A Blended Approach to Design an Introductory Programming Course for Non-CS Majors: Students' Feedback

Ms. Kwansun Cho, University of Florida

Kwansun Cho is an Instructional Assistant Professor of the Department of Engineering Education, in the UF Herbert Wertheim College of Engineering. She has been teaching introductory computer programming courses for engineers. She holds two Masters' degrees in Electrical and Computer Engineering from the University of Florida and Yonsei University, specializing in speech signal processing. Her educational research interests include improved flipped classroom teaching/learning for students, and computer- or web-assisted personalized learning.

Sung Je Bang
Dr. Saira Anwar, Texas A&M University

Saira Anwar is an Assistant Professor at Department of Multidisciplinary Engineering, Texas A &M University. Dr. Anwar has over 13 years of teaching experience, primarily in the disciplines of engineering education, computer science and software engineering. Her research focuses on studying the unique contribution of different instructional strategies on students' learning and motivation. Also, she is interested in designing interventions that help in understanding conceptually hard concepts in STEM courses. Dr. Anwar is the recipient of the 2020 outstanding researcher award by the School of Engineering Education, Purdue University. Also, she was the recipient of the "President of Pakistan Merit and Talent Scholarship" for her undergraduate studies.

A blended approach to design an introductory programming course for non-CS majors - Students' feedback

Abstract

Computer programming skills are no longer discipline bounded. Many national policy documents across disciplines make computer programming a fundamental skill needed for most occupation in modern economy. Resulting to the rise of the need, there is a growing demand over different disciplines, for developing an introductory programming course that targets non-Computer Science (non-CS) major students. This demand is more paramount as many students may not have been introduced to fundamentals of programming in high schools. According to a national survey, only 53% of high schools offer computer science courses. The scarcity of the availability of courses at high school level results in more difficulties, and no prior computer programming experience. For such students the deficit in base continues to grow in college with two important facets: 1) such students are reluctant to pursue engineering and computing majors and 2) these students find typical college programming courses more challenging and harder than many others who took programming in their high school, leaving them behind in courses.

Considering offering programming courses for non-CS major students as an ongoing challenge, in this paper, we present our blended approach to design an introductory Python programming course for students with non-CS majors. Also, as the course was designed for non-CS majors with diverse students from different academic backgrounds, it is imperative to hear from non-CS major students' perspective on the course and use their feedback for effective course changes and continuous improvement.

Keeping the blended approach, the instructor used various approaches in the courses for enhancing student engagement including 1) lecture in various forms including pre and post reading materials, live coding, and discussions, 2) informal student interaction with instructor and peer mentor, and in-person and online office hours, and 3) individual and collaborative assessments. For capturing students' feedback in this study we answer two research questions: RQ1) Which aspects of the Python programming course design do students like among activities related to lecture, informal interaction, and assessment, and why? RQ2) Is there any difference between students learning in individual and collaborative assessment? Which assessment works better?

In the full paper, the results are presented from the collected data of 26 students' perspectives, collected using an end-of-semester open-ended questionnaire. For research question 1, the data are analyzed using qualitative and quantitative content analysis approaches and described students' perspectives on blended approaches. For research question 2 , comparisons were performed using Mann Whitney statistical analysis.

We believe that sharing our blended approach, and students' perspective will help in providing us the information for improving the course. Also, for the community and society the meaningful insights will help the instructors and course designers for non-CS major courses for students engagement and learning.

Introduction

Computer programming skills are no longer discipline bounded [1] and have become increasingly ubiquitous, which impact every aspect of society in today's era [2], [3]. Not only curriculum designers and instructors but also students across disciplines have realized the importance of programming as a skill. Curriculum designers and instructors have designed courses to support students as much as they could. For example, Anderson and colleagues [4] designed and explored a learning environment to teach programming to liberal arts students. The authors argued the importance of teaching such skills to non-CS majors due to changing needs of the fields, specifically multimedia discipline. Where studies have identified the need for programming courses due to the reshaping of industry and graduate school demands [1], studies have also discussed the associated challenges for such courses and the need for separate courses for CS and non-CS majors students. The associated challenges include a lack of interest in programming courses among non-CS majors students and less exposure to mathematicsal skills required for computing majors [2], [4]. Also, such lack of exposure to basic skills makes learning to program difficult and conceptually complex for many students [5], [6]. However, studies have also examined various mechanisms to 1) design courses and 2) diversify pedagogical approaches to help students and learning the requisite skills. The approaches to designing and teaching courses include various active learning mechanisms to increase students' retention and success [7], [8]. Some examples of these mechanisms include group work, pair programming, and game-based approaches [7] [8]. It is noteworthy that among many strategies, researchers have argued that collaborative and social aspects of these activities work better for students learning. For example, Vihavainen and colleagues [8] reported the importance of collaborative work in a systematic review synthesizing the literature from 32 studies.

