Flexible Learning Cycles: Introducing Agile Constructivist Pedagogy

Dr. Mike Borowczak, University of Central Florida

Dr. Borowczak, currently an Associate Professor of Electrical and Computer Engineering at the University of Central Florida, has over two decades of academic and industry experience. He worked in the semiconductor, biomedical informatics, and storage/security sectors in early-stage and mature startups, medical/academic research centers, and large corporate entities before returning to the US public university system full-time in 2018. His current research interest are focused on automation for design, development, and assessment of resilience, robustness, and security of electronic devices and systems which currently includes topics such as: advanced/novel cryptographic logic primitives (polymorphic, homomorphic, quantum-enhanced), assessment and evaluation of assistive technologies on semiconductor design and post-manufacturing operation, development and detection methods for AI-based sabotage. He and his students have published over 100 journal and conference publications. His research has been funded (˜$8.5M since 2018) by federal, national, state, and industrial entities.

Dr. Andrea Carneal-Burrows Borowczak, University of Central Florida

Dr. Andrea C. Burrows Borowczak is the Director and a Professor in the School of Teacher Education at the University of Central Florida (UCF) in the College of Community Innovation and Education (CCIE). She received her doctorate degree from the University of Cincinnati in 2011. She has received multiple awards and grant funding, published many journal articles and conference works, and continues promoting STEM education and integration in traditional and non-traditional settings. She was elected PCEE Division Chair for 2022-2023 and 2023-2024.

Flexible Learning Cycles: Introducing an Agile Constructivist Pedagogy

Andey Robins, Mike Borowczak, and Andrea C. Borowczak
University of Central Florida

Abstract

This study explores a novel instructional approach that integrates agile methodologies with traditional teaching practices to enhance learning outcomes in an introductory Computer Science (CS1) course. A high rate of D, F, and Withdraw (DFW) grades, coupled with student dissatisfaction in previous iterations, prompted a comprehensive redesign of the CS1 curriculum at a mid-sized public land-grant university. The redesigned course emphasizes student-centered learning, incorporating strategies such as near-peer instruction, supplemental tutoring, and flexible assessment timelines. These changes aim to help students engage more effectively with core computing concepts at their own pace. Grounded in a constructivist framework, the course encourages active knowledge construction through hands-on activities and iterative feedback. Preliminary results show a substantial reduction in DFW rates and increased student satisfaction, leading to a cohort better prepared for advanced computing coursework and more capable of applying computing principles across disciplines. This paper details the redesigned course structure and evaluates its effectiveness through a constructivist lens, suggesting that this agile-inspired model promotes more positive learning experiences and improved outcomes in CS1.

Introduction

The first computer science course (CS1) serves as a critical foundation for students' understanding of computational thinking and core computer science principles. As students' initial exposure to programming concepts, CS1 significantly influences their perception of computer science and their potential success in the field. This work explores a novel instructional approach at a public land-grant institution in the Mountain West region, serving approximately 10,000 students. By adapting key principles from agile software development—including rapid concept iteration, regular student retrospectives, and sprint-based content delivery—this proposed approach responds to student needs and backgrounds to provide content specifically tailored to students with the goal of improving student motivation to complete the course as measured by the course success rate. This work builds upon on evidence that agile-aligned pedagogies support reflective learning and communication in computing courses [1].

The goal of this work is twofold:

1. Provide a retrospective analysis of a **novel instructional model**, offering sufficient detail for other educators to adopt, adapt, or extend the approach.
2. Demonstrate the effectiveness of this modified instructional approach in addressing stagnation in content delivery, preparing students for the rapidly evolving field of computer science.

In a field as rapidly changing as computer science, modifications to the methods of instruction may help intrinsically prepare students for this rapidly changing ecosystem.

Theoretical Framework

Constructivism as an educational theoretical framework has often been applied to the sub-field of computer science education (CSE). The theory, at its core, focuses on mental models created by students and emphasizes students' construction of mental models as a proxy for transferable understanding [2]. Therefore, the students are able to develop their understanding of computation and the inner workings of the computer as a way to understand the limits and possibilities of computational thinking. Developing computing artifacts, such as computer programs, is the primary learning approach employed by many constructivist approaches to CSE [3].

