DISSERTATION

A SPIRAL DESIGN: REDESIGNING CS 1 BASED ON TECHNIQUES FOR MEMORY RECALL

Submitted by

Albert Lionelle

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2021

Doctoral Committee:

Advisor: J. Ross Beveridge

Sudipto Ghosh
Nathaniel Blanchard
James Folkestad

ABSTRACT

A SPIRAL DESIGN: REDESIGNING CS 1 BASED ON TECHNIQUES FOR MEMORY RECALL

Computer Science (CS 1) offerings in most universities tend to be notoriously difficult. Over the past 60 years about a third of 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 this 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, CS 1 was fundamentally redesigned with a complete reordering of topics. The new pedagogy was evaluated by comparing the students with those coming from a traditional offering in terms of (1) CS 1 performance, (2) CS 2 performance, and (3) retention of students between CS 1 and 2. Additionally, students were tracked on the individual outcome / topic level of their performance, and students filled out surveys measuring learning motivation and self-regulation.

The Spiral pedagogy helped students outperform those who learned via the traditional pedagogy by 9% on final exam scores in CS 1 with a significant difference. Furthermore, 23% of students taught using the Spiral pedagogy mastered greater than 90% of the outcomes, where those taught with the traditional method only mastered 5% of the learning outcomes. Students who were taught with the Spiral pedagogy showed a greatly increased interest in Computer Science by the end of the course, with women showing the greatest increase in interest towards Computer Science. Retention is increased between CS 1 and CS 2 with a 19.2% increase for women, and a 9.2% increase overall.

Five weeks later, students were given the same final exam by way of a review exam in CS 2. With the gap in time to forget, those taught with the Spiral pedagogy scored 10-12.5% higher

than their peers taught using the traditional method. The change in pedagogy showed an influence with Cohen's $d = .69$. Furthermore, students continued to do better in CS 2 with increased grades across all assessments, including programming capabilities. By the end of CS 2 only 65% of students who learned by the traditional method passed CS 2 with a C or above while students who learned via the Spiral pedagogy had 80% pass with a C or above.

The framework for the Spiral Design is presented along with implementation suggestions if others wish to duplicate the pedagogy for their course along with future research suggestions; including building a Spiral Curriculum to enhance performance across courses and interactive tools to act as a means of intervention using techniques proven to improve recall of past content. Overall the Spiral Design shows promising results as the next generation in course design for supporting student achievement and provides additional pathways for future research.

ACKNOWLEDGEMENTS

There are many people who helped with this process and who deserve acknowledgement. Dr. Bruce Draper was my advisor my first time in graduate school working towards my Master of Science degree. He was also the one that I would continue to visit over the years, and who convinced me to return to the university. Dr. Adele Howe had a profound influence on my life from when I worked with her many years ago. She was also another person whom I made sure to stay in contact with over the years, and I was always amazed by the passion she showed towards her students and teaching. She is always missed, but her influence is everywhere. Dr. Chris Wilcox, whom I walked into his office, and he suggested I write a CS education paper with him, and helped me get started on a journey of CS education research. Debbie Bartlett and Elisa Cundiff both of whom spent many hours hearing ideas I had for curriculum, often asking pointed questions to help me formulate those ideas more. Three undergraduate teaching assistants put more time into the development of the spiral model compared to anyone else: Mary Carrigan, Matthew Ernst, and Audrey Dorin. They were the first to be comfortable with my diving into this new territory, making labs for the course, and then teaching other TAs how to teach the new model. Benjamin Say spent his time as an instructor learning the model, so he could teach it in my place.

Above all, the person whom deserves the most acknowledgement is my life partner, wife, and love, Heather Lionelle. When I first finished my Master's Thesis, she was the one I thanked as Heather Codanti. Now over the years, we have two wonderful children, a life together, and a shared future. She has continued to listen to my ideas, to encourage me, and support me. She also is an amazing copy-editor, making multiple grammar edits and comments throughout this work. Her comments always challenged me to word things better. I can continue to go on about all that she does, but all in all, this wouldn't have happened without her support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 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 [5, 6]. 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 [7, 8]. Furthermore, the cognitive model that is required to understand computational thinking is considered one of the more challenging models for individuals to develop [9], it has even been argued that we ask the impossible of students within their first semester [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 [11, 12] and Computer Science [13]. There have been multiple studies on improving student self-efficacy in Computer Science by peer instruction, paired programming, and more [14–17]. 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 [18]. 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.

