time-to-Degree over time remains mostly consistent. The fact that Time-to-Degree is longer for students in CS1-NPE is expected as CS1-NPE is the first of a two course sequence relative to a single CS1-PE course. Both courses feed into a long dependency chain of required courses for computing majors that might explain the additional quarter (approximately).

5.2.3 Regression Modelling for Time. Another approach to examining the impact of time on the results is to use regression modeling on CS1-NPE. Specifically, the goal is to build a regression model (logistical for binary outcomes, linear for continuous outcomes) to predict student outcomes. The first model is given years as dependent variables (along with an offset) to predict the particular outcome. The second model is provided with the dependent variable of best practices (a binary variable false for 2001–2007 and true for 2008–2012) on top of the year and offset variables. The question becomes whether the model's accuracy improves by adding the best practices variable. If the model's accuracy improves significantly, this means more of the variance in the outcomes can be explained if the model knows about the course revision than without. This

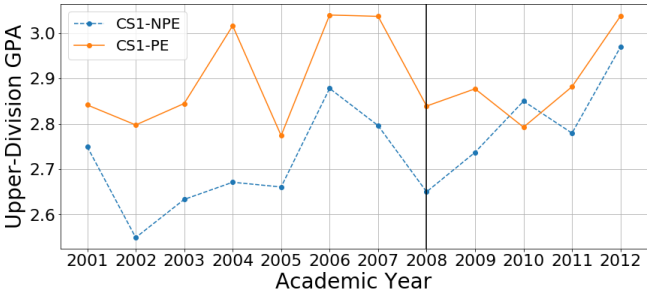


Figure 4: Average annual Upper-Division GPA for students who had at least five upper division course attempts. Y-axis begins at a GPA of 2.5 to help show the differences over time.

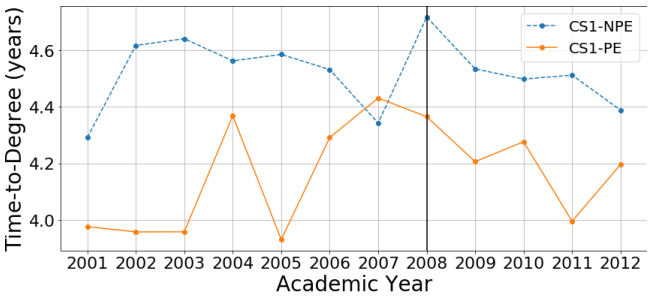


Figure 5: Average annual Time-to-Degree for computer science graduates. Y-axis begins at 3.9 years as expected time-to-degree is 4 years at UC San Diego.

Table 7: Results of likelihood ratio test and corresponding chi-squared p-values between regression models. For Fail Rates, Upper-Division GPA, and Retained, a best-practices feature significantly improves the model beyond the model that includes only time in years.

Metric	Likelihood Ratio	p
Fail Rates	11.021	**0.0009***
Retained	5.089	**0.024***
Switched	1.529	0.216
Upper-Division GPA	4.000	**0.046***
Time-to-Degree	2.800	0.094

would provide evidence that the transition to best practices changed the outcomes for CS1-NPE students beyond changes over time.

To accomplish this, we use the likelihood ratio test [29]. Specifically, we observe the likelihood ratio between models for each metric and their corresponding chi-squared p-value, with degree of freedom 1, to see if the addition of the intervention feature is significant in predicting student outcomes. The results of these tests appear in Table 7. For Fail Rates, Retained, and Upper-Division GPA, including the best-practices feature better informs the model than time alone. This provides further evidence that the course-redesign impacted outcomes beyond what one might expect from just changes over time.

5.3 RQ3: Underrepresented Groups

Table 8 provides the outcomes for male and female students across both time periods in CS1-NPE. The addition of best practices into CS1-NPE appears to have a positive benefit for both male and female students. Recall that our data set is heavily skewed towards males in each year observed in the study. Similar to our overall results, Fail Rates, Retained and Switched rates, and Upper-Division GPA improved significantly for male and female students after best practices were enacted. Examining the scale of the benefits, women appear to benefit more for some metrics and men for others.

