Increase Performance in CS 2 via a Spiral Redesign of CS 1

Albert Lionelle
Colorado State University
Albert.Lionelle@colostate.edu

Sudipto Ghosh
Colorado State University
sudipto.ghosh@colostate.edu

Benjamin Say
Metropolitan Community College
bcsay@mccneb.edu

J. Ross Beveridge
Colorado State University
ross@cs.colostate.edu

ABSTRACT

Computer Science (CS 1) offerings in most universities tend to be notoriously difficult. Over the past 60 years about a third of the students either fail or drop out of the course. Past research has focused on improving teaching methods through small changes without changing the overall course structure.

The premise of our research is that restructuring the CS 1 course using a Spiral pedagogy based on principles for improving memory and recall can help students learn the information and retain it for future courses. Using the principles of Spacing, Interleaving, Elaboration, Practiced Retrieval, and Reflection, we fundamentally redesigned CS 1 with a complete reordering of topics. We evaluated the newly designed CS 1 by comparing the students with those coming from a traditional offering in terms of (1) CS 1 performance, (2) retention of students between CS 1 and 2, and (3) CS 2 performance.

We demonstrate that the Spiral method helped students outperform those who learn via the traditional method by 9% on final exam scores in CS 1. Retention is increased between CS 1 and CS 2 with a 19.2% increase for women, and 9.2% overall. Furthermore, students continue to do better in CS 2 with increased grades across all assessments and show a 15% increase in passing grades.

We provide a framework for the Spiral methodology so that others may replicate the design. Our results lead us to consider evaluating and improving the underlying methodology with which we teach Computer Science.

CCS CONCEPTS

• **Social and professional topics** → *Computational thinking*; *Computer science education.*

KEYWORDS

Computing education, CS1, CS2, course design, pedagogy

ACM Reference Format:
Albert Lionelle, Sudipto Ghosh, Benjamin Say, and J. Ross Beveridge. 2022. Increase Performance in CS 2 via a Spiral Redesign of CS 1. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499339

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

ACM ISBN 978-1-4503-9070-5/22/03…$15.00
https://doi.org/10.1145/3478431.3499339

1 INTRODUCTION

Computer Science is often considered hard, with the data supporting that nearly one-third of all students who take Computer Science 1 (CS 1) either fail or drop out within the first year [3, 44]. Most CS 1 courses are teaching students not only how to read and write a new language, but also a much greater skill, Computational Thinking. Focusing on four broad cognitive skills: Abstraction, Decomposition, Pattern Recognition, and Algorithmic Design; Computational Thinking is a different way to look at problems, especially in data driven societies where our problems are very large with daunting amounts of data and issues surrounding each problem [4, 10].

In spite of this difficulty, there are many students who are able to rise to the challenge of their CS 1 course. Various reports suggest that student self-efficacy is a major contributing factor to success in both college [33, 34] and Computer Science [23]. There have been multiple studies of improving student self-efficacy in Computer Science by peer instruction, paired programming, and more [2, 32, 43, 48]. These successful students are able to handle the cognitive load of Computer Science; remembering concepts with cumulative expectations and having fundamental skills for studying those concepts. Students with high self-efficacy tend to have a mastery or growth mindset[13]. Unfortunately, these skills are not often considered part of the standard CS 1 curriculum, and arguably are not often directly taught in most college settings. Simply, students manage to succeed in spite of their introductory course in Computer Science.

It is our premise that we can do better at teaching if we design CS 1 in a manner that naturally promotes these skills and the ability to actively recall information in a subject that is always cumulative. Based on a similar premise, Lionelle et al. redesigned CS 0, around proven skills for recall and memory as a means to improve recall while introducing more topics in the standard CS 0 sequence to meet university standards. They termed this design the *Spiral Design* due to the intentional quick introduction to topics, and the return of topics with more in-depth chunks at each iteration. They were able to show increased student retention between courses, and increased performance in the following CS course [22]. This result was somewhat unexpected, and an earlier study by Wilcox and Lionelle showed that students often "catch up" by their second semester in programming [46], meaning students with previous programming experience start out performing better compared to other students, but then most students perform equally by the end of their second semester. These results led to the question,

could the same Spiral design be adapted to a CS 1 course without adding additional topics? Furthermore, would such a design enable students to continue to show improved performance in CS 2?

Following the same idea, we redesigned CS 1 using the Spiral Design principles as the foundation. The redesign caused a complete reordering of topics, new assignments, new approaches to labs, and most importantly a change in the underlining pedagogy in how to teach the material, but did not include any additional topics. We termed this redesign Gold. At the same time, we ran a section using the traditional CS 1 course design, termed Green, with a heavy emphasis on peer instruction, collaboration, many small programs, and other well researched topics. The goal was to take a traditional course designed around all the current well-documented best-practices, and see if the students who learn via the Spiral design outperform the traditional course. Furthermore, we then had all the students take the same CS 2: Data Structures together to gauge performance not only at the end of the first CS 1 course, but to see how they continued to perform in the following course.