The primary hurdle identified when applying constructivist frameworks to CSE is the lack of a functional model for students [2]. Since many students lack an understanding of how computers perform tasks they believe to be simple (such as opening a web page or saving a file), they enter CS courses without a foundational framework on which to scaffold new concepts and ideas. As a result, a primary goal of early computer science courses must be the development of a functional model for how computation is performed by computers and how computational problem solving differs from other domains.

Some researchers have argued that "[constructivism] is less effective and less efficient than instructional approaches that place a strong emphasis on guidance of the student learning process" [4]. While there is valid concern that students may not be successful working under the assumption voiced in [4] that "knowledge can best ... be learned through experience," the conclusion that this educational model must therefore be made up of "unguided practical or project work" is not necessarily true. In fact, it has been directly contested as "mistakenly conflat[ing] problem-based learning and inquiry learning with discovery learning" [5]. This interdisciplinary, constructivist approach aligns with our agile model's emphasis on learning-by-doing in CS1.

The instructional approach described in this work joins experimental spaces for students to construct knowledge with focused guidance and direction from instructors to avoid these specific problems.

Early Computer Science Course Content

CS1 courses generally serve two overlapping but distinct purposes. First, from a constructivist perspective, the goal for CS students is to develop a working mental model for how computers (devices) and computation interoperate. Second, a less-frequently articulated goal of CS1 courses is to share computational methods and means of reasoning to non-CS students. Meeting the needs

of these divergent groups of students requires meaningful content, guided programming exercises, opportunities for creative self-expression, and tangible rewards. These requirements conflict with existing collegiate education approaches, which often omit early emphasis on collaborative and inquiry-based practices that can improve engagement and retention [6].

Students see the best way to learn CS content as programming and practicing, but many students lack the intrinsic motivation to explore these ideas without the external motivation provided by a course and a grade [7]. As a result, student expectations center on being guided through programming exercises which are meaningful to them; being allowed some degree of creative self-expression; and being rewarded for such practice with the tangible outcome of good grades [8]. These needs are often at odds with traditional approaches to collegiate education which see an instructor lecturing and dictating assessments such as homework and exams to assess the retention of lecture content.

A number of works have sought to address this apparent gap between expectations of students and the reality of the courses they must take. A common approach is to formulate two distinct courses, targeting CS students with a "traditional" CS1 course and targeting non-CS students with a "computational thinking" course [9,10]. Treating these two groups as distinct and without overlap has been shown to be successful with respect to keeping students in CS1 courses when they are non-major students; however, not all institutions have the capacity and resources to separate CS1 into two unique courses. One intention between this distinction between major specific and non-major specific approaches to early CSE is to provide different motivations to different students [11].

A commonly held belief about the computer science field is that courses are "dry" and "boring," and it has been suggested that one possible explanation for such impressions is the use of "academic" languages for early courses such as Scheme or Logo [12]. Python has emerged as a clear favorite for early CSE, and CS1 specifically, in order to address the needs of students to learn a production ready language. A major benefit is that Python, as a language, is approachable for learners without a preconceived framework of what computation is.

Other Approaches

While the content focus of CS1 courses is well studied and documented in the literature, a sampling of interesting approaches is included here. Course content has been presented which ranges from games programming to robotics to data analysis [13]. The course in Anderson et al. [13] utilized a data centered focus with the goal of allowing non-major students to have meaningful skills they could adapt to the computational needs of their specialized domains. Data visualization played a major role in both instructing students, by visualizing the movement of data within the computational system, and in assignments, by developing visualization artifacts which were a result of students' computing efforts. This content focus was paired with the agile method of instruction laid out in the remainder of this work.