The premise of this research is 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 top-

ics in the standard CS 0 sequence to meet university standards. The design was termed the *Spiral Design* due to the intentional quick introduction to topics and the subsequent 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 [19]. This result was somewhat unexpected as an earlier study by Wilcox and Lionelle showed that students often "catch up" by their second semester in programming [3], 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, this research explores redesigning CS 1 using 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 learned via the Spiral Design would outperform the traditional course. Furthermore, we then had all the students take the same CS 2: Data Structures course 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.

Computer Science is hard, but learning doesn't have to be. While previous literature has focused on improving CS 1 with small changes to assignments and the way instructors teach, there hasn't been an in depth change to the very order in which we teach the topics. Instead, educators have come to the collective conclusion that topics follow a mostly set order, completely learning one topic before going onto the next topic. That order has never been questioned

and this research explores that question. Most notably, if a course is designed using the Spiral Method, can educators make a hard topic easier to learn for the students? Can educators improve performance by adopting this design?

In this research Chapter 2 explores the previous literature in the field with an emphasis on CS 1 design, best practices towards CS 1, including the importance of CS 0, cognitive studies around Computer Science, and the techniques of learning that are highlighted in the Spiral Design. Chapter 3 details the past work that has been done at CSU, including the redesign of CS 1 and the redesign of CS 0. Chapter 4 details the current design of CS 1, the proposed redesign using the Spiral Method, and the methods of evaluating and comparing the two courses. Three questions are presented in Chapter 4 that act as the foundation of the research:

- Q1: Given that going to spiral allowed us to add content without reducing learning, what happens when CS 1 is designed around the Spiral Model without adding content? Will student performance in the current course increase?

- Q2: Given CS 2 is often the great equalizer in student performance no matter their programming background, will students who learned via the Spiral Method in CS 1 outperform students in CS 2 who learned with the traditional methods in CS 1?

- Q3: Any modern changes should not hurt inclusion and retention of students between courses, does the Spiral Method at least encourage equal, if not improved retention of students who come from underrepresented backgrounds to continue onto CS 2 as compared to the traditional method?

After a year of testing, Chapter 5 details the results of three different CS 1 sections. The Traditional Design (Green) with No-Prior Programming Experience students, the Spiral Design (Gold) with No-Prior Programming Experience students, and the Spiral Design (CS164-Gold) with Prior Programming Experience students. The final exam along with outcome tracking, student surveys, and retention data acts as our primary data sources for analysis. Chapter 6 details the results of all students combined into a single CS 2 section in which the students are

tracked across all assignments throughout the semester. Additionally, students were given a review exam of CS 1 topics after the winter break in an effort to determine how much information was recalled using long term recall. Chapter 7 provides a discussion and analysis of the results across the year, highlighting key points and outcomes. There is also a discussion of how to implement the Spiral Design, tips and tricks uncovered over the past year, and provides a list of potential research opportunities that stemmed from this research.

The Spiral Design is a novel approach to teaching CS 1, and The results encourage us to reevaluate the common ordering of topics and assumptions we have if we are really going to improve how students learn Computer Science.

Chapter 2

Related Works

Introductory programming courses are notoriously demanding [20], with an extremely difficult cognitive model [9]. Programming discipline difficulty, course arrangement complexity, and limited student motivation are some of the reasons cited for high student dropout rates in introductory programming courses [21]. Most notably, it was estimated that of the two million students who took CS 1 in 1999, 650,000 of those students ended up failing their first course [5], roughly 33%. In a follow up 2014 study, it was shown that even after years of intervention, very little had changed. Across 161 courses, the average pass rate was 67%, making the failure or drop rate 33%. It is believed that with a low pass rate we see a reduction in student retention across student programs [6]. Given these high failure rates, coupled with the demand to see increased retention and graduation rates, course teaching pedagogy, student factors, and general curriculum must be critically evaluated. If we continually loose 1/3 of students who are interested in becoming CS majors, Computer Science will struggle. What factors go into improving student retention? What pedagogical techniques have been employed, and what exactly are we seeking to teach students?

This literature review examines those questions. Section 2.1 looks at the definition of introductory Computer Science courses, in particular CS 1. Section 2.2 looks at different ways performance has been measured along with common pedagogical techniques to improve either prediction of student performance, or directly to improve student performance. One such technique includes introducing a CS 0 course as a course to take before CS 1. Section 2.3 looks at the challenge of CS 1 as a metacognition and self-regulation issue in which interventions should not only target the what, but also focus on how students are learning to learn. Section 2.4 looks at four themes recommended by cognitive psychologists to improve learning and education.