To provide a pathway to non-CS students into programming and coding, many literature studies have also emphasized the importance of using a blended approach to teaching courses [9], [10]. Among many stated benefits, the literature suggests that a blended environment with a blend of collaborative and individual self-paced activities offered using various modalities, including face-to-face lectures and online activities, and incorporating formal and informal interactions can help students learn [10], [11]. Overcoming the issues faced by students in traditional approaches, such approach can help students by increasing autonomy, flexibility, and convenience. Students develop various solutions to problems, interact better with the instructional team [12], and show better engagement [13].

Considering the advantages and importance of designing a programming course for non-CS majors students and offering them the opportunity to learn with autonomy, we designed a blended learning course. Inspired by the flipped class model, we introduced a blend of various activities distributed at home and in class.

In this paper, we examined the effectiveness of this newly designed course from students' perspectives. More specifically, we addressed the two research questions:

1) Which aspects of the Python programming course design do students like among activities related to the lecture, informal interaction, and assessment, and why?

2) Is there any difference between students learning in individual and collaborative exams? Which exam type works better?

Literature Review

As reliance on technology increases in all fields, programming skills are considered a required competency [1] for all students. The competency requirement is more apparent with the trends in the job market, where experience and programming knowledge are considered essential elements for computing and non-computing majors [14]. According to the report by BurningGlass, even people who hold non-programming jobs are expected to understand programming and be able to develop software [14].

It must be noted that according to the report by Code.org, only 53% of public high schools teach fundamental computer science topics, and only 37 states implemented at least five policies to make computer science a mandatory part of their education systems at K-12 space [15]. Such a lack in high schools may create a void or lack of interest in studying programming for college students [16]. While some CS-major students had extensive programming and computing knowledge, others, such as non-CS-major students, may have never been exposed to the same [17]. Considering this limitation and void, many universities began offering programming courses to students in a computer science program as well as students who are not part of any of their computer science programs to meet future career options demand [18 - 20]. For example, Yale University created a program that allows non-computer science major students to receive a certificate in programming [19]. All these programs aim to equip the students with the skills and competencies needed for career growth.

While some other programs tried to offer the same programming course to computing and non-computing majors, there have been several challenges. For example, with the growing enrollment for CS courses, the class sizes for CS classes kept increasing [7], which would have been on a further rise with non-CS majors enrolling in the same course. This increase in class size causes students to have trouble knowing if they are making progress with learning in class [21]. Also, the mix of CS and non-CS students creates an issue with non-CS students who are, the minority of the class for not being able to connect and relate to others easily [21]. Large classes also cause instructors to "weed out" students who do not perform as well [21]. Another noted issue was related to non-CS students, who generally do not have prior programming experience. Students with prior programming experience most likely review what they already knew before taking the course rather than learning new concepts in the introductory course; non-CS majors or students with no programming background could be struggling with courses. This causes the non-CS students to get "weeded out" [21].

Because of the reasons mentioned above, CS courses generally suffer from the highest dropout rates compared to courses in other fields [22]. For example, Kinnunen and Malmi [23] found that many CS minor students dropped out of first-level computer science courses at the annual rate of around 30 to 50% at a local university. Although reasons varied for each student, the common reason was that the students found the courses too complicated and did not know how to manage their schedule to follow the courses [23]. It is noteworthy that one difficulty of programming courses partly comes from the fact that, traditionally, CS course instructors teach