Table 9 provides the outcomes for CS1-PE where Switch rates are significant for both genders whereas Fail Rates, Upper-Division GPA, and Time-to-Degree are significant only for women. It appears that much of the benefits that we saw overall for CS1-PE were due to large improvements for women over this time period. It is unclear what may have changed between these time periods to benefit female students in CS1-PE and remains a topic for future analysis.

Similar benefits can be seen when comparing non-URM versus URM students throughout both courses and time periods in Tables 10 and 11. However, it is important to acknowledge that the number of URM students that were considered in each metric analysis was small compared to non-URM students. The first finding that stands out is simply than URM students struggle at our institution relative to non-URM students in CS1-NPE and CS1-PE. Fail Rate is particularly striking as the failure rate for URMs for CS1-NPE was nearly twice that of CS1-PE (44.7% versus 24.4%) before the transition to best practices. After the transition, CS1-NPE Fail Rate for URM dropped remarkably from 44.7% to 17.5% between time frames whereas CS1-PE saw an increase for URM students from 24.4% to 28.0%. Also striking is that the percentage of URM students switching into the major for CS1-NPE and CS1-PE was 0 during the 2001-2007 time period. The numbers were small (between 2001–2007, only 62 URM students took CS1-NPE as non-majors and only 31 URM students took CS1-PE), but the fact none switched remains jarring. In the best practices time period, the Switch rate for CS1-PE increased to 7.5% while CS1-NPE rose to 14.7%.

Overall, for CS1-NPE, both URM and non-URM students benefited statistically significantly for three of the five metrics. In addition, for CS1-NPE, effect sizes for improvements for URMs were comparable to non-URMs except for Upper-Division GPA where URMs experienced a considerably larger improvement than non-URM. In contrast, for CS1-PE, none of the metrics are statistically significant for URM students. This is likely due to a combination of lower effect sizes for all students and lower numbers of URM students in the course (relative to CS1-NPE).

6 DISCUSSION

6.1 Better Instructors?

One concern for our analysis was that there might have just been an improvement in the quality of teachers in the later time period. Perhaps more dedicated teachers were willing to teach the best practices version of the course than those who taught the course previously. We examined this in two separate ways. First, we looked at four instructors who had taught both before and after the course redesign. Because sample sizes reduced when looking at individual instructors who might have taught only a single term during one of

Table 8: CS1-NPE Results by Gender

	Male				Female			
	2001–2007	2008–2012	p (Z-test)	Relative Risk	2001–2007	2008–2012	p (Z-test)	Relative Risk
Fail Rate	0.231	0.094	**4.514-23***	0.405	0.251	0.110	**4.799e-10***	0.438
Retained	0.471	0.634	**5.660e-10***	0.691	0.471	0.585	**0.033***	0.785
Switched	0.040	0.188	**1.763e-14***	0.846	0.074	0.122	0.057	0.948
	2001–2007	2008–2012	p (T-test)	Cohen's D	2001–2007	2008–2012	p (T-test)	Cohen's D
Upper-Division GPA	2.736	2.849	**0.023***	0.156	2.599	2.824	**0.016***	0.323
Time-to-Degree	4.489	4.505	0.786	0.008	4.509	4.433	0.477	-0.038

Table 9: CS1-PE Results by Gender

	Male				Female			
	2001–2007	2008–2012	p (Z-test)	Relative Risk	2001–2007	2008–2012	p (Z-test)	Relative Risk
Fail Rate	0.177	0.172	0.764	0.973	0.313	0.203	**0.005***	0.649
Retained	0.707	0.771	**0.012***	0.784	0.765	0.748	0.779	1.075
Switched	0.088	0.171	**0.0002***	0.910	0.062	0.138	**0.048***	0.919
	2001–2007	2008–2012	p (T-test)	Cohen's D	2001–2007	2008–2012	p (T-test)	Cohen's D
Upper-Division GPA	2.894	2.905	0.815	0.014	2.748	3.011	**0.018***	0.366
Time-to-Degree	4.126	4.177	0.464	0.025	3.693	4.297	**1.275e-05***	0.302