2.1 Defining CS 1

Introductory programming courses are often called “Computer Science I” (CS 1), as an introduction to both programming and computational thinking. While the term is commonly used, a question remaining is if there are common learning outcomes most colleges and universities have adopted over the past sixty years. Many universities use a document produced by a joint effort between ACM and IEEE, the Ironman Draft of Computer Science Curricula, simply termed CS2013. The CS2013 document gives a fair amount of leeway between CS 1 style courses, simply outlining broad learning outcomes. They encourage a CS 1 course to pull from multiple knowledge areas (KA), with the admission that most CS 1 courses pull outcomes from KA Software Development Fundamentals (SDF) [22]. This freedom to adopt from any knowledge area dependant on institutional goals leads to great flexibility and leaves the question; "What are the most common goals across institutions?". Becker and Fitzpatrick explore this question by evaluating syllabi from 916 different institutions [1].

The primary question Becker and Fitzpatrick seek to answer is, “What exactly do we expect our introductory programming students to achieve?”. They generated a database of the syllabi from 916 institutions, and provided a website for the community to search the results. When looking at their primary question, they came up with six questions to explore.

- Q1 What percentage of CS1 courses have explicit learning outcomes?
- Q2 What concepts do explicit CS1 learning outcomes cover?
- Q3 How do These Concepts Align With CS2013?
- Q4 What do explicit CS1 learning outcomes look like?
- Q5 What is the current CS1 teaching language distribution?
- Q6 What are the most common computing terms found in CS1 syllabi?

It should be noted that learning outcomes and learning objectives are used in literature and interchangeably refer to explicit statements that define what a student should be able to accomplish upon completion of the course.

After reviewing the websites of 916 institutions, they ended up with 234 CS 1 syllabi from 207 institutions, only 65 percent of the institutions had explicit learning outcomes (Q1). Of these syllabi, the most common topics that were explicitly listed as a learning outcome (Q2) are:

- testing and debugging
- writing programs
- selection (conditional) statements
- problem solving including computational thinking
- arrays and list structures
- basic object oriented programming
- variables, data types, operators, declarations
- loops and repetition
- methods
- algorithm design
- classes and objects
- file I/O
- documentation
- recursion

- data structure at a distance last place with 19 percent of institutions listing them as explicit learning outcomes.

For Q3, Becker and Fitzpatrick focused on the Knowledge Area of Software Development Fundamentals (SDF), and the Knowledge Unit (KU), Fundamental Programming Concepts (FPC). All the concepts in the list above are covered via SDF except for OOP, classes and objects, which are covered in KA Programming Languages, specifically the KU of Object-Oriented programming [1].

In the Ironman Draft for CS2013, it is pointed out that SDF units are a prerequisite to studying most of Computer Science, and that is the targeted area for CS 1. Even though CS2013 recommended going outside of that area, very few courses pulled from the other KAs. The Knowledge Units in SDF are:

- Algorithms and Design
- Fundamental Programming Concepts
- Fundamental Data Structures
- Development Methods

A common mistake is assuming every topic needs to be covered in CS 1. In CS2013, they point out that while these are all Core 1 skills (requirements for any CS program), they can and should be covered across a variety of classes. For example, CS 1 may mention data structures, but it isn't until later in the curriculum (often CS 2 or CS 3) that students learn to implement data structures [22].

Looking at programming language distribution, Becker and Fitzpatrick find the following as presented in Table 2.1. They found, as expected, that Java was the most dominate language being taught, but they noted that is down from past investigations on programming language selection with Python gaining popularity as a primary CS language [1].

Table 2.1: Language distribution among CS courses as found by Becker and Fitzpatrick [1]. Table does not equal 100% as only languages with greater than 5% are shown.

Language	Percentage
Java	49%
Python	36%
C++	20%
C	5%

Programming language analysis has been researched over the years, with Pears et al. conducting surveys in 1998, 2001, 2003, and 2005. They found Java, C++, and C as the most dominate languages, with a focus in object oriented programming [23]. Raadt et al. found that language selection focused more on the type of programming with institutions leaning towards object oriented programming (81%), some procedural (10%), and functional programming (9%). They also showed that programming language was most often selected by use in industry / marketability of the student based on the languages they knew [24, 25].

Becker and Fitzpatrick's language comparison also matches the work done by Giangrande, with Java being the most popular language [26]. Siegfried et al. pointed at the noticeable rise of Python and also noted that while Python was on the rise for CS 1 courses, it was not used very often in CS courses. They came to the conclusion that Python was being used to teach fundamentals while Java (or another OOP language) was being used once students were more involved and needing to understand data structures [27]. As an example, this model is currently followed at Cal-Poly in which students take a Computer Science course for non-majors (CS 0) that is interest topic based, take CS 1 in Python, and then an OOP course in Java as their CS 2 course [4].