We seek to answer three research questions:

- Q1: Do CS 1 students taught using the Spiral methodology outperform students taught in the traditional manner?
- Q2: Given CS 2 is often the great equalizer in student performance no matter their programming background, do students taught using the Spiral methodology in CS 1, continue to outperform other students in CS 2?
- Q3: As changes should not hurt inclusion and retention of students between courses, do students taught using the Spiral method continue onto CS 2 showing at least the same, if not better retention than the traditional?

In this paper we review the fundamental principles of the Spiral design and their current foundation in Computer Science research. We show the results of our analyses between Spiral design (Gold) and the Traditional best-practices design (Green). The results encourage us to reevaluate the common ordering of topics if we are really going to improve how students learn Computer Science.

2 RELATED WORK

Wilcox and Lionelle study the relationship between students with prior programming experience and students without prior programming experience in CS 1, who then continue onto CS 2. They find that while students with prior-programming experience do better than those without in CS 1, and have noticeably higher retention, students without prior-programming experience catch up by the end of CS 2 performing equally with students who had previous experience before CS 1 [46]. Their study provides a baseline of what to expect of students going from CS 1 to CS 2, and the question of if a different pedagogy can change this outcome.

Lionelle et al. restructured CS 0 using a Spiral methodology in an effort to increase recall while reducing the amount of time on each topic. It was later found the design was based on principles of memory and recall, both student performance and retention increased. Furthermore, students in the Spiral class continued to outperform students who were taught via the traditional manner in the following CS 1 course [22]. The principles of recall are detailed in the following, with a focus on applications in Computer Science.

There is a long history with memory research dating back to the Seminal 1885 work of Ebbinghaus [11, 36, 39]. Recent studies in memory research have focused on developing practices that can be applied to real-world educational problems. Most notably, if a student is in a situation that promotes memory and recall, the more efficiently they learn and retain new material. This learning and maintaining should not be confused with simple memorization, but instead the overall aspect in which they process information. Schwartz et al. present four broad principles of memory improvement: process material actively, practiced retrieval, distributed practice, and use meta-memory [39], while Roediger et al. presents five themes: Spacing, Interleaving, Elaboration, Practiced Recall/Retrieval, and Reflection [36]. Although Schwartz et al. and Roediger et al. utilize different themes for improving memory efficiency, which in their studies directly lead towards student self-efficacy, there is noticeable overlap with only slight differences in the naming schemes: active learning maps to elaboration, practiced retrieval maps directly, distributed practice is a combination of spacing and interleaving, and meta-memory has reflection and metacognition at its center. Within Computer Science, Elaboration and Reflection have been extensively studied, but Spacing, Interleaving, and Practiced Retrieval have been largely ignored.

Elaboration is asking questions about the topics, and seeking answers to those questions. Peer instruction [9, 29, 30, 40, 41, 48] and paired programming [5, 14, 26, 47] are both methods of active learning. Kirkpatrick et al. used RSQC^2 and group work integrated into an operating systems course, and found improved student outcomes and readiness to participate. These initial results indicate improved self-efficacy [16].

Prather et al. review a number of instructional practices, and the dominant theme is reflection [31]. Reflection or self-explanation is known to elicit improved recall and understanding [7]. The benefits of reflection after exams and assignments has been extensively studied in Computer Science Education, with noticeable benefits to student performance [8, 12, 24, 25, 27, 28, 31]. Kann et al. confirmed that going beyond reflections with discussions about those reflections, improves both their reflections and their future coding activities [15]. Overall, there are many benefits to reflection, and while not the silver-bullet [42], teaching and having students reflect is a very strong tool used to promote meta-memory, metacognition, and self-regulation for students.

Practiced retrieval, better known as testing or practiced recall, is often misunderstood as a means of evaluation of progress and knowledge in an area. Roediger provides an overview of ten different benefits of testing [35] in which only one is about providing feedback to instructors. Furthermore, it is shown that more frequent low-stakes tests are better than the occasional midterm, encouraging long term memory retrieval [18, 19, 36]. In CS Education Research, there is very little on testing as a means to improve student performance, even though tests are extensively used within the field. Watson et al. analyzed 25 predictors for programming performance, and found the only predictor was self-efficacy. This leads Watson to encourage less testing [45]. However, this does not take into account the other nine benefits of testing, and the value of short frequent tests as compared to midterm style tests.