Other novel methods of instruction have been explored in the past with the intention of adapting modernized CSE approaches to CS1 courses. For instance, robotics-based constructivist instruction has demonstrated potential to foster creativity and deep engagement in computing concepts [14]. Active learning, flipped classrooms, and collaborative assignments are all modifications to the traditional lecture-first course structure which have seen assorted degrees of improvement to

instruction [15]. Other domain specific practices, such as pair-programming, have also seen adoption within early CSE courses [16], specifically as an instructional method within lab sections and assignments. These approaches address concerns of their own, but often still suffer from the lag time between instruction, assessment, and feedback. The novel instructional approach presented in this work seeks to address this secondary hurdle and so applied a different model than prior works.

Agility

Agile software development practices are likely to be at least familiar to many CS faculty, but for educators for whom the software development life-cycle is less ingrained, we seek to provide a brief description of "agility" as it is used within the field of software engineering. Agile software development practices emerged in the early 2000s as a response to the limitations of traditional, linear design methodologies that dominated much of the 20th century [17]. Instead of creating a comprehensive initial design, agile employs sequential iterations that continuously refine both design and implementation. [18]. The principle of observing the state of the process, responding to current needs, and modifying future plans to address current concerns, embodies the aspects of agile being employed in this course. For a comprehensive examination of agile practices in software engineering, readers are directed to seminal works [19, 20].

This work first explores the non-standard structure of this course, including motivations for the modification from traditional lecture style modalities. Both qualitative and quantitative analyses are employed to assess the efficacy of this pedagogical model. We conclude by offering anecdotal advice after presenting two semesters of this modality and a call for further analysis in the short and long terms of this model in additional classroom settings.

Course Structure

While traditional lecture-based instruction remains prevalent in post-secondary education, it has faced criticism for potentially fostering student passivity. Despite these concerns, the lecture model persists for valid pedagogical reasons, as it enables efficient knowledge dissemination to large student populations, particularly in subject areas where structured, sequential content delivery proves most effective. Many students have also developed successful learning strategies within this conventional framework. This course's design deliberately sought to bridge the gap between traditional lectures and more active learning approaches by implementing a collaborative **"I Do, We Do, You Do"** methodology. This structured pipeline facilitates knowledge transfer from instructors to students while promoting increased learner engagement

Pedagogical Model: Agile Constructivist

Two primary objectives informed the development of the new instructional mode - Agile Constructivist Pedagogy - presented in this work. First, working under the premise that students value practice as a means to develop content mastery, a primary goal was to utilize assessments which can double as practice. Secondly, to allow for early intervention and correction of misconceptions

held by students, a cyclic, agile system which can adapt to the student needs of each unique course section was desired. These two aims converge in a course structure which intentionally mirrors the agile software development patterns common in the software engineering field while also aligning to gradual release of responsibility from instructor to learners - informally ”I Do, We Do, You Do.”

Each cycle, often referred to as ”sprints” in the agile world, takes place over a three week period. One topic is presented as the primary focus of the sprint. This is covered in a bottom-up approach by one of the course instructors (”I Do”). This first presentation of content makes heavy use of debugging techniques, step-by-step execution, and other investigative approaches which provide explicit insight into how code translates to actions within the computer. Chunks of complete code are presented with specific ”hang-ups” being pointed out such as the difference between == (the comparison operator) and = (the assignment operator). Student questions about how code structures work and how new content ties to previous content is explored in depth by hand.

The second presentation of the same content, referred to as ”reinforcement” and occurring the week after first presentation, focuses on a top-down perspective by presenting problems which require application of the content. The content of reinforcement lectures varies in accordance with perceptions of understanding from students. When topics require more explicit explanations, reinforcement lectures may appear very similar to the content lectures from the previous week, but when students have grasped a topic to a greater degree, a situation which occurs with increasing frequency as the semester progresses, example problems are solved collaboratively with the instructor guiding solutions and pointing out potential problems as they come up (”We Do”). These reinforcement lectures also place additional focus on providing working demonstrations of programming concepts beyond the field of computer science. Exercises are undertaken collaboratively to explore topics in game theory, astronomy, and genetics among other domains.