Overall, Becker and Fitzpatrick's work supported work done by others before them, and illustrated that for the most part schools follow the CS2013 recommendations with slight modifications based on the language they teach. A challenging question isn't just what people are teaching, but if schools are teaching the correct topics in the first semester for Computer Science Majors. With high drop out rates, and retention issues [5, 6, 21], are schools asking too

much of students or are they simply lacking the techniques and tools with which to reach the population that needs intervention?

Becker and Fitzpatrick based their work off of a challenge put to the Computer Science community by Luxton-Reilly. Their thought was that for the challenge to be satisfied, the community needed to confirm the current baseline of what is actually taught in CS 1. The challenge from Luxton-Reilly is:

> collect research-based evidence of what novice programmer can achieve at the end of a first programming course and use evidence to derive realistic expectations for achievement [10].

Luxton-Reilly point out that while countries such as UK [28], Australia [29], New Zealand [30], and Denmark [31] are requiring programming in K-12 education, Computer Science education at the university level is considered difficult with an acceptance of a high drop out rate [5, 6, 20]. Furthermore, the cognitive model required for Computer Science is one of the most difficult models to learn [9].

Luxton-Reilly believe the expectations of CS 1 students are too high, and that the community needs to seriously evaluate what they expect of students [10]. For this goal to be met, it is also important to look at the current measures used when comparing student performance, and also what interventions are used to improve student performance under the current model.

2.2 Measuring Student Performance

When measuring student performance, Alturki looks at mapping student performance across assignment types and the final grade of the course. In order to accomplish this task, they provide a detailed survey of the different ways in which student performance have been measured over the years, and some factors which have been examined that affect student performance [2].

2.2.1 Student Background and Performance

Student background plays a role in CS 1 course performance. In most CS departments, there is a discussion about mathematical maturity and its relationship to programming. An often cited 1983 paper by Konvalina et al. researched the affects of math proficiency on student performance. They found three major factors affecting the withdrawal rate of students: high school performance, student age, and amount of math the student took before taking the programming course. Based on their results, they determined either a math placement test, additional college level math courses, or a CS 0 style course that was open to all students would be ideal before accepting students into the programming course. The reason presented for the barriers was to deal with increased growth while having limited resources to help students who may just withdraw anyway. Once they implemented the controls on CS 1, they reduced their withdrawal rate from 40 to 23 percent [32]. This model is often followed by universities today, including Colorado State University which uses either math classes or a CS 0 course as the prerequisite to their CS 1 courses. Related to mathematical performance, problem-solving also plays a factor in student performance, as does the spoken language of the course compared to the students' native language. For purposes of measuring problem solving capability, Pillay & Jugoo looked at previous mathematical experience and other problem solving based courses, which were CS 0 style courses that taught problem solving. They also found gender was not an influence on performance. [33].

2.2.2 CS 0: Improving Performance and Retention

There is no single definition of CS 0 in literature, and it is often assumed to be a course that is much easier, with fewer programming requirements than a CS 1 course. The benefits of a CS 0 course as a means to improve student performance and retention in CS 1 have been studied in depth, with one of the earlier works dating to 1984 by Campbell. In 1981, the University of Maryland introduced a two-credit introductory programming course. Upon completion they were allowed to take the CS 1 course. They noted that 46% of students taking CS 1 did not have

programming experience, yet those who did have programming experience made up 70% of the A and B grades in the course. The failure rate, measured by both dropouts and failed grades, was between 30-35%, and approximately 10-12% of students retook their CS 1 course every semester. To reduce these issues, they had all Computer Science majors take the preliminary two-credit course before taking the CS 1 course in the Spring. They also allowed students who started in the CS 1 course to transfer back to CS 0, if they did it before the first exam. The CS 0 course covered within 15 weeks, what the CS 1 course covered in their first five weeks and at the same time had lectures focused on how to problem solve. They showed a significant increase of grades in CS 1 for the students who took CS 0. However, they noted there was a large number of students who took CS 0 who chose to not continue onto CS 1. So while their conclusions shows a benefit of CS 0, they wonder if that was due to early filtering of students [34].

