THE EFFECT OF CHANGING PROGRAMMING LANGUAGE TO STUDENT SUCCESS IN UNDERGRADUATE COMPUTER SCIENCE CURRICULUM

Ebru Celikel Cankaya
University of Texas at Dallas, Richardson, TX, USA
{ebru.cankaya@utdallas.edu}

Abstract- . We measure and compare the effects of changing the programming language in introductory level courses (Computer Science I and II) in undergraduate Computer Science (CS) curriculum. The study presents the example of University of Texas at Dallas (UTD) case, where we present the results of two approaches: Approach 1 where both CS I and CS II courses are taught in Java, and Aprroach 2 where CS I is taught in C++, and CS II in Java. The paper presents the data collected from four consecutive semesters and analyzes and compares the effect of using different programming languages on student success, as well as drop rates. The results show that changing the programming language in consecutive sequences of introductory level undergraduate courses in CS curriculum do not adversely affect the student performance, and in some presented cases it even helps students score better. Also, we find that this change causes once an improvement, and another time a deterioration in the drop rate, which suggests the requirement for more data to be collected in coming semesters.

Key Words- Programming language, CS I, CS II, C++, Java, undergraduate curriculum.

1. INTRODUCTION

As any education in a specific field of science would, Computer Science education also takes enthusiasm, dedication, an ability to have analytical thinking skills and to use them properly. Therefore, it gets utterly important to attract and retain students to this demanding major in colleges and universities. There are factors that obstruct us from achieving this goal: Ironically, we need to employ proven to have worked methods as otherwise will be too costly. In the meantime, we need to be open to trying new methods so as to see if they work for us. Moreover, we need to repeat trying a new method several semesters to observe consistency and to determine if the result (either appearing to be an improvement or decline). There is a tradeoff behind all these as we can hardly recover from any losses that took place.

This study focuses on the effect of changing the programming language in introductory level courses at undergraduate Computer Science curriculum to the success of students, and therefore to the retention rate. It presents the results of a 6 semesters long study obtained at introductory level programming courses at undergraduate level at University of Texas at Dallas (UTD) Department of Computer Science. Specifically, we measure the effect of teaching CS I and CSII courses as once both in Java, then as CS I in C++ and CS II in Java. We compare the outcomes based on the students' success rates per assessment and see that changing the programming language has no significant effect on the performance of students that is measured in numeric grade. Furthermore, the results also show that the change in the programming language had no significant effect on the overall retention rate for these students.

The rest of this paper is organized as follows: In section 2, we present similar efforts that are adopted in higher education institutes nationwide and even internationally that aim at improving Computer Science education in undergraduate level as well as keeping the retention rates high. Section 3 shows in detail the outcomes we obtained from the data gathered from totally four consecutive semesters at UTD Computer Science Department. Section 4 concludes the work and presents some insight on future work to expand the current study.

2. BACKGROUND AND MOTIVATION

The effect of changing the programming language in introductory level courses at undergraduate Computer Science curriculum has direct consequences as changes in student success, and the retention rate. To improve retention among CS students, several methods have been introduced and are being applied. As mentioned in literature: Some of these efforts focus on a particular group such as minorities, women (Haines, 2013), needy students, etc. Among these methods lie such activities as organizing computer related workshops for prospective students at high school - or even at middle school - level so that they can play, program, and enjoy using computers and hopefully develop interest towards computer science (Robinson, Pérez-Quiñones, 2014, Yardi 2007), or taking freshmen CS major students to targeted conferences, such as the Grace Hopper Celebration of Women in Computing (Alvarado, Judson, 2014), or providing extra help to students via mentoring programs (Brown, Yuan, 2014), incorporating a peer led team learning (PLTL) scheme that involves participating students into small groups (4-8 students) to meet regularly for additional problem solving sessions in the lead of a peer leader (Horwitz et al., 2009). Other scholar work focuses on developing social bonds among CS majors to nourish and reinforce community identity (Crenshaw et al., 2008). Yet another approach is to change the curriculum, such as incorporating a theme based approach and supporting hands-on labs (Barr, 2012).

Some of these methods are general enough to get better retention rates in almost any major in Universities. There are also more specific actions we can take such as choosing a programming language that will engage and keep student interest in programming. Newhall et al. 2014 presents the successful results they have obtained by changing the programming language from several years in C, then one year in Java, and afterwards adopting Python in Swarthmore College. Similarly, we present the effect of programming language on student success, though we specifically analyze changing programming language in the two introductory level CS courses: CS I and CS II at UTD.