There is an extensive body of research on the benefits of the spacing effect, and Cepeda et al. provide an updated overview of

using spacing to enhance recall [6]. The spacing effect describes the benefit in memory retrieval of topics when those topics are studied with a gap between sessions. It has been found that spacing of material is one of the most powerful methods people can use to increase long term memory without increasing the total amount of time dedicated to studying [17].

In conjunction with spacing, students are encouraged to interleave topics by including a variation of topics in a study session. For example, a student who is learning math by mixing multiplication and division is interleaving the topics compared to a student who first learns multiplication and then division. It has been shown that interleaving of these topics improves recall in multiple STEM domains, including mathematics [37, 38]. It is also believed a strength of interleaving is that students are better at discerning and figuring out the problem, causing fewer discrimination errors [36].

There is very little on spacing or interleaving within Computer Science Education. Leppänen et al. studied pauses and spaces when students were working on programming projects. They found the longer the breaks students took, the worse they did. Their study only looked at spacing of work done on week long programming assignments [20]. This leaves an area open for debate on the benefits or disadvantages of spacing or interleaving within Computer Science.

3 CS 1 - DESIGN

This section highlights the two methodologies for teaching CS 1 with a focus on the order they present the topics. The "Green" design is based on the traditional layout suggested in most textbooks, and the "Gold" design is based on the Spiral teaching method.

3.1 Traditional Design: Green

Table 1 details the weekly schedule for the Green section, roughly in the order of the Liang book [21], albeit with ample use of an interactive textbook, Zybooks. New slides were generated for the course through a joint effort to bring in best practices of more active learning and peer instruction opportunities into the course, along with adding interesting topic discussions encouraging reflection.

The Green design adopted the many small programs approach [1] for labs and homework which encouraged multiple small programming assignments every week. Students were given the opportunity to take practice exams, making use of course question banks to improve their recall and knowledge. Thus, the Green design includes elaboration, some reflection, and some practiced retrieval.

3.2 Spiral Design: Gold

The Spiral Design is a redesign of CS 1 grounded on the five principles to improve memory recall presented by Roediger [36]. The belief is that while one cannot memorize how to code, recalling topics one has seen helps reconstruct the solution, while reducing the intimidation factor of not knowing where to start on the problem.

The process of redesigning the course and defining the best layout requires determining what to teach and when. Our decisions were based on teaching a computational mindset and keeping ease of grading in mind, which is why we started with methods early. We focused heavily on breaking every topic into smaller components. In the first pass, we would determine the smallest component we could teach about the topic and after 2-4 weeks we added another component of that topic. This allowed us to have students quickly using the concept, then with each pass they would find themselves exploring concepts at a deeper level.

Table 1: Schedule of topics for Fall 2020 Course Syllabus.

Week	Topic
Week 1	Computers; Program; Java
Week 2	Java Variables; Data Types; and Expressions
Week 3	Selections/Booleans/Conditionals/Switch Statements
Week 4	Mathematical Functions/Characters/Strings
Week 5	Control Loops
Week 6	Methods and Parameters
Week 7	Single-Dimensional Arrays
Week 8	Review Week
Week 9	Multi-Dimensional Arrays
Week 10	Inheritance
Week 11	Abstract Classes and Interfaces
Week 12	Review Week
Week 13	Exceptions and File Input/Output
Week 14	Recursion
Week 15	Sorting and Complexity
Week 16	Final Exam

The five principles and how they were implemented:

Spacing: The spiral design purposely defines "what not to teach, yet", so students can return to the topic, typically after 2-4 weeks.

Interleaving: While a traditional design teaches one major topic a week, the Spiral design purposely teaches multiple topics a week.

Practiced Recall: The Spiral design makes use of short weekly quizzes focused on reading code and concepts. Students were encouraged, but not required to retake these quizzes to study.

Elaboration: The Spiral design involves elaboration in two areas, (1) active learning approach of peer instruction and discussion, and (2) practical projects. The projects encourage students to view what they are learning in the larger context of industrial style applications. The projects are themed, which encourages students to think about the direction they would want to take in CS.

Reflection: Students reflect on what they are learning by submitting a paragraph reflection at the end of each of the five practical projects describing how they have progressed over the semester.

Unlike CS 0 presented by Lionelle et al. [22], no new topics were added. The hope is that by reordering the topics, we encourage recall and improved performance. The structure of the topics can be found in Table 2. It is notable that each week covers multiple differentiated topics to promote interleaving. Then every 2-4 weeks students intentionally return to a topic, but go more in depth.

The most noticeable difference between Tables 1 and 2 is that the Gold section was able to visit topics sooner because it did not go in depth at first. However, in the long term, the Gold section took a bit longer to get to Interfaces and Abstract Classes. All assignments had to be unique between Green and Gold because the order of topics drastically changed what previous knowledge could be assumed in the assignment. Course staff who are used to the traditional model had to be cognizant of the Spiral model while teaching a topic to make sure that they did not assume prior knowledge.