programming through static coding [24]. Static coding is when instructors teach students how to program using pre-written code and compile them as examples. Researchers began to question if this is truly an effective way to teach students how to write programs [24]. Due to the ineffective nature of the previous attempts, it is essential to use instructional methods that may take an approach that resonates with varying learning styles [25]. Literature suggests that one such way to design courses could be the blended approach [26], which uses various teaching pedagogies to create an active and constructive environment [9]. For example, one approach that could be part of the blended approach is the live coding method. Live coding is a teaching method where instructors write the example code on their own in front of the class and then compile it [24]. Prior studies suggest that live coding is more effective in helping students understand programming concepts than static code examples traditional CS lectures generally use [24]. Another approach included in this blended approach is the positive informal relationship between students and lecturers. Jaasma and Koper [27] found that this informal relationship generally builds a stronger motivation within students to perform better in class. Similarly, Guerrero and Rod found a strong correlation between students' office hours visits and their academic performances [28]. Also, studies emphasized the use of individual and collaborative assessments. Individual assessments help students gain more motivation and improve their academic performance [29]. Collaborative assessments allow students to learn more effectively in class because they make the students do their own interpretation of the learning material and build knowledge and experience through social communication [30].

Considering the effectiveness of different approaches discussed in the literature and combining them in one course, we designed an introductory Python programming course for non-CS students that uses a blended approach. This course was designed using the feedback of non-CS students. We believe that combining these approaches and implementing them into this course will significantly help the non-CS students gain motivation to perform well despite having little to no background in programming.

Research Methods

We used a multimethod approach to understand students' feedback and perspective on the new course offered to non-CS majors students.

Site and participants

We collected the data from 26 non-CS majors enrolled in an elective Python programming course titled "Python Programming for Engineers" at a large R1 university. Table 1 provides information about the demographics of the participants.The topics of the course revolved around various programming principles such as variables, opeators, computing with numbers, objects and graphics, sequences, functions, decision structures, loop structures, and basic object-oriented design.

Table 1 Demographics of the students

	No of Students	**Percentage**
Gender		
Male	22	85%
Female	4	15%

Race/Ethnicity		
Hispanic or Latino, or Spanish Origin of any race	6	23%
Asian	2	8%
Black or African American	1	4%
White	16	62%
Two or more races	1	4%
Academic Year		
Freshmen	7	27%
Sophomore	9	34%
Junior	8	31%
Senior	2	8%
Programming Experience		
Some Experience	12	46%
No Experience	14	54%

Course Design

The course was designed around a semi-flipped model with activities and assessments spread over before class, during class, and after class

The course was designed with the following activities.

Before class

1) Pre and post-class reading material (Reading Materials): Students were asked to read an assigned chapter before class and the posted in-class lecture notes and examples after class. The weekly assigned chapter provided some background and concepts to the students and helped them to understand when the instructor used a variety of teaching techniques and to engage in problem-solving activities.
2) Individual readiness assurance test (iRAT): Students were asked to take a simple quiz that had eight to ten questions relating to the contents of the assigned chapter of the textbook.

During class

3) Live coding or demonstration by the instructor (Live coding): For each topic, the instructor demonstrated the programming concepts using live coding in each class. During live coding, the instructor demonstrated the mechanism to solve a problem from start to finish.
4) Informal class discussion with peer (Peer Discussion): Students were highly encouraged to discuss their high-level approaches (or pseudo codes) to solve weekly in-class assignments.
5) Informal interaction with the instructor (Instructor Discussion): Instructor was available to individually discuss any thoughts and questions that were related to each week's concepts.
6) Informal discission with the peer mentor (Mentor Discussion): Peer mentor was available to individually discuss any thoughts and questions that were related to each week's concepts.
7) Individual class assignments (iCAT): Students were required to complete three individual programming tasks per week.

After class

8) In-person office hours with the instructor (Instructor in-Person OH): Students could use the in-person office hours to discuss their questions and/or concerns.
9) In-person office hours with the peer mentor (Mentor in-Person OH): Students could use the in-person office hours to discuss their questions and/or concerns.

10) Online office hours with the instructor (Instructor Virtual OH): Students could use the zoom office hours to discuss their questions and/or concerns.
11) Online office hours with the peer mentor (Mentor Virtual OH): Students could use the zoom office hours to discuss their questions and/or concerns.
12) Individual exams (iExams): There were three in-person exams
13) Individual homework assignments (iHW): There were three individual programming homework assignments during the semester.
14) Collaborative exams (cExams): Right after each iExam, students was assigned to a group of five peers to discuss and solve the exam questions collaboratively.