Table 10: CS1-NPE Results for non-URM and URM Students

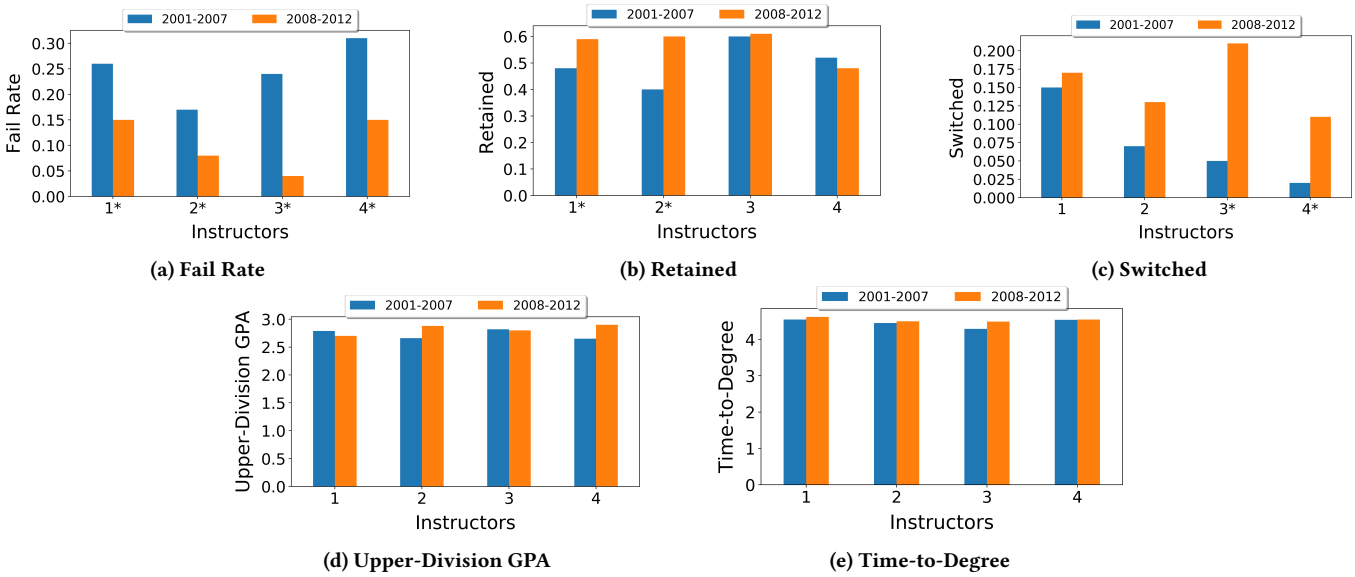
	Non-URM				URM			
	2001–2007	2008–2012	p (Z-test)	Relative Risk	2001–2007	2008–2012	p (Z-test)	Relative Risk
Fail Rate	0.206	0.086	**1.548e-23***	0.416	0.447	0.175	**2.839e-12***	0.392
Retained	0.492	0.662	**1.748e-11***	0.666	0.330	0.420	0.150	0.866
Switched	0.055	0.161	**7.005e-11***	0.888	0	0.147	**0.001***	0.853
	2001–2007	2008–2012	p (T-test)	Cohen's D	2001–2007	2008–2012	p (T-test)	Cohen's D
Upper-Division GPA	2.754	2.877	**0.007***	0.175	2.273	2.581	**0.033***	0.409
Time-to-Degree	4.461	4.453	0.884	-0.004	4.859	4.792	0.744	-0.034

Table 11: CS1-PE Results for non-URM and URM Students

	Non-URM				URM			
	2001–2007	2008–2012	p (Z-test)	Relative Risk	2001–2007	2008–2012	p (Z-test)	Relative Risk
Fail Rate	0.197	0.168	0.055	0.850	0.244	0.280	0.567	1.148
Retained	0.726	0.783	**0.016***	0.791	0.548	0.606	0.551	0.871
Switched	0.087	0.171	**3.303e-05***	0.908	0	0.075	0.118	0.925
	2001–2007	2008–2012	p (T-test)	Cohen's D	2001–2007	2008–2012	p (T-test)	Cohen's D
Upper-Division GPA	2.894	2.938	0.318	0.059	2.503	2.720	0.307	0.248
Time-to-Degree	4.061	4.190	**0.044***	0.064	4.261	4.279	0.947	0.009

the time periods, we expected few findings to be statistically significant. However, we still found statistically significant reductions for Fail Rates, Retained, and Switched. Figure 6 provides the results per instructor and sample sizes for each instructor appears in Table 12. From Figure 6, we see that student outcomes improved for each instructor during the best practices time period for all significant differences in performance. Overall, it appears the transition to best practices by these instructors corresponded to improved student outcomes (Fail Rates, Retained, and Switched).