All of the above mentioned methods have been applied successfully and yielded promising results in improving student retention rates in undergraduate, as well as graduate levels. In this work, in addition to the eventual goal of improving retention rates, we also focus purely on student success: We investigate if exposing students to two different programming languages, namely C++ and Java, at introductory level programming sequence undergraduate courses CS I and CS II, or covering both courses (CS I and CS II) in one single programming language (in Java) yields better performance results. For this, we use student score as a measure of performance.

3. EXPERIMENTAL WORK AND RESULTS

We base our experimental results on the data collected on four consecutive semesters at two undergraduate introductory level CS major courses as CS I and CS II at University of Texas at Dallas (UTD) Department of Computer Science.

Table 1: The programming languages used for each CS I and CS II courses during four consecutive semesters.

	Preceding CS I	CS II	Method
Fall 2012	Java	Java	Approach 1
Spring 2013	C++	Java	Approach 2
Fall 2013	C++	Java	Approach 2
Spring 2014	C++	Java	Approach 2

As seen in Table 1, at first Java was used as the programming language for teaching both CS I and CS II introductory level undergraduate courses in Fall 2012. Then, starting with Spring 2013, during four consecutive semesters, where Preceding CS I column indicates the CS I level course

that students needed to take as a prerequisite for CS II. For the sake of ease of reference, we define and refer to the following labels for two separate approaches: Approach 1: Both CS I and CS II are taught in Java (only in Fall 2012). Approach 2: CS I is taught in C++ and CS II is taught in Java (starting with Spring 2013, continuously through Spring 2014).

Deciding which programming language to use is a Departmental level decision and the reason behind choosing Java (and C++) is to help students get exposed to an object oriented programming language so as to learn fundamental CS concepts easier and in a popular way (rather than conventional procedural languages).

It should also be noted that the main focus of CS I and CS II courses is to teach fundamental CS concepts and a programming language is used as a means to help program sample code while teaching these topics. According to UTD Department of Computer Science course curriculum (UTD Course curriculum URL, 2014), the CS I course introduces fundamental concepts as introduction to object-oriented software analysis, design, and development, classes and objects, object composition and polymorphism, sorting, searching, recursion, strings using core classes, inheritance and interfaces, and Graphical User Interfaces. The CS II course teaches more advanced topics as exceptions and number formatting, file I/O, implementation of primitive data structures - including linked lists, stacks, queues, and binary trees -, advanced data manipulation using core classes, and introduction to multi-threading, multimedia, and networking.

To examine how students perform under each approach, i.e. Approach I and Approach II, we measure student performance (based on numeric grade out of 100) per semester. Figure 1 illustrates these performance values.

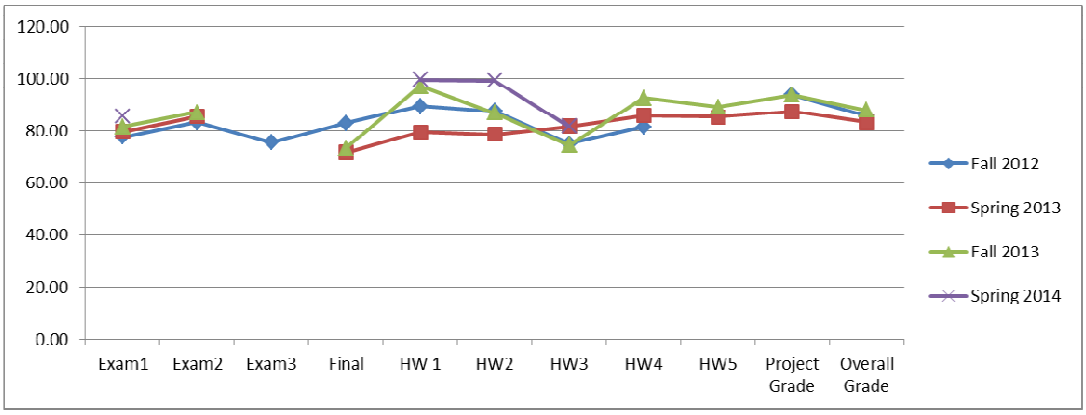


Figure 1. Plot for average student performance per assessment for each semester.