Measures

The data were collected using a survey questionnaire with a blend of Likert scale questions to measure how much students like or dislike each course activity and open-ended questions. The Likert scale questions asked the students to rate each activity on a scale of likeness from 1 (dislike extremely) to 5 (like extremely). The open-ended questions were designed to understand their reasons for liking or disliking some activities over others. Table 2 presents the sample items.

Table 2 Sample questions from the survey questionnaire

Category	**Sample Questions**
Likert scale question	In the course, the instructor introduced you to various activities. Please rate each activity on a scale of likeness from 1 (dislike extremely) to 5 (like extremely).
Reason for liking activities	Please pick your top three liked choices of class activities and describe your reasons for liking them.

Also, we used students' scores in the exams to measure students' learning. The course has three exams with individual and collaborative sections. We considered the score of these to assess to examine difference between students learning and to understand which assessment works better.

Data Analysis

We used Microsoft Excel for qualitative and quantitative content analysis. Also, we used SPSS v29.0 to conduct the related samples Wilcoxon signed ranked test.

For research question 1, we calculated the average of the Likert scale questions. Also, we coded the top three choices students liked or disliked using open-ended coding for content analysis. We used the identified codes for quantitative content analysis to examine how many students selected the activity type. We converted the results into percentages based on the response of the 26 students in total. For qualitative content analysis, we used open coding to understand students' reasons for liking or disliking an activity.

We selected the related-samples Wilcoxon signed ranked test for the second research question due to sample size and issues related to data normality. To conduct the analysis, we convert students' scores of individual exams (Part 1) and collaborative exam (Part 2) into percentages so that each exam portion's score is evaluated out of 100%.

Results

Course Activities

To answer the research question and explain which aspects of the programming course students liked the most, we first calculated the average students' rating of the Likert scale question asked for each activity. Table 3 presents the averages for each aspect of the course.

Table 3 Average on Ratings of Likert Scale Survey

Class Activity	**Average**
Reading materials	3.62
iRAT	2.81~
Live Coding	3.85
Peer Discussion	4.31*
Instructor Discussion	4.31*
Mentor Discussion	4.19
iCAT	4.12
Instructor in-Person OH	3.57
Mentor in-Person OH	3.69
Instructor Virtual OH	3.5
Mentor Virtual OH	3.69
iExams	2.81~
iHW	4.26*
cExams	4.35*

* indicates top choices students liked; ~ indicates top choices students disliked

From the averages, it was evident that the highest average was assigned to the collaborative exam (4.35) section of the assessments, and the least liked assessment was individual exams (2.81) and individual readiness assessments (2.81). Within the "before class" category, the lowest rating was assigned to individual readiness assessment (2.81). Within the "During class time" category, students liked the discussion with both peers and instructors equally (4.31), while although the average is high, with the in-class category, they didn't like the individual class assignments (4.12). Within the category of "After class," students least liked individual exams in assessment (2.81), and instructor virtual office hours (3.5).

In the second step, we examined these results in connection with the open-ended questions, where students picked their top three choices for activities they liked or disliked and explained their reasons. Using Quantitative Content Analysis, we coded and identified students' top three choices they liked and the top three choices they disliked.

For the top choices students liked, the results indicate that when asked qualitatively, most students selected individual homework assignments (77%), individual class assignments (77%), and collaborative exams (46%) as their top three choices. Afterward, we conducted the qualitative content analysis and identified that students felt positive emotions and liked the learning provided during these activities. For example, a student with no prior programming experience selected the three as their top choice and said, "*Class assignments, homework assignments, and collaborative exams were my favorite parts of this course. I enjoyed the*

freedom of being able to create my own solution to a problem. The assignments were engaging and challenging. I also especially liked being able to pool my knowledge with other classmates in order to do the very best that we can on problems." In the same context, another student with no prior programming experience selected the same choices and said "*I really enjoyed the class assignments and homework assignments because I feel they accurately covered the material in class while also challenging me enough to have to think creatively to build the program. I also enjoyed collaborative exams because I liked being able to hear how others think and argue my points.*"

Considering the perspectives of students with some programming experience, and selected the same three options and said, "*My 3 favorite activities from class were the HW assignments, in class assignments, and part 2 of the exams. The bigger HW assignments were very familiar to me because I have experience programming and gave me a chance to learn by doing. I liked the in class assignments for the same reason but I think they're smaller scale made them good for learning about specific topics. Part 2 of the exams were a nice change of pace that I think helped facilitate extra learning from the exams that wouldn't have been possible otherwise*."