The third week, which has students engaging with assessment content without explicit direction from instructors, provides yet another problem for students to solve, this time through prompted application of learned skills. This ”You do” section of the course has assignments which are referred to as ”quests,” an assessment with greater impact than a quiz but less than a test. **Quests, covering the same topic though presented as a different challenge, may be repeated by students during any subsequent reinforcement week to improve the grade received by a student.** This explicitly provides the chance for non-major students, which historically drop CS1 courses at greater rates than CS majors [7], to develop a mental model for computational thinking without having a punishment to their grade applied before they’re able to reach the level of understanding necessary for success in the course. Quests are divided into two segments, a knowledge evaluation which makes use of multiple-choice questions, Parson’s puzzles [21], and short-answer questions to assess students conceptualization of course content and a programming assignment which asks students to complete a partially finished program which is provided to them. Either segment may be re-attempted independently in subsequent weeks by students, and only their best scoring quest attempt is finalized in their overall course grade.

This approach seeks to optimize for the development of meaningful mental models within students as well as optimizing for the prompt correction of mistaken assumptions baked into those models. The assessments administered to students during the final week of the content cycle is reviewed by the team of instructors prior to presentation of new content the following week. Feedback is then

presented to all students in the course prior to new content being introduced to ”close the loop” and ensure students possess a meaningful, applicable mental model of the concepts introduced up to this point before moving on to more complex topics which rely upon a working mental framework to be most effectively introduced.

Agile Constructivist Pedagogy emphasizes quick iteration and feedback loops, mirroring agile ”sprints,” to let students actively build understanding with instructor guidance. Such an approach aligns with evidence that agile-inspired teaching can adapt to student needs in real-time, fostering positive learning experiences and outcomes

Content Sprints

The content presented in the course is broken up into five high-level categories. Each builds sequentially upon the content presented in the previous sprint, but some degree of isolation is still possible between the categories. The section on ”Advanced Collections and Files” certainly must come after the section on ”Collections of Data,” but exempting this, the segregation of content serves to keep students engaged and returning to the course content even when one topic presents a more difficult challenge to student mastery than another topic. The overall structure and sequencing is presented in Table 1.

Week #	Cycle	Concepts	Lab	Available Quests
1	C		1: Hello World	
2	R	Storing and Manipulating Data	2: Cash Register	
3	Q			**Quest A**
4	C		3: Collatz Numbers	
5	R	Controlling Flow and Process	4: RPS	*Retake Quest A*
6	Q			**Quest B**
7	C		5: toki pona Translator	
8	R	Collections of Data	6: Genetics Analysis	*Retake Quest A / B*
9	Q			**Quest C**
10	C		7: Movies I: Pre-Processing	
11	R	Files & More	Movies II: Structured Data	*Retake Quest A / B / C*
12	Q			**Quest D**
13	C		9: Movies III: Data Visualization	
14	R	Advanced Concepts	10: Yacht Dice	
15	Q			*Retake Quest A / B / C / D*
16	Q			**FINAL QUEST**

Table 1: The 16 week schedule for the CS1 course including lab major topics, lab names, and available quests and retakes. Labs are available on github [22].

The selection of appropriate content for CS1 courses has been extensively debated in computer science education research. This course adopts a data-first pedagogical framework, positioning computation as a tool for analyzing and modeling problems across multiple disciplines. By emphasizing data processing and analysis, this approach aligns with similar curriculum designs that prioritize practical data manipulation skills [13]. The focus on data-driven problem-solving provides students with immediately applicable skills while building foundational programming concepts through concrete examples.

Lecture Structures

The content lectures begin with a brief retrospective on the last sprint of content. This enabled students to adjust their internal models of how CS works to have a better working model leading into the next sprint. Instruction is conducted by the course professor in a hands-on, demonstration of the concept. As an example, in the case of introducing variables, students are shown 1) an IDE with a few assignment statements in a code file, 2) a debugger which shows the currently allocated variables and values, and 3) a step-by-step visualizer which allows for the program to be stepped through and dissected during execution.

Reinforcement lectures presented take the concept manually demonstrated from the previous week and embed it in a number of example programs which are analyzed interactively with students. Continuing the variables example from the previous paragraph, multiple programs are developed which showcase using variables for embedding other domain principles into computing problems. The ”ideal gas law,” a foundational formula learned in chemistry courses, showcases the power for quickly performing manual calculations with programming solutions and how variables enable this type of computation.