Table 2: Topic Schedule for CS1 Gold Section (Spiral Design)

Week	Topics
Week 1	Fundamentals (computers, variables, simple types)
Week 2	Objects, Methods, Strings, Conditionals
Week 3	Loops, Code Tracing
Week 4	Review and Exam 1
Week 5	Data Types, Classes, Common Classes (e.g. Math.java)
Week 6	Logical Operators, More Loops, More Methods
Week 7	Arrays, File Input
Week 8	Review and Exam 2
Week 9	File Output, Exceptions, More Classes
Week 10	More Branching, 2D Arrays, Introduction to Recursion
Week 11	Inheritance, ArrayList, UML
Week 12	Review and Exam 3
Week 13	Abstract Classes, Interfaces, Polymorphism
Week 14	Recursion, Sorting and Complexity, Collections
Week 15	Review Week
Week 16	Final Exam

The primary purpose of reordering topics and going more in depth later was to promote **spacing** and **interleaving** of the material. Instead of relying on students to study in an interleaved manner, assignments were given that covered the topic again. Thus forcing a spacing plus interleaving approach to their studies.

Students had one or two labs a week for three weeks. The fourth week of a unit was a review and catch-up week. The labs had to be designed from scratch due to the previous background knowledge assumption shown in the labs used for the Green section. For example, the branching lab had assumed they covered all three branching topics which were spread out across multiple weeks in the Spiral design. We focused on using a reordered version of the Green CS 1 Zybooks and added our own labs to accomplish this change. We were unable to use a physical book as no book currently matches this design. Due to the lack of support from the literature and textbooks, significant time was spent in reordering the topics in Zybooks and redesigning existing labs that assumed prior knowledge on topics.

Five times throughout a semester, the lab introduced a practical project. This project focused on having students write code within a much larger context. The goal was to provide more real world examples for students. They included both code examples for students to trace and harder problems for students to work on. Students were asked to write a reflection after each project.

The primary goal of the practical projects was to promote **elaboration** and **reflection**. Elaboration as students were able to see programming in the context of social good and larger applications. Reflection was a required assignment as part of the practicals. The Green section does not have a comparable assignment to the practicals, other than certain labs being intentionally harder than others.

Exams were given every four weeks in both the Green and Gold sections and were always cumulative, though exams 1-3 were redone since the topic ordering was different. When possible, exam questions were duplicated, even if asked at different points in the semester. Exam 4 was exactly the same between Green and Gold.

Overall, the two designs shared the following in common: the same instructor taught both sections for students without prior programming experience, the same TAs and lab rooms were used but at different times, same topics but different order, same focus on peer instruction and interesting topics, similar scaling of exams, interactive textbooks, active learning methods, peer instruction, ability to go back and resubmit assignments, and in the end - they both took the exact same final exam as we wanted to gauge them by the same expected cumulative outcomes.

Spiral Design CS 1 lecture sides, labs and practical write ups, and course syllabus are available at http://www.cs.colostate.edu/~cs164.

4 RESULTS

The experiment evaluates two different designs for CS 1. Students were grouped into their experimental groups based on which section they registered to take. Everyone who signed up for CS 1 With No-Prior Programming Experience Section 2 of the course, was signing up for the traditional method, which we term Green. Everyone who signed up for CS 1 With No-Prior Programming Experience Section 1 were taught using the Spiral methodology termed Gold. Students with Prior Programming experience had their own section, and were taught using the Spiral methodology and termed CS164-Gold. Required labs alternated throughout the day between Green and Gold.

The same instructor taught both Green and Gold sections. They had more familiarity with the Green section, being the designer of the Green section. They were willing to learn the Gold design to teach it simultaneously with the Green section.

For CS 2, all students were joined back together in a single course that was not redesigned. Only minor changes occurred that typically happen from semester to semester in course development. This allowed us to track students within the same course to see if Gold section students continued to perform better.

4.1 CS 1 Results

The different ordering of topics makes it difficult to evaluate student performance in CS 1 throughout the semester. Thus, we focus on the cumulative final exam and the number of students who chose to continue to CS 2. Success is achieved when (1) students taught with Spiral methodology (Gold) outperform students taught with the Traditional methodology (Green), and (2) Gold obtains at least the same, if not improved, retention compared to Green.

Table 3 shows the median final exam scores for each group. While the scores were low, arguably due to an overly difficult exam, everyone took the same exam, which was proctored online. Students were allowed to pick their own exam time slot throughout the week. The exam pulled from question banks randomly based on a question group, and each question group was matched up to a very similar question in difficulty, often only changing inputs or variables slightly. The Gold groups scored 9% higher than the Green group showing that those who learned via the Spiral method retained more for the final cumulative exam.