Among the choices of lecture and informal interaction, students loved having an informal discussion with the instructor (23%). For example, a student said, "*informal interactions with the instructor and peer mentor allowed me to learn the best. This is because I was not afraid to ask questions, and I could discuss anything about programming, which broadened my understanding.*"

For the choices students disliked, the results indicate that when asked qualitatively, although a good number of students didn't dislike anything (19%), students selected individual readiness assurance test (46%), individual exams (19%), and Live coding (15%) as their top three choices. However, with qualitative content analysis, it is also noteworthy that most students, when showing their disapproval of these activities, also indicated the reason for course setup or time constraint as a factor. For example, a student mentioned, "*The main thing I disliked were the iRats because I found the timing to be very inconvenient. I would have strongly preferred if they were due at midnight the day of class instead of before class. There were no other class activities I disliked, but being able to work on the in class assignments throughout the first part of class would have been nice*." Another student who selected both the individual readiness assurance test and individual exam as the disliked aspect suggested "*iRat - Did not have the patience to read the book or could not remember key words in the questions. Exam Part 1 - Didn't really enjoy the setup/ felt so easy to make a mistake in the output questions that costs the whole problem.*" Similarly, a student who disliked live coding said, "*The in-class coding demos weren't very useful, mostly covered the information from the textbook that we were already required to read before that class session.*" Also, the other student mentioned "*live coding or demo, could be tedious if there is a lot to watch.*"

Students Learning in Exams

We used the data from three exams to understand the difference between students learning in individual and collaborative exams. The results of the related samples Wilcoxon signed ranked test are presented in Table 4.

Table 4. Results of the related samples Wilcoxon signed ranked test

	Individual exam Mean ± SD	Collaborative exam Mean ± SD	Test Statistics N = 26	p
Exam1	74.27 ± 15.82	96.31 ± 5.64	350.00	<.001**
Exam2	54.46 ± 14.98	79.41 ± 12.90	350.00	<.001**
Exam3	68.46 ± 15.18	86.45± 7.20	356.50	<.001**

*indicates <0.05; ** indicates <0.01

For Exam 1 and Exam2, 25 students showed the positive difference between collaborative and individual exams, and only one student showed a negative difference (see Figure 1 and Figure 2, respectively). However, for Exam3, although most students showed a positive difference between collaborative and individual exams, two students showed a negative difference. These results indicate that most students on the rank scored higher in collaborative exams than individual exams.