The level of independence provided during quest weeks serves students which have full confidence very well; and students which are intrinsically motivated, most commonly the computer science majors [7], need no further direction and are able to complete their work independently. Instructor attention is therefore diverted to students which are motivated to succeed in class based on grade requirements, but are not internally invested in computing. These students are frequently the non-computer science majors and these students are major contributors to course DFW counts [11]. Instructors then open and provide small group instruction for students during this time period.

Lab Timing

Lab sessions were held weekly with a maximum group size of thirty students, facilitated by a graduate teaching assistant, and assisted by one or two undergraduate teaching assistants. Following the agile modality presented in this work, the sprint structure was also adapted to the lab setting. Two weeks were content focused, with the third week being an open work period in which full support is available while students are given the option of attending to have a dedicated space, with support, for the completion of the quests and other outside assignments. During lab sections, students are presented with the lab tasks and work is facilitated by the teaching assistants in the room.

The first week of each sprint has a simpler version of the problem space being explored in lecture. The second week extends these problems by having a more abstract application of the sprint topic. One example of this is the move from the *Collatz number lab* (which asks students to manually check whether a number satisfies the Collatz conjecture) to the *RPS lab* (which asks students to implement the logic for an application which plays the game Rock, Paper, Scissors). Each lab assess students' ability to create control flow structures; however the number and complexity of said structures is far lower in the *Collatz number lab*. The former has only a single loop and a single if statement while the latter has multiple of each.

Each of the elements within this novel instructional model may itself be utilized in existing CS1 courses. The novelty comes in the way these approaches are integrated together and in the way the course adapts current best practices from the field of software engineering to implicitly expose students to these ideas. The approach utilizes this agile method to more efficiently respond to student struggles and to improve student confidence in their development of computational thinking throughout the course.

Student Outcomes

This work employs both qualitative and quantitative methods to evaluate the pedagogical approach. Qualitative data was obtained through voluntary, anonymous student evaluations during the implementation period of the agile methodology. Student perception of the learning environment serves as a critical indicator for necessary adjustments, independent of grade performance trends. The quantitative analysis utilized existing University records and assignment grades collected through standard course evaluation procedures. All data analysis was performed on aggregated, anonymized information that was already being collected as part of normal course operations.

The qualitative assessment draws from student feedback collected in the Fall 2022 and Spring 2023 course sections. The anonymous nature of the departmental surveys precludes the inclusion of demographic or performance-related metadata such as student majors or final grades. The Electrical Engineering and Computer Science department's survey instrument focuses on three key areas: the effectiveness of assignments in facilitating learning objectives, student engagement levels throughout the course, and students' self-reported correlation between their engagement and academic success.

Student answers to the survey were given in the form of open response text questions. The surveys on which the qualitative analysis is based were offered to the entire class across two semesters. The fall semester had 77 respondents out of 129 students (a 60% response rate) and the spring semester had 34 respondents out of 90 students (a 38% response rate).

The quantitative analysis utilizes institutional data to examine student performance trends across multiple course sections. Key variables include academic performance metrics (midterm and final grades), student demographics (sex and class standing), and program enrollment status. For analytical purposes, grades were dichotomized into pass/fail outcomes, and student majors were classified as either computer science or non-computer science, with undeclared students categorized in the latter group.

Qualitative Analysis

Instructor perception of the course's performance is undoubtedly important; it is a guiding force behind nearly all recommendations for improvements to CS1 course presentation. However, students' conceptions of the course and their learning throughout the experience are perhaps equally valuable. If students are able to see the value in what they were taught, as well as perceiving value in the *way* they were taught, it lends credence to the value of this novel modality. Comments

from the students within the survey administered at the end of the course provide this exact feedback. One student says: *"The way [the course was designed] was absolutely genius. The only stress I felt was motivational stress because I had so many opportunities to improve my grade and understanding with [quest] retakes."* While addressing student motivation is a complicated issue, moving to a level of instruction where motivation is the primary hurdle for students to overcome represents a strong result for breaking down many of the barriers encountered by students entering CS1 courses [10, 23].