Using a t-test, we found a significant difference between Green and both Gold and CS164-Gold. Between Green and Gold, the p-value is .00025, and the result is significant at $p < .05$. Between Green and CS164-Gold, the p-value is .031565, and the result is significant at $p < .05$. However, between Gold and CS164-Gold, p-value is .257433, and the result is not significant at $p < .10$.

Table 3: CS 1 Final Exam Grades Between Sections. All Gold groups did about 9% better than their Green counterparts.

Teaching Methodology	Median Exam Grade	Total N	Men	Women
Green (Traditional)	56.4%	91	58	33
Gold (Spiral)	65.5%	86	66	20
CS164-Gold	65.7%	106	82	24

Gold and CS164-Gold only differing by .2% is a noticeable change from the 2018 paper by Wilcox and Lionelle, which showed by the end of CS 1 students with prior programming experience (CS 164) outperformed students without prior programming experience by 6% [46]. Our results show that with the Spiral method of teaching, students with no-prior programming experience managed to catch up to those with prior programming within the same semester.

The final grade distribution of the courses proved uninteresting because of the course curve applied in both sections by the instructor. While mostly unintentional, all three sections ended up with exactly the same number of passing grades as compared to failing grades, though slightly distributed differently between A, B, and C grades for the passing grades. It is interesting to note that the dropout rate was relatively the same at about 10%.

Focusing on retention, we looked at the students who could go onto CS 2, and chose to go onto CS 2. This gives an idea of which course setup is more motivating to students.

Table 4: Percentage of students who took CS 1 that chose to go into CS 2, split by reported gender. Tech students are ones with declared majors requiring CS 2 as part of their degree.

	Female(All)	Male(All)	Female(Tech)	Male(Tech)
Green	43.3%	65.3%	72.7%	79.2%
Gold	62.5%	66.1%	83.3%	91.7%
CS164-Gold	95.7%	79.7%	100.0%	92.9%

Table 4 shows the percentage of students who took CS 1 and chose to continue on to CS 2. There is not a notable difference for males between Gold and Green, but the very slight increase shows that the Spiral design did not hurt retention. However, for females, the Gold section proved to have a much higher retention rate than the Green, by nearly 19.2%. Overall, Gold had less women than Green starting out, but in the end had more go on. The high retention rate of women in CS 164 demonstrates a replication of the Wilcox and Lionelle paper, which also reported near 100% retention of women if they come in with prior-programming experience. If we only look at men and women who were enrolled in a technology major that requires CS 2 for their degree, Gold students have a higher rate of retention than Green students. For women, the Green had 72.7%, and Gold had 83.3% continue. For men, there is a notable difference. Green had 79.2% compared to Gold's 91.7%. Students who start out interested in technology and programming, remain interested when using the Spiral method of teaching, as compared to the traditional.

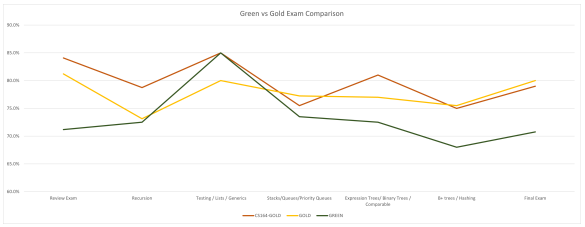

Figure 1: Module Exam Grades, Final exam for CS 2, broken up by CS 1 teaching methodology.

4.2 CS 2 Results

Since the overarching goal was information retention for a student completing a course, it is important to analyze the next course in the sequence. Which is CS 2: Data-structures. CS 2 had only one section where both Green and Gold students were grouped together. Students self-selected lab time slots when signing up. We tracked both Gold and Green section students throughout this course. While the primary comparison is between Gold and Green students, we continue to separate out CS164-Gold.

CS 2 students were given a review exam at the end of week 2. The exam was the CS 1 final exam, though due to the random question banks, the questions could be slightly different. Adding in the winter break, students had a 5-6 week break between exams. Furthermore, the first two weeks of CS 2 were meant to be a review of CS 1 with an emphasis on object oriented design. The Green group scored 63% as their median grade, and the Gold 73.2%, showing a 10% difference. Between the non-prior programming and prior programming Gold categories, CS164-Gold scored 76.1% for their median grade, which is 2.9% higher than Gold but the results are not significant with a p-value of .24 at $p < .05$. Both Gold and CS164-Gold had significantly higher results than Green with p-values of .03 and < .001 respectively at $p < .05$.