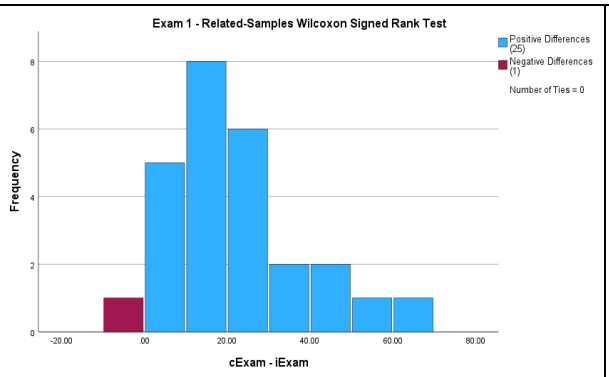

Figure 1. Collaborative and individual Exam1

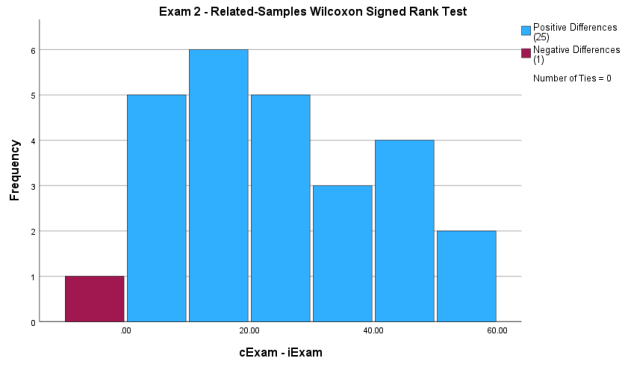

Figure 2. Collaborative and individual Exam2

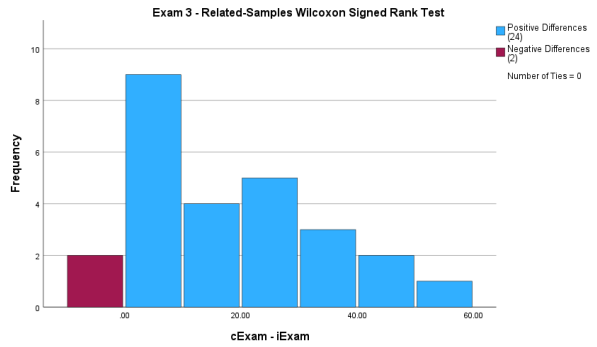

Figure 3. Collaborative and individual Exam3

Discussion and Conclusion

In this paper, we examined the students' perspective about the newly designed programming course offered in Python for non-CS majors students. In this blended course, several aspects were introduced which are different from traditional approaches to teaching programming, especially to non-CS majors. The end-of-semester survey questionnaire was circulated to

examine the students' perspectives, including both Likert Scale and open-ended questions. Additionally, students' learning was examined in individual and collaborative exams within three-course exams, spaced relatively at equal intervals during the course.

The study results indicate that most students liked the course and found it useful for their future careers (88% found it helpful, 4% found it not helpful, and 8% didn't respond). Also, when looking at class activities, most students found collaborative activities, i.e., collaborative exams, informal discussion with peers, or informal interaction with the instructor, as the highest-rated liked activities. Similarly, in the open-ended questions, most students, besides individual work, students picked collaborative exams as their top three choices. These results align with existing literature that suggests that collaborative activities are helpful in students learning [8], [31] and engagement [12], [32] within programming courses. Also, students' comparisons of individual and collaborative exams indicated that for most students, their learning was better in collaborative exams over individual ones.

One noteworthy result was related to students' choice of the most disliked activities. While students liked individual homework and individual class assignment, individual exams were not appreciated. One difference between the three activities was the pace at which students were required to complete these activities. While individual homework assignments and class assignments were self-paced, the exam was time-bounded, and the setup was structured. Another difference was in the nature of the activities. While individual homework and activities were related to problem-solving, individual exams were more of a test of understanding and knowledge with output, debugging, and multiple-choice questions. These findings align with previous literature that suggests that students feel frustrated with activities that test knowledge over skills [33], [34].

The results of this study must be viewed in light of several limitations and future directions. First, this study examined students' perceptions of a newly designed course. Although the approach is closer to mixed methods, we used a multimethod approach. Future studies can consider a more holistic examination of students' engagement and perception using multi-modal approaches [35], [36]. Also, the approach could have considered a defined quantitative and qualitative phase with a larger sample for the quantitative phase. The results of this study are limited to one small class size, one offering only. A future offering with larger student enrollments or several course offerings may provide an enhanced perspective. Also, in this study, no process data was included. Future studies could rely on supplementary information such as observation protocols [32], which may help to see researchers' perspectives on students' engagement and learning. Also, studies suggested that gender, ethnicity, and other demographics could significantly impact students' ways of perceiving needs for learning programming [37]. Future studies could account for such variables.

These results are interesting as they highlight that novices, especially non-computing majors, require variation in teaching pedagogies to perform and learn effectively. Also, it informs the instructors that they need to blend the pedagogies to provide ample social and interactive activities for students' engagement and learning. Also, the results emphasize the importance of developing problem-solving skills by programming over designing activities that revolve around understanding concepts.