In contrast to setting up math prerequisites or even required preliminary courses, Towson University, opted to have a preliminary exam. Based on the results of the exam, they would move students to a CS 0 course or they would remain in CS 1. Similarly, they found the placement test to be a good predictor of final grades in the CS 1 course, though they reported getting the right placement test was challenging. They made changes to it over five years; increasing its prediction accuracy. The exam focused on pseudo code, and solving programming problems. Then using the exam, they were able to group students into those who needed CS 0, and those who would be fine without it. Those who took CS 0, ended up performing better than their predicted grade in CS 1, which helped increase the overall grades of CS 1 students [35]. While this helps provide both a measure of student performance, and improves CS 1 grades, it does not handle the issue presented by Campbell of the high lack of retention between CS 0 and CS 1. Campbell also noticed, this had a particular affect on gender, with fewer women continuing in the program even though they had strong CS 0 grades [34].

2.2.3 Non-Academic Factors Affecting Grades

ACT[1] found in a 2004 study, that non-academic factors had a strong correlation with student success: the academic self-confidence, academic goals, institutional commitment, social support, certain contextual influences (institutional selectivity and financial support), and social involvement all having a positive relationship to retention. Self-confidence and goals had the highest correlation with college GPA as non-academic factors. [36].

Directly related to Computer Science, Wilson showed that comfort level had the highest correlation, much higher than math background. Comfort level was measured based on the likelihood of asking questions given certain situations, and perceived difficulty and understanding of assignments and concepts in class [37]. Alturki pointed out that while there is a correlation, there is an unknown relationship if that confidence is directly related to grades or the fact they are confident as they are reviewing subjects they already understood [2]. However, when taking into account teaching practices that are designed to increase self-learning and self-confidence, we continue to see an improvement in student performance.

A common technique now utilized in introductory Computer Science
courses is Peer Instruction. A technique that was first used in other fields, such as physics [38] and biology [39–42], it involves asking students questions, the encouraging them to discuss the answer with a partner or as a small group before they answer the actual question for a graded amount. This technique has been developed in many different forms, but it has been shown to increase student confidence levels as they see they are not the only ones struggling with the topics. Not only has Peer Instruction been shown to decrease failure rates, it has also been shown to increase satisfaction with a topic [43–46].

Another popular technique that builds upon social interaction with students, and thus encouraging non-academic factors is Pair Programming. In this sense, paired programming is defined as two students working on a single assignment in a lab environment, often due at the

[1]ACT traditionally stood for "American College Testing", though the organization is formally ACT. In the papers surveyed, the term “non-academic factors” was used in the same context ACT uses in their analysis.

end of the lab. While the method varies, it is common to make sure there is a coder and a guider, the coder is typing, while the guider provides the algorithm and guides the coder's actions. They then switch roles, so both students experience guiding and coding. Pair Programming has shown that students are more engaged and motivated, higher student confidence, and better quality programs/labs which leads to higher pass rates for students by about 10% [47–51]. However, the really strong aspect of Pair Programming is its influence on underrepresented students, particularly women. In most cases, their retention is close to double, with increased performance, satisfaction, and enjoyment [48].

Motivation and student performance are often related, and the lack of student motivation can be a major factor in high failure rates. Nikula et al. conclude that low motivation and immature student behavior is a major problem affecting pass-fail rates, and they seek to introduce three steps to improve student motivation by: (1) reducing the complexity of the course, or demotivators, (2) increasing intrinsic motivators by making assignments more appealing, and (3) add extrinsic motivators by asking students to submit assignments weekly, rather than all at once. These three changes increased their pass rate from 44% to 68% [2,21]. While a noticeable increase, the changes only brought them up to the international average of a 33% failure rate. In a similar approach, Corney et al. focused on improving student engagement through assignments termed practicals. Students were asked to build larger projects, but in teams throughout the semester. The projects focused on "real world" examples: such as database programming, web applications, and even graphics. While already showing a higher than average pass rate of 81%, they did show an increase to a 94% pass rate, dropping the failure rate to 6% [52]. This work showed that assignments play a critical role in student motivation, with a focus on increasing student interest in the topic.