According to Figure 1, student success rates under Approach 1 (in Fall 2012, when both CS I and CS II are taught in Java) outperform Approach 2 (in Spring 2013 through Spring 2014, when CS I is taught in C++ and CS II in Java) only in the final exam assessment type. For all other assessments, students either perform the lowest under Approach 1, or their grades are better at least in one of the Approach 2 based semesters. It should be noted that in Figure 1, some assessment scores are not applicable to all semesters, (such as there is no Exam 3 in Spring 2013 and Fall 2013, and for the Spring 2014 semester, some grades are not available yet as the semester still continues as of March 2014).

Table 2: Average student performance data per assessment for each semester.

Semester	# of students	Exam1	Exam2	Exam3	Final	HW 1	HW2	HW3	HW4	HW5	Project	Overall Grade
Fall 2012	185	77.52	83.28	75.39	82.82	89.37	87.51	74.95	81.44		93.65	85.53
Spring 2013	45	79.53	85.44		71.71	79.41	78.52	81.57	85.72	85.30	87.26	83.11
Fall 2013	120	81.42	86.93		73.19	96.86	86.75	74.36	92.50	89.02	93.65	87.78
Spring 2014	47	85.55				99.52	99.11	81.60				

Table 2 lists the numeric scores that Figure 1 bases itself on. We solely focus on student performance in CS II, as this should shed light on whether adopting Approach 1 or Approach 2 helps students perform better. Therefore, the *# of students* column in Table 2 contains the number of students enrolled to CS II course for the corresponding semester. Table 2 shows that Exam 1 scores continue to improve as we progress chronologically through semesters. Exam 2 scores are very close to each other, though with insignificant differences. Students seem to have performed the best in the Final exam, when UTD adopted Approach 1, and it deteriorated later on. In terms of homeworks, students seem to be better doing in each homeworks. Actually, the underlying reasons behind them should not only be related to solely student effort, or to the case where Approach 2 is adopted, but also the instructors should be credited for putting their experience and insight into making it a smoother transition to switch from C++ to Java. Still, it would be too soon to conclude that all semesters under Approach 2 do better on average, as some data are not available by the end of Spring 2014.

In order to analyze individual student performances per semester, we plot the charts in Figures 2, 3, and 4 for Fall 2012, Spring 2013, and Fall 2013 respectively. For privacy concerns, student names are masked with alphabet letters.

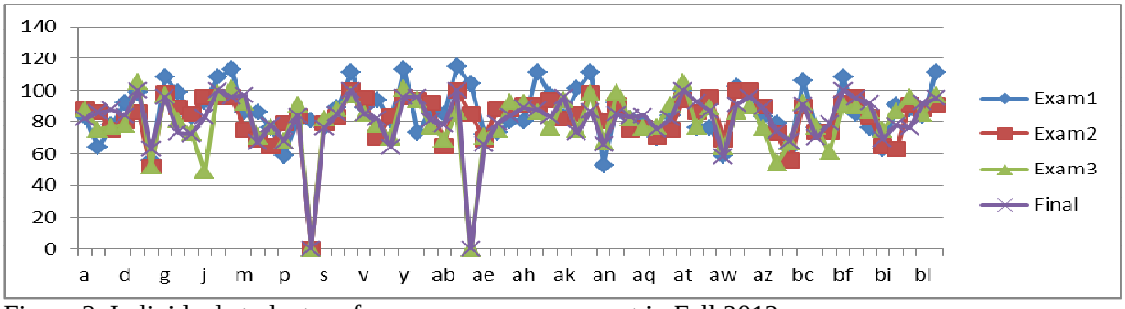


Figure 2. Individual student performances per assessment in Fall 2012.

As seen in Figure 2, individual students perform consistently good (or bad) in the 3 midterm exams and Final exam for Fall 2012. It should be noted that for visibility purposes, the chart in Figure 2 is plot on selected data (particular students and particular assessments only, even though the full set of data is available for all 185 students and 11 assessment categories). The sudden drops in the figure shows students who did not participate in this particular assessment.