Figure 1 shows topic exams for the students grouped by their CS 1 teaching methodology. After the review exam, students took an exam on every topic introduced in the course. Green and Gold did not have a significant difference with Recursion or Testing. A significant difference at $p < .1$ started to appear between groups as seen in Table 5. By the end of the course, the Green group had a median score of 70.75% and the Gold a score of 80%, showing a 9.25% difference. The p-value is .04, and the result is significant at $p < .05$.

Table 5: Difference in exams between groups. Gold score relative to Green.

	Review	Recursion	Testing	Stacks
Green vs. Gold	+10% (.03)	+0.6% (.38)	-5.0% (.47)	+3.8% (.07)
Green vs. CS164-Gold	+12.9% (<.001)	+4.4% (.06)	+0.0% (.41)	+2.0% (.01)
	Trees	**B+ Trees**	**Final**	
Green vs. Gold	+4.5% (.06)	+7.5% (.07)	+9.25% (.04)	
Green vs. CS164-Gold	+8.5% (.01)	+7.0% (<.001)	+8.25% (.01)	

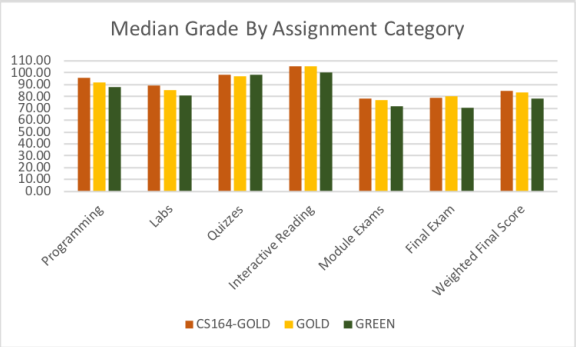

Figure 2: CS 2 grades broken down by average percent in each assignment category, compared across CS 1 groupings

Figure 2 shows the grades based on the assignment grouping for the course, in which the Gold students outperformed Green via assignment types. Furthermore, when looking at significance, a t-test showed a significant difference across Programming, Labs, Module Exams, Final Exam, and Final Score. Programming and labs being significantly higher would be an indicator that students improve both their programming ability in addition to test skills.

Overall, the Gold students are outperforming the Green students, and Green students are not "catching up" by the end of the course. Table 6 details the final grades for the CS 2 students. There is a noticeable difference between the groups, as Gold had 80% of students pass with a C or above, while Green had 65% pass with a C or above.

Table 6: CS 165 final grade distribution based on CS 1 Sections.

	CS164-GOLD		GOLD		GREEN	
A	22	31%	8	17%	21	23%
B	26	37%	20	43%	23	25%
C	16	23%	9	20%	16	17%
DFW	7	9%	9	20%	32	35%

5 THREATS TO VALIDITY

There were several variables that could not be controlled, such as time slots and instructors. For such variables, we attempted to bias the results towards the traditional methodology (Green section). For example, when selecting which section time students needed to sign up for, we set the more popular time slot, often with higher grades, as the Green group. When alternating labs, we started with Gold labs, so the dreaded 8 AM lab was a Gold section lab. When having one instructor teach both Green and Gold styles, we selected the instructor who designed the Green style, so they would be more comfortable teaching the Green style.

When processing the data, we opted to remove a group of students who received a directed intervention in CS 2 teaching the same study techniques. Students who scored poorly on the CS 2 review exam were encouraged to take a course covering the five techniques that are the foundation of the Spiral design as a means to study for CS 2. As the course was teaching the same techniques as the Spiral Design causing Green students to be learning Gold skill sets, we removed these 12 students from the CS 2 analysis for this paper, 64% of the students flagged for intervention were Green students. The hope is that with the Spiral design in CS 1, we may be able to reduce the need of the booster course. This is something to be addressed in a future study.

6 CONCLUSION

By redesigning CS 1 from the ground up using five methods based on psychology and memory recall research, we were able to show an improvement in student performance and retention in CS 1, and when they take the following course, CS 2. As part of the evaluation, we presented three major questions.

Q1: Do CS 1 students taught using the Spiral methodology outperform students taught in the traditional manner? Students taught by the Spiral (Gold) methodology showed a 9% increase over the Traditional methodology of one topic at a time.

Q2: Do students taught using the Spiral methodology in CS 1, continue to outperform other students in CS 2? Gold students showed a 10% increase on the same topics over the Green group. They continued to outperform throughout the whole class, showing 80% passing with a C or above as compared to 65% of Green students.

Q3: Do students taught using the Spiral method continue onto CS 2 showing at least the same, if not better retention than the traditional? Students in the Gold sections had noticeably increased retention; especially for women, which had a 19.2% increase in retention.