2.2.4 Interest Based CS 0 and CS 1

To increase diversity, inclusion, and motivation for students universities have turned towards "interest based" CS 0 and CS 1 courses. Between 1995 and 2000, Carnegie Mellon Univer-

sity (CMU) went from 7 to 42 percent women in their program and continues to be high today. Given that the average percent of women in any program in 2019 is 20.8 percent [53], CMU has led the charge in gender inclusively within Computer Science. The areas in which CMU changed are as reported. (1) Change the culture by emphasizing there are multiple ways to be a computer scientist. The idea of hacker culture was a deterrent for many, especially women. Faculty were especially challenged to bring diversity questions into their research and their discussions with students. (2) Update curriculum to focus on computing in context. It is argued that science must be in the context of their social implications. They concentrated on bringing in interdisciplinary courses focusing on both programming and diverse problems, added an undergraduate concentration in human computer interaction, and worked with non-profit groups for projects. (3) They created multiple entry points for students with all different coding backgrounds, this includes adding CS 0 style courses and courses focusing on prior and non-prior programming experience. (4) They encourage faculty and staff to pay more attention to good teaching and good teaching practices. They reported that confidence for women is closely linked to teaching pedagogy, and their relations with faculty. (5) Departments need to focus on building in support for confidence issues. They point out it may be ideal to bring in cognitive psychology studies in the future, with a primary focus on educating all students in the department about their comments and actions that could be causing a toxic environment for women [54].

Harvey Mudd College (HMC) revamped their program in 2006 with three target areas. These changes stemmed from them having about 19 percent women: this included those who were majors, interested in becoming majors, or just simply taking an additional CS course; increasing to 51 percent by 2008 who were either majors or interested in majoring in CS. The three initiatives they implemented were using the Grace Hopper Celebration (GHC) as a means of recruiting, and not retention. Most notably, they spent funds on sending non-majors who were taking their introductory Computer Science course to GHC. They introduced undergraduate research experiences for women, focusing on students with little programming experience. Most no-

tably, the research experiences focused on students who had completed only the introductory programming course, and the primary change they made was to the introductory CS courses. They revamped their course to be broken up into interest based courses based on the majors taking them. For example, biology takes a course more focused on biology interests, but with a heavy emphasis on how Computer Science influences biology. They made the course more breadth oriented while changing the language to Python. When they surveyed the students, for both males and females, the change to CS 1 was the primary influence in them changing majors to Computer Science [55].

While both CMU and HMC focused on overall program revamps with an emphasis of changes in CS 1, Cal Poly focused their changes to CS 0, as all students are required to start in CS 0 unless they specifically have previous college credit in programming. In "Mixed Approaches to CS0: Exploring Topic and Pedagogy Variance After Six Years of CS0" the authors review the changes they put in place and how the different interest based courses worked out, including mapping to different teaching styles and techniques.

Cal Poly focused on six different versions of CS 0, and studied the results across six years. The six courses are based on the CSC 123 curriculum. The common principles of the 123 curriculum are: equal participation, basic learning outcomes focusing on computing fundamentals, software development process, team-based learning, student mentoring, applications with external clients, and all courses will use the same grading scheme. Their focus was to promote academic self-confidence, motivate students, and encourage social support [56].

By the end of the six years, they listed their common principles as: context based introduction to computing, project based learning, learn by doing via lab work, group work, relating CS topics to real-world problems, and individual creativity in assignments. The courses did vary a bit more by the end of the six years, but the overall goal remained the same; teach students a bit of programming, get them excited, encourage self-confidence and motivate them before they take CS 1 [4].

Each of the six courses had a different focus and slightly different pedagogy. For example, Computational Art focused on topical introduction of topics as they relate to art. While learning about loops they explored the artistic principles of texture, as well as the art movements of pointillism and impressionism. Each course type is paired with their pedagogical methods based in Figure 2.1 [4].

Pedagogy	Comp. Art	Games	Mobile	Music	Robotics	Security
community building	✓			✓		✓
flipped model		✓			★	
individual creativity	★	★	★	★	✓	✓
individual learning		★	✓		✓	
lecture based learning	★	✓		★		✓
pair programming.			✓			
student feedback (polling)	✓	★		✓		★
student reflections (journal, etc.)	✓	★				★
team projects		★	★	★	★	✓
test 1st design (Htdp)				★	★	
topic based intro to CS	★		✓	✓	★	✓

Figure 2.1: Cal Poly CS0 course pedagogy used by interest topic. Stars indicate a predominantly important pedagogy used, while check marks denote somewhat important in their teaching the course. This figure is a copy of table 1 as listed in [4].

Looking at Figure 2.1, the stars indicate techniques the instructors felt were predominantly important in their course, while the check boxes indicate techniques that were somewhat important.