Other feedback from students lacked the direct commentary on the structure of the course. Instead, they placed focus on the value of the multiple instructors and their differing approaches to content. Specifically, the utilization of "reinforcement lectures" were identified by multiple students as being helpful for developing student models of computation. Some comments received stated:

- "The lectures led [during reinforcement weeks] were extremely easy to follow and down to earth;"
- "[The instructor] provided many examples of how the information from this course can be used in real life in the CS field and outside of it. I used the knowledge from this class in another class because it made the content easier to comprehend in that class;" , and
- "[the instructor] made sure that we understood what was happening [when things went wrong] and how to do it better next time."

Student feedback demonstrated strong satisfaction with both the course content and instructional approach. Survey responses emphasized the practical value of the material, with multiple students specifically noting how the knowledge could be applied beyond the course. Students also responded favorably to their active role in the content delivery process, indicating that this participatory approach enhanced their learning experience

Quantitative Analysis

Positive student perception of a course is a welcome opinion, but unless this perception is paired with a similar improvement in students' understanding of course material, it is unlikely to translate to a deep understanding of computer science topics. Thus, it is valuable to also consider student success rates in the course and place those rates in the larger context of the historic success rates at the same institution. When we discuss "success" in this context, we refer to a commonly used metric of students who received passing grades (A, B, or C which correspond to 70%+ for this course) in comparison with students who received a non-passing grade, including students who withdraw from the course.

The success of CS1 courses is a topic of much focus in the CSE field, as without students who pass the introductory course and develop mastery over introductory content, the totality of CS remains unavailable. A range of pass rates has been reported in the literature, with the most common values being in the upper seventy percent of students passing CS1 courses [24–26]. Specialized classes, such as media computational courses have been introduced over the previous decades, with the explicit purpose of being more relatable to non-major students. These courses, while

being successful and even achieving pass rates of approximately 90%, may been criticized for not necessarily achieving the same student understanding of CS as a traditional CS1 course.

Semester *Students*	Major Students *(n=72)*	Non-Major Students *(n=147)*	Total *(n=219)*
Fall 2022	98.15	91.42	94.35
Spring 2023	88.88	90.41	90.11
AY 22-23	95.83	90.90	92.56

Table 2: Pass rates for major and non-major students across two semesters of the new instructional method.

This course, using the instructional method presented in this work, has been offered for two semesters consecutively with similar passing rates in both semesters. This generalization holds for both major and non-major students, with non-majors having a higher passing rate than declared computer science students in total. Major specific pass rates were more variable than non-majors with 98.15% of students passing in the Fall 22 semester (53 of 54) and 88.88% of students passing in the Spring 23 semester (16 of 18). Non-major pass-rates had more consistency with Fall 22 seeing 91.42% of students passing (71 of 82) while 90.41% of non-major students passed in Spring 23 (66 of 73). See Table 2 for semester-by-semester combined pass rates. The total pass rate across both semesters and for both major and non-major students was 92.56%, a significantly higher rate than traditional CS1 courses reported in the literature, and only a few percentage points higher than specialized, media computation classes which are seeing interests rising in response to the proliferation of computing courses for non-major audiences.

Discussion

Much of the impact of this novel modality is available either anecdotally from student testimony or is represented in the pass-fail performance of students. With that said, the instructor perception of a classroom can be a useful barometer for evaluating whether more rigorous study of a modality would be beneficial, or even if a modality should be abandoned altogether. One of the greatest strengths seen in this modality is the flexibility surrounding the timing of student learning. In a traditional CS1 course if a student hasn't been able to learn the content which is being assessed, there is often no recourse for them to recover their grade if they later understand a topic. This problem, which exacerbates the problems already identified by students as to why they are unsuccessful in CS1: lack of time, the unfamiliarity of the content, or even just the foreign nature of computational thinking to newcomers in the field, is directly alleviated by a flexible evaluation schedule such as the one utilized in this course.

While exam retakes are a well-established pedagogical practice, the agile approach innovates by implementing targeted content revision immediately following student evaluations. This immediacy allows for rapid correction of misconceptions and enhancement of student understanding. The post-evaluation instruction periods are structured to address both common challenges faced by the cohort and individual student difficulties.