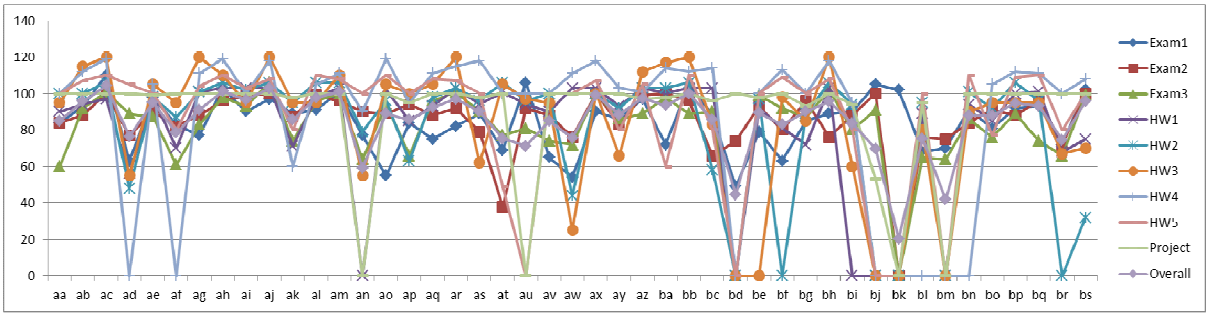


Figure 3. Individual student performances per assessment in Spring 2013.

Figure 3 illustrates the student performance for 2 midterm exams, Final exam, 5 hws and project and overall grade on an individual basis for Spring 2013. According to the figure, individual student performances are consistent per assessment.

Similar to Figures 2 and 3 above, Figure 4 shows the individual student performances on arbitrarily selected students and assessment categories demonstrate a consistent trend.

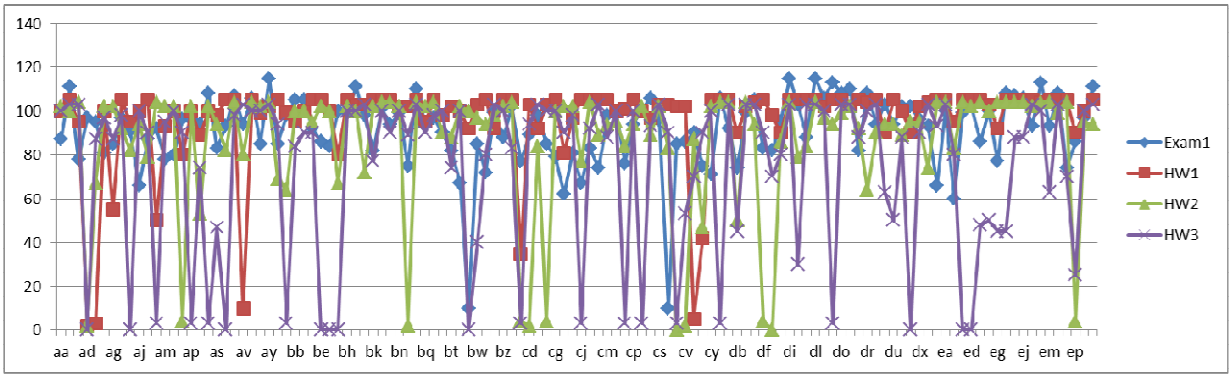
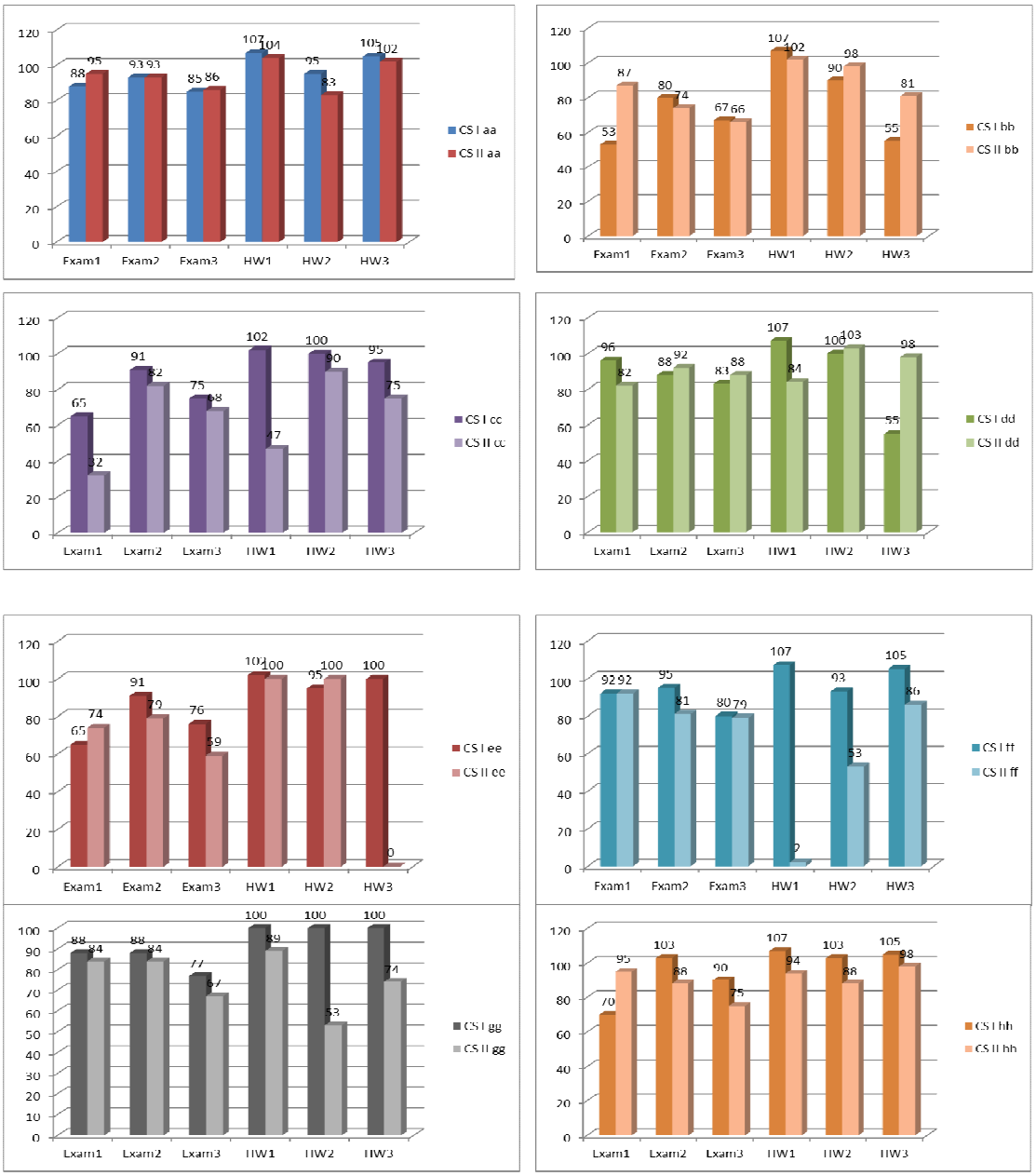