When looking at measuring their results, they focused on overall grade comparisons, retention numbers, and student surveys. In their initial results, they showed that those who took CSC 123 increased their grades by 10% in CS 1 while reducing the number of students failing. They saw a 12% increase in the number of students taking CS 1 [56]. After six years, they showed the 5 year graduation rates increased from 51.7% for CS majors to 66.7% by their second cohort. They showed an increased pass rate in their CS 1 course while also looking at the GPA distribution. While the music focused CS 0 course had a slightly higher GPA distribution, there was

not a significant difference between the "flavors" of CS 0 courses the students took and their performance in CS 1 and CS 2. From their survey data, they found that those who took CS 0 first had a 10% higher rate in their excitement based on those who marked strongly agreed to being excited for Computer Science after taking CS 3. However, even with students taking CS 0 they found Computer Science to be "harder" and "more antisocial". Overall, Students find Computer Science hard, but those who took CS 0 are more motivated to succeed [4].

The strength of Wood et al. is that they are able to show even with a variety of techniques and a variety of interests, students perform better. They will go on to state that the existence and goals of a CS 0 course matters more than specific content or even pedagogy [4]. Looking at their work, the goal is important as the overarching goals of their CS 0 courses are uniform, with an emphasis on promoting student self-efficacy, motivation, and excitement. These are the same non-academic factors that show prominence in the work presented by Watson [37], and is then supported by teaching techniques that all promote student self confidence; such as peer instruction and pair programming. Wood et al. are not alone in promoting interest based CS 0 courses that encourage student self-efficacy, and it is clear there is a benefit to a CS 0 course that encourages students to be interested in Computer Science [57–62].

2.2.5 Another Measurement

Alturki presents an extensive review of Computer Science education, as the context of their work is focused on measuring student performance. Most of Computing Education is focused on improving student performance and most of the papers reviewed use the following techniques as standard for the field: grade changes between cohorts, exam results, and student opinion surveys. Alturki presents another way to measure performance that is focused along the line of assignments compared with grades. Their goal was to discover a correlation between assignment grades within a single course. Their course was broken up into the following categories for assignments: quizzes, assignments, midterm, labs, and final exam. The weights for

each category are shown in Table 2.2, with different course sections having slightly different weights [2].

Table 2.2: Data presented by Alturki on point distribution between assignment groups [2].

Assessment Method	Period 1	Period 2
Quizzes	10%	10%
Assignments	10%	0%
Midterm	20%	20%
Labs	20%	30%
Final Exam	40%	40%

The way they measured the relationship is by generating a difference score for each category. For example, if a student earned 100% on their quizzes, but only 95% on their assignments, there would be a 5% deviation between the two. Across all students you can calculate the average deviation. Ideally, if assignments were good predictors of exams, the deviation value would be low. If they are poor predictors, the deviation value would be high. Comparing assignment groups, Alturki found that midterm exams were great predictors for the final score in the course [2].

When comparing assignment categories, knowing which changes helped influence the final score in the class helps with understanding which categories work best for helping students retain information, and also these early indicators can help with interventions for students. While this comparison may seem to be a performance comparison like the papers reviewed, the actual performance of the students doesn't matter. What matters is how variable that student performance is between assignments, and if assignment types are good indicators of final grades. The downside of Alturki's work is that in addition to comparing assignment categories, they compared categories against the final score in the class. While one may assume that is a natural extension, the problem comes in when the weight of the assignment groups affect the final score for the course. As such, Labs and Midterm are going to have a high correlation with final grades by the nature of the category weights. They concluded that midterms are the nat-

ural indicators for final grade performance, but this is also due to the extra weight applied to exams, especially the midterm. Overall, Alturki has an impressive literature review covering an extensive background defining Computer Science education as the means in which one makes changes to courses to improve student performance and retention, but whose way of measuring that performance while a way to look at the course from within instead of student performance, fails to define a way to normalize the weights between the groups and the final grade.

The research indicates that Computer Science can be difficult for students, and if motivating students and encouraging confidence and self-efficacy are essential skills for learning Computer Science, what grounding is there for these techniques within education and psychology? Is the goal in CS 1 not only to teach the programming, but also teach students a different way of thinking and looking at the world?

2.3 CS 1: Teaching Students Different Ways of Thinking

To be successful in Computer Science, it is believed that the development of specific cognitive skills, including metacognition, and self-regulation [63] and self-efficacy [64–66], are essential to success. Including the development of these skills, programming itself is an inherently cognitive endeavor [67]. VanDeGrift et al argue that it is the job of CS faculty to teach both programming and metacognitive skills [68]. Malmi et al. focus on the importance of building a better theoretical understanding of how students learn and understand computing concepts and processes. They categorize papers into 65 different theoretical constructors, grouped into 11 different focus areas, as teaching constructs within Computer Science. Phenomenography, qualitative research on student experience, was the most common research method used to develop a theory, with most theories attempting to model the rich relationship between students and computing concepts [69]. However, it was noted by Prather et al. that while there is a tradition of co-occurrence of metacognition and self-efficacy in computing education, the review done by Malmi et al. did not have papers on metacognition and self-regulation. Motivated by this gap, Prather et al. provide a review of literature that mentions metacognition and self-

regulation specifically published within the ACM Special Interest Group in Computer Science Education (SIGCSE) context. They chose the SIGCSE venue as a means to narrow down their search to papers focused specifically on teaching Computer Science [70].