Second, we also used a likelihood ratio test to examine whether our linear or logistic model that includes offset, years, and instructor parameters would improve if given best practices improved the



(a) Fail Rate (b) Retained (c) Switched

(d) Upper-Division GPA (e) Time-to-Degree

Figure 6: Averages for each metric for each instructor who taught CS1-NPE in both time periods. Instructor significance for a given metric between time periods is indicated with an asterisk next to the instructor's label.

Table 12: CS1-NPE sample sizes for instructors who taught during both time frames.

	Instructor 1		Instructor 2		Instructor 3		Instructor 4	
	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012	2001–2007	2008–2012
# Enrolled	333	499	453	168	129	502	612	172
# Unique	280	455	404	158	113	481	522	163
# CS Majors Entered	178	260	224	67	47	271	265	69
# Non-CS Majors Entered	102	195	180	91	66	210	257	94
# >= 5 Upper Div Courses	101	195	110	53	34	209	146	44
# CS Graduates	101	187	102	52	31	210	142	43

Table 13: Results of likelihood ratio test and corresponding chi-squared p-values comparing a model with time and instructor features against a model that also includes the best-practices feature. For Fail Rates and Retained, the best-practices feature significantly improves the model.

Metric	Likelihood Ratio	*p*
Fail Rate	62.926	**2.146e-15***
Retained	6.580	**0.010***
Switched	0.819	0.365
Upper-Division GPA	3.000	0.083
Time-to-Degree	2.000	0.157

models' accuracy. Results can be found in Table 13. For both Fail Rates and Retained, the best practices feature improves the model's performance significantly, suggesting it was the adoption of best practices that resulted in the improved outcomes for those metrics.

6.2 Implications of Findings

Challenges from Colleagues: A motivation for this study was faculty colleagues challenging research results on Peer Instruction, Media Computation, and Pair Programming. A common refrain was that although these approaches may have succeeded in lowering failure rates or briefly improving retention, it was really just bringing in poorly prepared students who were bound to struggle and leave the program later. We are quite pleased that, for this particular redesign, the larger body of students went on to succeed in the rest of the program at the same rate as before (or better).

Factors in Success: Given the positive outcomes related to the redesign, what factors led to its success? We suspect there were at least two main contributors. The first was the mandate to adopt evidence-based practices and the selection of three practices that, although they each seemed promising at the time, have each been shown to be broadly effective in the research since. The second was a broad commitment to the redesign from the faculty who drove the changes themselves, the graduate and undergraduate instructional staff who helped enact those changes, and the other faculty who adopted the course when they later taught it.

Underrepresented Groups: A recent meta-analysis of pedagogy in STEM found that active learning causes the achievement gap for underrepresented groups (relative to represented groups) to be narrowed [46]. We found that students from both groups benefited from the course redesign and hence found no consistent reduction in the achievement gap for our metrics. However, the raw difference in the benefit was larger in many cases because the outcomes for students from underrepresented groups (particularly URMs) were worse at the start. For example, the Fail Rate for represented students in CS1-NPE dropped from 20.6% to 8.6% after the addition of best practices while the Fail Rate for URM students in CS1-NPE dropped from 44.7% to 17.5%. As raw percentages, URM students could be seen as benefiting more from the change. However, as a ratio, URM students failed CS1-NPE 2.16 times more than represented groups before the change and 2.04 times after the change. Although the improved outcomes for URM students is encouraging, the resistant disparity in outcomes for URM students is a clear problem that urgently requires more research and critical analysis.
Longitudinal Studies: This project suffered from several challenges due to its longitudinal nature, including gaining approval to collect sensitive data and inconsistent data reporting within internal databases. Despite these challenges, we strongly encourage similar studies so that the community can improve understanding of the long-term impact of pedagogical changes.