Figure 4. Individual student performances per assessment in Fall 2013.

Our work also explores the effect of incorporating Approach 2, i.e. how students perform when they take CS I in C++, and CS II in Java. There are 10 such students monitored, and we present the performance scores for these students who took CS I in Spring 2013, and CS II in Fall 2013. Again, student names are masked with alphabet letters for privacy (Figure 5).



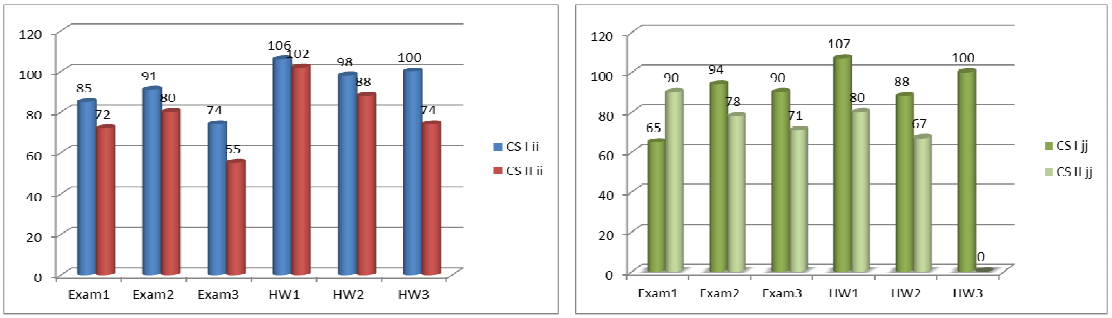


Figure 5. Same student taking CS I in C++ (Spring 2013) and CS II in Java (Fall 2013).

According to Figure 5, in CS II (in Java) the same student performed sometimes better, and some other times worse than how he performed in CS I (in C++). Specifically, for Exam 1: on average in 6 out of 10 cases, students perform better (or the same) in CS II with Java, when they had CS I in C++. This actually explains why Exam 1 performs the poorest in Fall 2012, when Approach 1 (both CS I and CS II in Java) was employed (Table 1). For Exam 2 and Exam3: Only in 20% of Exam2 (as well as Exam3) grades are equal or better in CS II (in Java), which follows CS I taught in C++. For HW1: All students get higher scores in CS I (in C++) then they do in CS II (in Java). This explains well that students need to gain some experience and acquaintance to start performing satisfactorily in a new programming language. Starting with HW2, 30% of students start to score better in CS II with Java, in spite of a C++ background in CS I. And finally for HW3, 20% of the students could achieve satisfactory (better than or equal) scores with CS I (in C++) and CS II (in Java). For HW3, it should be taken into consideration that some students have totally missed the submission and this is why they scored that low.