A concrete example of this approach's effectiveness emerged in teaching array indexing concepts to

CS1 students. Python's half-open interval notation for array slicing typically presents a significant challenge. During the first semester implementation, students struggled considerably with this concept. However, the second cohort demonstrated markedly better comprehension of the same material, despite minimal changes to the content presentation.

Traditional course development often relies on end-of-semester reflection and subsequent modifications to improve content delivery. While instructors can provide clarification when they identify student misconceptions, the traditional approach often results in a delay of a month or longer between identifying and addressing these learning challenges. The agile instructional model significantly reduces this timeline, ensuring that any misconceptions are addressed within three weeks of the initial concept introduction.

This quick turnaround, coupled with a lack of pressure placed on students to learn something immediately and in the exact way it was presented, was identified by the teaching team as a major contributor to the improved student performances in this version of a CS1 course.

Limitations

Several limitations warrant consideration in analyzing this course's outcomes. The COVID-19 pandemic's impact on degree completion patterns presents a significant confounding factor. While enrollment numbers recovered to pre-pandemic levels during the implementation of this course, the pandemic may have influenced student willingness to engage with interdisciplinary content, particularly during online and asynchronous instruction periods.

These effects extend beyond what enrollment data alone can capture. Additionally, while the student feedback was collected anonymously through institutional channels rather than course staff, its anecdotal nature limits the strength of our conclusions. To validate the effectiveness of the presented teaching methods, future research would benefit from more rigorous empirical assessment, including pre/post evaluations and longitudinal studies tracking student performance in subsequent coursework.

Conclusion

This work introduces a novel mode for computer science education which directly parallels professional standards from the field of computer science. Students achieved high levels of success in the course, and non-major retention was strong. Multiple students directly credited the modality with their strong understanding of computer science after a CS1 course taught at a medium-sized, land-grant, state institution in the United States of America. A case is made for this instructional method and a number of the atypical aspects which were identified as contributing to the success of the method are explored in detail.

The transferability of this method to other institutions is an important future question remaining after this work. While success was found in these offerings of the course, more data must be gathered to determine the efficacy of this mode within a multitude of educational environments. A common concern with trying novel methods of instruction are the potential downsides to student understanding. No instructor wants to take the risk of putting students at a disadvantage because

of experimentation in the classroom; however, this method provides integrated tools to address struggles in student understanding, allowing an instructor to navigate the difficulty surrounding novel course implementations while still placing student success and student learning at the center of the course.

Acknowledgments

The authors would like to acknowledge and thank the undergraduate TA team who enabled the success of this course: Jarek Brown, Francis Korsah, Ilona van der Linden, Alicia Thoney, and Marc Wodahl. Their dedication and commitment to students enabled the student success achieved in this class. The authors would also like to acknowledge and individually thank Alicia Thoney's additional work as a supplemental instructor for the course, offering yet another view of course content for students to engage with.

References

[1] M. Borowczak, "Communication in stem education: A non-intrusive method for assessment & k20 educator feedback," *Problems of Education in the 21st Century*, vol. 65, no. 1, pp. 18–27, 2015.

[2] M. Ben-Ari, "Constructivism in computer science education," *ACM SIGCSE Bulletin*, vol. 30, no. 1, pp. 257–261, 1998. [Online]. Available: https://dl.acm.org/doi/10.1145/274790.274308

[3] C. Szabo, N. Falkner, A. Petersen, H. Bort, K. Cunningham, P. Donaldson, A. Hellas, J. Robinson, and J. Sheard, "Review and use of learning theories within computer science education research: Primer for researchers and practitioners," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '19. Association for Computing Machinery, 2019, pp. 89–109. [Online]. Available: https://dl.acm.org/doi/10.1145/3344429.3372504

[4] P. A. Kirschner, J. Sweller, and R. E. Clark, "Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching," *Educational psychologist*, vol. 41, no. 2, pp. 75–86, 2006.