6.3 Threats to Validity

Change in Time: Our comparisons of the CS1-PE and CS1-NPE courses, as well as regression modeling of course outcomes, both suggest that the redesign of the course explains the improvement in Fail Rates and Retained. However, it is impossible to know if there were changes in the perception of the computing field at the time of the redesign—particularly given the increases in enrollments in CS between 2008 and 2013 at UC San Diego and nationally [27]. In addition, changes elsewhere in the major may have impacted student outcomes. However, there were no significant systemic changes to our major over the examined time period.
Better Teachers: Although the improved Fail Rates and Retained for the four instructors who taught before and after the course redesign suggest that the improvement in the course are associated with the better outcomes found more broadly, there were still differences in instructors between time frames. For example, one instructor who had slightly higher Fail Rates in general taught the course more during the earlier time frame than the later. As such, it is difficult to know definitely how large an impact the variation in instructors had on the outcomes of this study.
Combining Best Practices: One challenge in interpreting these results is that by combining multiple best practices in one course, we cannot distinguish which were more important or if they were needed in combination. As such, we can only conclude the combination appears to have been successful.

6.4 Call to Action

Jobs in the computing industry are plentiful and pay well. Unfortunately, diversity in the computing industry continues to languish. We adopted practices in one single course that increased the number of successful students graduating with computing degrees. To be candid, the implementation of the course itself was significantly less challenging (and more fun and rewarding) than conducting this longitudinal analysis. Given the extensive evidence showing these best practices provide substantial benefit to our students, why are more institutions not replicating or implementing similar changes? We do not know for sure, but can posit several factors. First, we recognize faculty change can be hard [18]. Second, we suspect institutions feel the need to distinguish themselves. In particular, computer scientists may feel the need to build their own solutions as the computing field favors the "new" (albeit un- or less-tested) and denigrates anything "old" (where old is a handful years?). Are these self-centered factors worth the cost of keeping the discipline less accessible to students who could succeed?

We also recognize US society is biased and does not provide a level playing field for many subgroups, but especially people of color. The computing profession is infamous for its lack of diversity and for limited results in efforts to make change in this area. While URM students benefited substantially from the best practices we implemented, at the end they were still *2 times* more likely to fail the course than majority students. We as a community should find this deeply concerning. The time has come for CS instructors to adopt evidence-based instructional practices to improve outcomes for their underrepresented students and for the CER community to prioritize finding further solutions to help URM students.

7 CONCLUSION

In this longitudinal study that spans two decades, we examined the impact of redesigning a CS1 course for students without prior programming experience to include Peer Instruction, Media Computation, and Pair Programming. We find that over the five years after the redesign, students in the course had lower failure rates and computing majors had a higher chance of being retained through graduation. These improvements appear connected to the course revisions, even when examining changes in other courses (without changes) and examining changes over time. Moreover, instructors who taught before and after the revision saw similar improvements to student outcomes after the course revision.

Despite this larger group of students succeeding and progressing into later computer science courses, we find no evidence that their outcomes were worse than those of students from before the redesign. Although there are encouraging signs that students from underrepresented and represented groups benefited from the changes and that the magnitude of those benefits were higher for URM students, there remains a large gap between URM students and represented students that deserves further study. Given the multiple research studies documenting the success of each of these instructional practices and the evidence from this study that these improved outcomes are sustained over many years, we hope CS1 instructors will be further encouraged to adopt these evidence-based instructional practices for the betterment of their students.

ACKNOWLEDGMENTS

The authors thank Rita Keil, Laura Kertz, Carolyn Sandoval, and Ying Xiong for their assistance as well as the reviewers for their helpful feedback. This work was supported in part by NSF award 1712508 as well as a UCSD Sloan Scholar Fellowship, a UCSD STARS Fellowship, and a Gates Millennium Scholarship.

REFERENCES

[1] D. G. Altman. *Practical statistics for medical research*. CRC press, 1990.

[2] American Association of University Women. Educational Foundation. Commission on Technology and Gender and Teacher Education. *Tech-savvy: Educating girls in the new computer age*. American Association of University Women, 2000.

[3] L. S. Blackwell, K. H. Trzesniewski, and C. S. Dweck. Implicit theories of intelligence predict achievement across an adolescent transition: A longitudinal study and an intervention. *Child development*, 78(1):246–263, 2007.

[4] D. Bouvier, E. Lovellette, J. Matta, J. Bai, J. Chetty, S. Kurkovsky, and J. Wan. Factors affecting the adoption of peer instruction in computing courses. In *Proceedings of the Working Group Reports on Global Computing Education*, pages 1–25, 2019.