References

[1] S. Mitchell, K. Cole, and A. Joshi, "X+ CS: A computing pathway for non-computer science Majors," in *ASEE Mid Atlantic Section Spring Conference*, *Baltimore, MD, USA, March 28, 2020.*

[2] T. L. Lenox, C. R. Woratschek, and G. A. Davis, "Exploring declining cs/is/it enrollments," *Jornal of Information System Education*, vol. 6, no. 44, pp. 1–11, 2008. Available: http://isedj.org/6/44

[3] I. Becerra-Fernandez, J. Elam, and S. Clemmons, "Reversing the landslide in computer-related degree programs," *Communications of the ACM*, vol. 53, no. 2, pp. 127–133, 2010. Available: https://doi.org/10.1145/1646353.1646387

[4] P. B. Andersen, J. Bennedsen, S. Brandorff, M. E. Caspersen, and J. Mosegaard, "Teaching programming to liberal arts students: a narrative media approach," *ACM SIGCSE Bulletin*, vol. 35, no. 3, pp. 109–113, 2003. Available: https://doi.org/10.1145/961290.961543

[5] M. Guzdial, "Programming environments for novices," in *Computer Science Education Research, S. Fincher and M. Petre, Eds.* London: Taylor & Francis, 2004, pp. 127–154. Available: https://doi.org/10.1201/9781482287325

[6] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *ACM SIGCSE Bulletin*, vol. 39, no. 2, pp. 32–36, 2007. Available: https://doi.org/10.1145/1272848.1272879

[7] N. Nagappan, *et al.*, "Improving the CS1 experience with pair programming," *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 359–362, 2003. Available: https://doi.org/10.1145/792548.612006

[8] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review of approaches for teaching introductory programming and their influence on success," in *International Computing Education Research*: *Proceedings of the tenth Annucal Conference on International Computing Education Research, ICER '14, Glasgow Scotland United Kingdom, August 11-13, 2014*, pp. 19–26. Available: https://doi.org/10.1145/2632320.2632349

[9] T. B. Bati, H. Gelderblom, and J. van Biljon, "A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa," *Computer Science Education*, vol. 24, no. 1, pp. 71–99, 2014. Available: https://doi.org/10.1080/08993408.2014.897850

[10] A. Alammary, "Blended learning models for introductory programming courses: A systematic review," *PloS one*, vol. 14, no. 9, 2019. Available: https://doi.org/10.1371/journal.pone.0221765

[11] J. Q. Dawson, M. Allen, A. Campbell, and A. Valair, "Designing an introductory programming course to improve non-majors' experiences," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, *SIGCSE '18, February 2018*, pp. 26–31. Available: https://doi.org/10.1145/3159450.3159548

[12] C. R. Graham, "Blended learning systems," in *Handbook of blended learning: Global perspective, local designs, C. J. Bonk and C. R. Graham, Eds.* San Franscisco, CA: Pfeiffer Publishing, 2006, pp. 3–21.

[13] D. Bath and J. Bourke, Getting started with blended learning. GIHE, 2010.

[14] Burning Glass Technologies, "Beyond point and Click: the expanding demand for coding skills," June 2016. [Online]. Available: http://hdl.voced.edu.au/10707/429586