Our department found the time and effort spent in building the Spiral design of CS 1 to be worthwhile. We are now considering the implementation of the Spiral design in other courses. To start, implementing short weekly quizzes that can be repeated without penalty is a "cheap and easy" change, without having to build their own book. The key then becomes teaching students how to study using those quizzes. We find building the course around those study habits was overall an easier approach. For a later course, such as CS 2, one approach may be to have students first use each data structure to develop applications. After understanding the benefits, they can learn to code the data structures. The key point is to purposely create some time between topics.

Overall, the results show incredible promise of benefiting from a Spiral design for CS 1, and potentially other courses. The downside is such courses often require redesigns from the ground up, necessitating careful planning. We plan to develop a textbook specific to the Spiral design of CS 1 to save others the trouble of creating labs and assignments tailored to the Spiral model.

ACKNOWLEDGMENTS

Colorado State University and CSU Online supplied funds for course development. Thanks to the TAs, graduate students, and faculty, for making it possible to do a full course design in the short window.

IRB Approval PROTOCOL NUMBER: 20-10279H

REFERENCES

[1] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 585–591, New York, NY, USA, 2019. ACM.

[2] Ashish Amresh, Adam R. Carberry, and John Femiani. Evaluating the effectiveness of flipped classrooms for teaching CS1. In *Proceedings - Frontiers in Education Conference, FIE*, pages 733–735, 2013.

[3] Jens Bennedsen and Michael E Caspersen. Failure Rates in Introductory Programming. *SIGCSE Bull.*, 39(2):32–36, jun 2007.

[4] Fatih Kursat Cansu and Sibel Kilicarslan Cansu. An Overview of Computational Thinking. *International Journal of Computer Science Education in Schools*, 3(1):17–30, 2019.

[5] Jeffrey C Carver, Lisa Henderson, Lulu He, Julia Hodges, and Donna Reese. Increased retention of early computer science and software engineering students using pair programming. In *20th Conference on Software Engineering Education \& Training (CSEET'07)*, pages 115–122. IEEE, 2007.

[6] Nicholas J Cepeda, Harold Pashler, Edward Vul, John T Wixted, and Doug Rohrer. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological bulletin*, 132(3):354, 2006.

[7] Michelene T H Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian LaVancher. Eliciting self-explanations improves understanding. *Cognitive science*, 18(3):439–477, 1994.

[8] Michelle Craig, Diane Horton, Daniel Zingaro, and Danny Heap. Introducing and Evaluating Exam Wrappers in CS2. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 285–290, New York, NY, USA, 2016. ACM.

[9] Catherine H Crouch and Eric Mazur. Peer Instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977, 2001.

[10] Kevin Cummins. Five reasons why computational thinking is an essential tool for teachers and students. , 2020.

[11] Hermann Ebbinghaus. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155, 2013.

[12] Katrina Falkner, Rebecca Vivian, and Nickolas J G Falkner. Identifying Computer Science Self-Regulated Learning Strategies. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, ITiCSE '14, pages 291–296, New York, NY, USA, 2014. ACM.

[13] Gerry Geitz, Desirée Joosten Ten Brinke, and Paul A. Kirschner. Changing learning behaviour: Self-efficacy and goal orientation in PBL groups in higher education. *International Journal of Educational Research*, 75:146–158, jan 2016.

[14] Brian Hanks. Student performance in CS1 with distributed pair programming. *ACM SIGCSE Bulletin*, 37(3):316–320, 2005.

[15] Viggo Kann and Anna-Karin Karin Högfeldt. Effects of a program integrating course for students of computer science and engineering. In *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 510–515, New York, NY, USA, 2016. ACM.

[16] Michael S Kirkpatrick and Samantha Prins. Using the readiness assurance process and metacognition in an operating systems course. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, volume 2015-June, pages 183–188, New York, NY, USA, 2015. ACM.

[17] Nate Kornell. Optimising learning using flashcards: Spacing is more effective than cramming. *Applied Cognitive Psychology*, 23(9):1297–1317, 2009.

[18] Douglas P Larsen, Andrew C Butler, and Henry L Roediger III. Repeated testing improves long-term retention relative to repeated study: a randomised controlled trial. *Medical education*, 43(12):1174–1181, 2009.

[19] Frank C Leeming. The exam-a-day procedure improves performance in psychology classes. *Teaching of Psychology*, 29(3):210–212, 2002.

[20] Leo Leppänen, Juho Leinonen, and Arto Hellas. Pauses and spacing in learning to program. In *ACM International Conference Proceeding Series*, pages 41–50, 2016.

[21] Y. Daniel Liang. *Introduction to JAVA Programming*, volume 8. Pearson Education, 2011.

[22] Albert Lionelle, Josette Grinslad, and J Ross Beveridge. CS 0: Culture and Coding. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020.

[23] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, pages 211–220, New York, NY, USA, aug 2016. ACM, Inc.