[5] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational researcher*, 18(1):32–42, 1989.

[6] M. T. Chi and R. Wylie. The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist*, 49(4):219–243, 2014.

[7] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69, 2001.

[8] P. Deshpande, C. B. Lee, and I. Ahmed. Evaluation of peer instruction for cybersecurity education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 720–725, 2019.

[9] A. Forte and M. Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. In *37th Annual Hawaii International Conference on System Sciences*, pages 10–pp, 2004.

[10] J. Greeno, D. Smith, and J. Moore. Transfer of situated learning. In D. Detterman and R. Sternberg, editors, *Transfer on trial: intelligence, cognition, and instructionh*, pages 99–167. 1993.

[11] M. Guzdial. A media computation course for non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 104–108, 2003.

[12] M. Guzdial. Exploring hypotheses about media computation. In *Proceedings of the 9th Annual ACM Conference on International Computing Education Research*, pages 19–26, 2013.

[13] M. Guzdial. Computing for other disciplines. In S. A. Fincher and A. V. Robins, editors, *The Cambridge Handbook of Computing Education Research*, chapter 19, pages 584–605. Cambridge University Press, 2019.

[14] M. Guzdial and B. Ericson. *Introduction to computing & programming in Java: a multimedia approach*. Pearson Prentice Hall, 2007.

[15] M. Guzdial and A. Forte. Design process for a non-majors computing course. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education*, pages 361–365, 2005.

[16] R. R. Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1):64–74, 1998.

[17] B. Hanks, S. Fitzgerald, R. McCauley, L. Murphy, and C. Zander. Pair programming in education: A literature review. *Computer Science Education*, 21(2):135–173, 2011.

[18] C. Henderson and M. H. Dancy. Barriers to the use of research-based instructional strategies: The influence of both individual and situational characteristics. *Physical Review Special Topics-Physics Education Research*, 3(2):020102, 2007.

[19] J. K. Knight and W. B. Wood. Teaching more by lecturing less. *Cell biology education*, 4(4):298–310, 2005.

[20] J. Lave and E. Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[21] C. B. Lee, S. Garcia, and L. Porter. Can peer instruction be effective in upper-division computer science courses? *Transactions on Computing Education*, 13(3), Aug. 2013.

[22] C. M. Lewis and N. Shah. How equity and inequity can emerge in pair programming. In *Proceedings of the 11th Annual ACM Conference on International Computing Education Research*, pages 41–50, 2015.

[23] S. N. Liao, D. Zingaro, C. Alvarado, W. G. Griswold, and L. Porter. Exploring the value of different data sources for predicting student performance in multiple cs courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 112–118, 2019.

[24] S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter. Lightweight, early identification of at-risk cs1 students. In *Proceedings of the 12th Annual ACM Conference on International Computing Education Research*, pages 123–131, 2016.

[25] S. N. Liao, D. Zingaro, K. Thai, C. Alvarado, W. G. Griswold, and L. Porter. A robust machine learning technique to predict low-performing students. *Transactions on Computing Education*, 19(3):1–19, 2019.

[26] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8):90–95, 2006.

[27] National Academies of Sciences, Engineering, and Medicine and others. *Assessing and responding to the growth of computer science undergraduate enrollments*. National Academies Press, 2018.

[28] National Center for Science and Engineering Statistics. Women, minorities, and persons with disabilities in science and engineering: Special report NSF 19-340. 2019.

[29] M. Natrella. NIST/SEMATECH e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook, 2010.

[30] A. Nguyen and C. M. Lewis. Competitive enrollment policies in computing departments negatively predict first-year students' sense of belonging, self-efficacy, and perception of department. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 685–691, 2020.

[31] L. Porter, D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro, and B. Simon. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 358–363, 2016.

[32] L. Porter, S. Garcia, J. Glick, A. Matusiewicz, and C. Taylor. Peer instruction in computer science at small liberal arts colleges. In *Proceedings of the 18th Annual Conference on Innovation and Technology in Computer Science Education*, 2013.

[33] L. Porter, C. B. Lee, and B. Simon. Halving fail rates using peer instruction: A study of four computer science courses. In *Proceedings of the 44th Special Interest Group on Computer Science Education Technical Symposium*, 2013.