[5] C. E. Hmelo-Silver, R. G. Duncan, and C. A. Chinn, "Scaffolding and achievement in problem-based and inquiry learning: a response to kirschner, sweller, and," *Educational psychologist*, vol. 42, no. 2, pp. 99–107, 2007.

[6] A. C. Burrows and M. Borowczak, "Hardening freshman engineering student soft skills," in *2017 FYEE Conference*, 2017.

[7] K. Umapathy, A. D. Ritzhaupt, and Z. Xu, "College students' conceptions of learning of and approaches to learning computer science," *Journal of Educational Computing Research*, vol. 58, no. 3, pp. 662–686, 2020. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0735633119872659

[8] A. Kaur and K. K. Chahal, "Exploring Personality and Learning Motivation Influences on Students' Computational Thinking Skills in Introductory Programming Courses," *Journal of Science Education and Technology*, May 2023. [Online]. Available: https://doi.org/10.1007/s10956-023-10052-1

[9] Z. Xu, A. D. Ritzhaupt, K. Umapathy, Y. Ning, and C.-C. Tsai, "Exploring college students' conceptions of learning computer science: a draw-a-picture technique study," *Computer Science Education*, vol. 31, no. 1, pp. 60–82, 2021. [Online]. Available: https://doi.org/10.1080/08993408.2020.1783155

[10] A. Forte and M. Guzdial, "Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 248–253, 2005, conference Name: IEEE Transactions on Education.

[11] M. Guzdial and A. Forte, "Design process for a non-majors computing course," *ACM SIGCSE Bulletin*, vol. 37, no. 1, pp. 361–365, 2005. [Online]. Available: https://dl.acm.org/doi/10.1145/1047124.1047468

[12] A. Radenski, ""python first": a lab-based digital introduction to computer science," *ACM SIGCSE Bulletin*, vol. 38, no. 3, pp. 197–201, 2006. [Online]. Available: https://dl.acm.org/doi/10.1145/1140123.1140177

[13] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and B. Tribelhorn, "A data programming cs1 course," in *Proceedings of the 46th ACM technical symposium on computer science education*, 2015, pp. 150–155.

[14] A. C. Burrows, M. Borowczak, T. F. Slater, and C. Haynes, "Teaching computer science & engineering through robotics: Science & art form," *Problems of Education in the 21st Century*, vol. 47, p. 6, 2012.

[15] H. M. Walker, "A lab-based approach for introductory computing that emphasizes collaboration," in *Computer Science Education Research Conference*, ser. CSERC '11. Heerlen, NLD: Open Universiteit, Heerlen, Apr. 2011, pp. 21–31.

[16] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik, "Improving the CS1 experience with pair programming," *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 359–362, Jan. 2003. [Online]. Available: https://dl.acm.org/doi/10.1145/792548.612006

[17] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[18] T. Dyba and T. Dingsoyr, "What do we know about agile software development?" *IEEE software*, vol. 26, no. 5, pp. 6–9, 2009.

[19] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review," *Information and software technology*, vol. 61, pp. 163–181, 2015.

[20] A. Alami, O. Krancher, and M. Paasivaara, "The journey to technical excellence in agile software development," *Information and Software Technology*, vol. 150, p. 106959, 2022.

[21] D. Parsons and P. Haden, "Parson's programming puzzles: a fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, 2006, pp. 157–163.

[22] A. Robins, "Cosc 1010 intro labs," 2023, accessed on April 10, 2025. [Online]. Available: https://github.com/cxedhub/cosc-1010-intro-labs

[23] P. Kinnunen and L. Malmi, "Why students drop out cs1 course?" in *Proceedings of the second international workshop on Computing education research*, 2006, pp. 97–108.

[24] A. Mohamed, "Teaching highly mixed-ability cs1 classes: A proposed approach," *Education and Information Technologies*, vol. 27, no. 1, pp. 961–978, 2022.

[25] B. Simon, P. Kinnunen, L. Porter, and D. Zazkis, "Experience report: Cs1 for majors with media computation," in *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 2010, pp. 214–218.

[26] D. Horton and M. Craig, "Drop, fail, pass, continue: Persistence in cs1 and beyond in traditional and inverted delivery," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 235–240.