[15] S. Roberts, M. O. Glennon, and H. Weissman, "2022 State of Computer Science Education," code.org, 2022. [Online]. Available: https://advocacy.code.org/2022_state_of_cs.pdf
[16] L. Carter, "Why students with an apparent aptitude for computer science don't choose to major in computer science," *ACM SIGCSE Bulletin*, vol. 38, no. 1, pp. 27–31, 2006. Available: https://doi.org/10.1145/1124706.1121352
[17] L. J. Sax, K. J. Lehman, and C. Zavala, "Examining the enrollment growth: Non-CS majors in CS1 courses," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, Seattle, Washington, March 8-11, 2017*, pp. 513–518. Available: https://doi.org/10.1145/3017680.3017781
[18] P. K. Chilana, *et al.*, "Perceptions of non-CS majors in intro programming: The rise of the conversational programmer," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC),* pp. 251–259, October 2015. Avaialable: https://doi.org/10.1109/VLHCC.2015.7357224
[19] Yale University, "Yale now offering a certificate in programming for non-CS majors," September 1, 2020. [Online]. Available: https://cpsc.yale.edu/news/yale-now-offering-certificate-programming-non-cs-majors
[20] University of Illinois, "Illinois computing accelerator for non-specialists (iCAN)." [Online]. Available: https://cs.illinois.edu/academics/graduate/ican
[21] T. Camp, S. Zweben, E. Walker, and L. Barker, "Booming Enrollments: Good Times?," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, Kansas City, Missouri, March 4-7, 2015*, pp. 80–81. Available: https://doi.org/10.1145/2676723.2677333
[22] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003. Available: https://doi.org/10.1076/csed.13.2.137.14200
[23] P. Kinnunen and L. Malmi, "Why Students Drop Out CS1 Course?," in *Proceedings of the second International Workshop on Computing Education Research, ICER '06, Canterbury United Kingdom, September 9-10, 2006*, pp. 97–108. Available: https://doi.org/10.1145/1151588.1151604
[24] M. J. Rubin, "The effectiveness of live-coding to teach introductory programming," in *Proceedings of the 44th SIGCSE Technical Symposium on Computer Science Education, March 2013,* pp. 651–656. Available: https://doi.org/10.1145/2445196.2445388
[25] A. T. Chamillard and D. Karolick, "Using learning style data in an introductory computer science course," in *Proceedings of the thirtieth SIGCSE Technical Symposium on Computer Science Education, SIGCSE '99, March 1999,* pp. 291–295. Available: https://doi.org/10.1145/299649.299790
[26] F. Alonso, D. Manrique, L. Martinez, and J. M. Vines, "How blended learning reduces underachievement in higher education: An experience in teaching computer sciences," in *IEEE Transactions on Education*, vol. 54, no. 3, pp. 471–478, 2010. Available: https://doi.org/10.1109/TE.2010.2083665
[27] M. A. Jaasma and R. J. Koper, "The relationship of student-faculty out-of-class communication to instructor immediacy and trust and to student motivation," *Communication Education*, vol. 48, no. 1, pp. 41–47, 1999. Available: https://www.doi.org/10.1080/03634529909379151

[28] M. Guerrero and A. B. Rod, "Engaging in office hours: A study of student-faculty interaction and academic performance," *Journal of Political Science Education*, vol. 9, no. 4, pp. 403–416, 2013. Available: https://doi.org/10.1080/15512169.2013.835554
[29] J. H. McMillan and J. Hearn, "Student self-assessment: The key to stronger student motivation and higher achievement," *Educational Horizons*, vol. 87, no. 1, pp. 40–49, 2008. Available: https://www.jstor.org/stable/42923742
[30] E. Hargreaves, "The validity of collaborative assessment for learning," *Assessment in Education*, vol. 14, no. 2, pp. 185–199, 2007. Available: https://doi.org/10.1080/09695940701478594
[31] S. Anwar and M. Menekse, "Unique contributions of individual reflections and teamwork on engineering students' academic performance and achievement goals," *International Journal of Engineering Education,* vol. 36, no. 3, pp. 1018-1033, May 2020.
[32] S. Anwar and M. Menekse, "A systematic review of observation protocols used in postsecondary STEM classrooms," *Review of Education*, vol. 9, no. 1, pp. 81-120, 2021. Available: https://doi.org/10.1002/rev3.3235
[33] A. K. Lui, R. Kwan, M. Poon, and Y. H. Cheung, "Saving weak programming students: Applying constructivism in a first programming course," *ACM SIGCSE Bulletin*, vol. 36, no. 2, pp. 72–76, 2004. Available: https://doi.org/10.1145/1024338.1024376
[34] M. Guzdial and E. Soloway, "Teaching the Nintendo generation to program," *Communications of the ACM*, vol. 45, no. 4, pp. 17–21, 2002.
[35] I. Villanueva and S. Anwar, "Situating the place of multi-modal approaches place in engineering education research," *(Guest Editorial) Journal of Engineering Education*, vol. 111, no. 2, pp. 277-282, 2022. Available: https://doi.org/10.1002/jee.20460
[36] S. Anwar and M. Menekse, "First-Year engineering students' experiences and motivation amid emergency remote instruction," in *IEEE Transactions on Education*, 2023. Available: https://doi.org/10.1109/TE.2023.3236241
[37] M. Menekse, X. Zheng, and S. Anwar, "Computer science students' perceived needs for support and their academic performance by gender and residency: An exploratory study," *Journal of Applied Research in Higher Education*, vol. 12, no. 5, pp. 1025-1044, 2020. Available: https://doi.org/10.1108/JARHE-07-2019-0194