[34] L. Porter, C. B. Lee, B. Simon, and D. Zingaro. Peer instruction: Do students really learn from peer discussion in computing? In *Proceedings of the 7th Annual ACM Conference on International Computing Education Research*, 2011.

[35] L. Porter and B. Simon. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceedings of the 44th Special Interest Group on Computer Science Education Technical Symposium*, 2013.

[36] L. Porter and B. Simon. A Case Study of Peer Instruction: From University of California, San Diego to the Computer Science Community. In S. A. Fincher and A. V. Robins, editors, *The Cambridge Handbook of Computing Education Research*, chapter 30, pages 861–874. Cambridge University Press, 2019.

[37] L. Porter, D. Zingaro, and R. Lister. Predicting student success using fine grain clicker data. In *Proceedings of the 10th Annual ACM Conference on International Computing Education Research*, pages 51–58, 2014.

[38] L. Rich, H. Perry, and M. Guzdial. A CS1 course designed to address interests of women. *ACM SIGCSE Bulletin*, 36(1):190–194, 2004.

[39] N. Salleh, E. Mendes, and J. Grundy. Empirical studies of pair programming for CS/SE teaching in higher education: a systematic literature review. *IEEE Transactions on Software Engineering*, 37(4):509–525, 2010.

[40] B. Simon, S. Esper, L. Porter, and Q. Cutts. Student experience in a student-centered peer instruction classroom. In *Proceedings of the 9th Annual ACM Conference on International Computing Education Research*, 2013.

[41] B. Simon, C. Hundhausen, C. McDowell, L. Werner, H. Hu, , and C. Kussmaul. Students as teachers and communicators. In S. A. Fincher and A. V. Robins, editors, *The Cambridge Handbook of Computing Education Research*, chapter 29, pages 827–857. Cambridge University Press, 2019.

[42] B. Simon, J. Parris, and J. Spacco. How we teach impacts learning: peer instruction vs. lecture in CS0. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 2013.

[43] R. H. Sloan and P. Troy. CS 0.5: a better approach to introductory computer science for majors. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, pages 271–275, 2008.

[44] M. K. Smith, W. B. Wood, W. K. Adams, C. Wieman, J. K. Knight, N. Guild, and T. T. Su. Why peer discussion improves student performance on in-class concept questions. *Science*, 323(5910):122–124, 2009.

[45] A. E. Tew, C. Fowler, and M. Guzdial. Tracking an innovation in introductory cs education from a research university to a two-year college. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education*, pages 416–420, 2005.

[46] E. J. Theobald, M. J. Hill, E. Tran, S. Agrawal, E. N. Arroyo, S. Behling, N. Chambwe, D. L. Cintrón, J. D. Cooper, G. Dunster, et al. Active learning narrows achievement gaps for underrepresented students in undergraduate science, technology, engineering, and math. *Proceedings of the National Academy of Sciences*, 117(12):6476–6483, 2020.

[47] K. Umapathy and A. D. Ritzhaupt. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education*, 17(4):1–13, 2017.

[48] L. S. Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, 1980.

[49] L. L. Werner. Female computer science students who pair program persist. In *ACM Journal of Educational Resources in Computing*. Citeseer, 2004.

[50] L. L. Werner, B. Hanks, and C. McDowell. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing*, 4(1):4–es, 2004.

[51] L. Williams and R. Kessler. *Pair programming illuminated*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[52] T. D. Wilson, M. Damiani, and N. Shelton. Improving the academic performance of college students with brief attributional interventions. In J. Aronson, editor, *Improving Academic Achievement: Impact of Psychological Factors on Education*, chapter 5, pages 91–110. Academic Press, 2002.

[53] K. Wuensch. Standardized effect size esitmation: Why and how? http://core.ecu.edu/psyc/wuenschk/StatHelp/Effect%20Size%20Estimation.pdf, 2015. Accessed: 2020-04-02.

[54] H. Yuan and Y. Cao. Hybrid pair programming-a promising alternative to standard pair programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1046–1052, 2019.

[55] D. Zingaro. Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 373–378, 2014.

[56] D. Zingaro, C. Bailey Lee, and L. Porter. Peer instruction in computing: the role of reading quizzes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 47–52, 2013.

[57] D. Zingaro and L. Porter. Peer instruction in computing: The value of instructor intervention. *Computers and Education*, 71, 2014.