[24] Murali Mani and Quamrul Mazumder. Incorporating metacognition into learning. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 53–58, 2013.

[25] Joshua Martin, Stephen H Edwards, and Clfford A Shaffer. The effects of procrastination interventions on programming project success. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 3–11, 2015.

[26] Charlie McDowell, Linda Werner, Heather E Bullock, and Julian Fernald. The impact of pair programming on student performance, perception and persistence. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 602–607. IEEE, 2003.

[27] Laurie Murphy and Josh Tenenberg. Do Computer Science Students Know What They Know? A Calibration Study of Data Structure Knowledge. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, pages 148–152, New York, NY, USA, 2005. ACM.

[28] Jennifer Parham, Leo Gugerty, and D E Stevenson. Empirical Evidence for the Existence and Uses of Metacognition in Computer Science Problem Solving. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 416–420, New York, NY, USA, 2010. ACM.

[29] Leo Porter, Cynthia Bailey Lee, and Beth Simon. Halving Fail Rates Using Peer Instruction: A Study of Four Computer Science Courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 177–182, New York, NY, USA, 2013. ACM.

[30] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. A Multi-Institutional Study of Peer Instruction in Introductory Computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 358–363, New York, NY, USA, 2016. ACM.

[31] James Prather, Brett A Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, pages 2–13, New York, NY, USA, 2020. ACM.

[32] Vennila Ramalingam and Susan Wiedenbeck. Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. *Journal of Educational Computing Research*, 19(4):367–381, dec 1998.

[33] Michelle Richardson, Charles Abraham, and Rod Bond. Psychological correlates of university students' academic performance: A systematic review and meta-analysis. *Psychological Bulletin*, 138(2):353–387, 2012.

[34] Steven B. Robbins, Huy Le, Daniel Davis, Kristy Lauver, Ronelle Langley, and Aaron Carlstrom. Do Psychosocial and Study Skill Factors Predict College Outcomes? A Meta-Analysis. *Psychological Bulletin*, 130(2):261–288, 2004.

[35] Henry L Roediger III, Adam L Putnam, and Megan A Smith. Ten benefits of testing and their applications to educational practice. In *Psychology of learning and motivation*, volume 55, pages 1–36. Elsevier, 2011.

[36] Henry L Roediger III and Mary A Pyc. Inexpensive techniques to improve education: Applying cognitive psychology to enhance educational practice. *Journal of Applied Research in Memory and Cognition*, 1(4):242–248, 2012.

[37] Doug Rohrer, Robert F Dedrick, Marissa K Hartwig, and Chi-Ngai Cheung. A randomized controlled trial of interleaved mathematics practice. *Journal of Educational Psychology*, 112(1):40, 2020.

[38] Doug Rohrer and Kelli Taylor. The shuffling of mathematics problems improves learning. *Instructional Science*, 35(6):481–498, 2007.

[39] Bennett L Schwartz, Lisa K Son, Nate Kornell, and Bridgid Finn. Four principles of memory improvement: A guide to improving learning efficiency. *IJCPS-International Journal of Creativity and Problem Solving*, 21(1):7, 2011.

[40] Beth Simon, Michael Kohanfars, Jeff Lee, Karen Tamayo, and Quintin Cutts. Experience Report: Peer Instruction in Introductory Computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 341–345, New York, NY, USA, 2010. ACM.

[41] Beth Simon, Julian Parris, and Jaime Spacco. How We Teach Impacts Student Learning: Peer Instruction vs. Lecture in CS0. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 41–46, New York, NY, USA, 2013. ACM.

[42] Ben Stephenson, Michelle Craig, Daniel Zingaro, Diane Horton, Danny Heap, and Elaine Huynh. Exam wrappers: Not a silver bullet. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pages 573–578, New York, NY, USA, 2017. ACM.

[43] Laura Toma and Jan Vahrenhold. Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs - A case study. In *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, volume 10, pages 1–10, New York, NY, USA, 2018. ACM.

[44] Christopher Watson and Frederick W B Li. Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, ITiCSE '14, pages 39–44, New York, NY, USA, 2014. ACM.

[45] Christopher Watson, Frederick W.B. B Li, and Jamie L. Godwin. No tests required: Comparing traditional and dynamic predictors of programming success. In *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 469–474, New York, New York, USA, 2014. ACM.

[46] Chris Wilcox and Albert Lionelle. Quantifying the benefits of prior programming experience in an introductory computer science course. *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018-Janua:80–85, 2018.

[47] Krissi Wood, Dale Parsons, Joy Gasson, and Patricia Haden. It's never too early: pair programming in CS1. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 13–21, 2013.

[48] Daniel Zingaro. Peer Instruction Contributes to Self-Efficacy in CS1. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 373–378, New York, New York, USA, 2014. ACM.