One of the goals of this paper is to observe the effect of changing the programming language in two introductory level courses into retention rates. To investigate this, we inquire the course drop rates for the four semesters that we have collected the statistics for. Certainly, our focus is on CS II, as it is the course where we see the immediate results of changing the programming language from what was taught in CS I (that is C++) to Java. For the sake of generality, we calculate the *Failures* column as the total of number of F (failure) grades and W (withdraw) end of semester grades for the CS II course per semester. Table 2 shows these values.

Table 2. Drop rates per semester.

	Method	# of students	Failures	Drop rate
Fall 2012	Approach 1	185	6	3.24%
Spring 2013	Approach 2	45	4	8.89%
Fall 2013	Approach 2	120	2	1.67%
Spring 2014	Approach 2	42	N/A	

We see that drop rates under each approach vary in a large span. Though in Fall 2012 under Approach 1 the drop rate is quite low at 3.24%, in Fall 2013 under Approach 2 this rate is even lower at 1.67%. So, this can be considered as a promising result. Still, the considerable jump at 8.89% drop rate is a significant change that needs to be investigated to make sure whether it is an isolated result, or an ongoing tendency with data in coming semesters.

4. CONCLUSION AND FUTURE WORK

We present the four consecutive semester results of changing programming language in two introductory level courses CS I and CS II in Computer Science major at University of Texas at Dallas (UTD) Department of Computer Science. We first analyze Approach 1 where both CS I and CS II are taught in Java, then Approach 2, where CS I is taught in C++ and CS II in Java. Results show that as opposed to the prejudice, students perform better (or at least now worse) under Approach 2, as compared to Approach 1.

We also investigate if this change causes increased drop rates. Our results show that for one semester Approach 2 yielded better (lower) drop rates than Approach 1, and in another worse (higher) rates than Approach 1. This indicates that the study needs to be expanded to future semesters to reach a solid conclusion.

As part of the future work, authors are considering to involve other factors, such as course content, measuring the effect of mentoring efforts into the success rates of students, as well as the drop rates.

5. REFERENCES

Alvarado C., Judson E., "Using Targeted Conferences to Recruit Women into Computer Science", Communications of the ACM, Vol. 57, Issue 3, March 2014, pp. 70-77.

Barr V., "Create two, three, many courses: An experiment in contextualized introductory computer science", Journal of Computing Sciences in Colleges , 27(6), pp. 19-25.

Brown S., Yuan X., "Techniques for retaining low performing students: high-need student mentoring program", SIGCSE'14, pp. 708.

Crenshaw T. L., Chambers E. W., Metcalf H., Thakkar U., "A Case Study of Retention Practices at the University of Illinois at Urbana-Champaign", SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA, pp. 412-416.

Haines A., "How one college president is breaking down barriers for women in tech", Forbes;http://www.forbes.com/sites/85broads/2011/12/12/how-one-college-president-is-breaking-down-barriersfor-women-in-tech/ [Accessed on 3/11/2014]

Horwitz, S., Rodger, S. H., Biggers, M., Binkley, D., Frantz, C. K., Gundermann, D., Hambrusch, S., Huss-Lederman, S., Munson, E., Ryder, B., and Sweat, M. Using peer-led team learning to increase participation and success of under-represented groups in introductory computer science", SIGCSE 2009, pp. 163-167.

Newhall T., Meeden L., Danner A., Soni A., Ruiz F., Wicentowski R., "A Support Program for Introductory CS Courses that Improves Student Performance and Retains Students from Underrepresented Groups", SIGCSE'14, March 5–8, 2014, Atlanta, GA.

Robinson A., Pérez-Quiñones M. A., "Underrepresented Middle School Girls: On the Path to Computer Science through Paper Prototyping" SIGCSE'14, March 5–8, 2014, Atlanta, GA, USA.

UTD Course curriculum URL, 2014: http://coursebook.utdallas.edu/

Yardi S., Bruckman A., “What is computing? bridging the gap between teenagers' perceptions and graduate students' experiences. ICER '07 International Computing Education Research Workshop, Atlanta, GA, September 15 - 16, 2007, pp. 39-49.