Defining metacognition as the ability to describe knowledge about one's own cognitive control, including strategies that are successful or unsuccessful to learning, monitoring emotions and self-efficacy, evaluating feedback, and defining self-regulation as the process in which one executes cognitive control, Prather et al. admit the terms are interlinked between knowledge and practice. While not directly stated, they reference the ability to develop self-efficacy within metacognition and self-regulation. Their survey originally looks at 2531 papers across all ACM publications, 357 papers within the SIGCSE group, and then refined that number to 214 papers across the following range of venues: ACE, ACSE, CompEd, FDG, ICER, ITiCSE, Koli Calling, SIGCSE, WCCCE, and WiPSCE. The 214 papers were grouped into passing, peripheral, and depth categories with passing referencing papers that only mentioned the terms briefly, peripheral discussing the terms and their relationship to Computer Science, but are not used to support methodology or interpret results, and depth treating metacognition and self-regulation as central themes to the paper. The papers were broken up into group A, B, and C with each group evaluated by two of the six authors of the survey paper, and during this process, seven more were removed due to misuse of the terms in the context of their search, leaving a total of 207 papers. Passing papers consisted of the majority at 123. Peripheral papers consisted of 53 papers, and depth papers consisted of 31 papers. They also noted that a flaw in their potential study was that they were missing papers that talked about self-regulation and metacognition, but did not use the terms explicitly citing a paper written by Prather et al. that was excluded early on in the process due to the lack of key word use. They felt this was a rare case and not enough to affect the overall validity of the survey. While their earliest paper dated back to 1978, half the papers (106) were published within the past five years, and the greatest amount published in 2019 (35 of the 207 papers). This shows an increased interest in Computer Science and how it relates to metacognition and self-regulation [70].

Prather et al.'s review took a high level overview of the passing and peripheral papers focusing on the general theories of metacognition and self-regulation used, along with the types of measures used to determine metacognitive growth. They then focus on the pedagogical techniques used within the depth papers.

2.3.1 Measuring Metacognition and Self-Regulation

Models to measure metacognition and self-regulation have evolved over the years, and there is not a set evaluation measure. Prather et al. provide summaries of the seven assessments used across the papers they survey. The seven techniques are:

- Motivated Strategies for Learning Questionnaire (MSLQ) by Pintrich and De Groot [71].
- Classroom Assessment Techniques (CATs), in particular Recall, Summarize, Question, Comment and Connect ($RSQC^2$) and Muddiest Point by Angelo and Cross [72].
- Self-Efficacy Scales by Ramalingam and Wiedenbeck [65].
- Self-Regulation Questionnaires by Ryan and Connell [73].
- Student Perceptions of Classroom Knowledge Building (SPOCK) by Shell et al. [74].
- Epistemological Beliefs Questionnaire (EBQ) by Schommer [75].
- Achievement Goal Framework Questionnaire by Elliot and McGregor [76].

Used across 11 different studies, the Motivated Strategies for Learning Questionnaire (MSLQ) was the most commonly used measurement in the literature. The first version of the questionnaire was published in 1990 by Pintrich and De Groot with the goal of analyzing student performance in STEM disciplines. The questionnaire has 44 questions broken up into the categories of Self-Efficacy (9 questions), Intrinsic Value (9 questions), Test Anxiety (4 questions), Cognitive Strategy Use (13 questions), and Self-Regulation (9 questions). Self-Efficacy, Intrinsic Value, and Test Anxiety are then grouped together for gauging student Motivational Beliefs,

and Cognitive Strategy Use and Self Regulation are grouped together in Self-Regulated Learning Strategies. The categories are meant to be used either together or independently, and students answer on a 7-point Likert-type scale from “not at all true of me” to “very true of me” [71]. A later work, went from the seven groups to nine groups: Task Value, Self-Efficacy for Learning and Performance, Test Anxiety, Rehearsal, Elaboration, Organization, Metacognition, Time and Study Environment Management, and Effort Regulation [77]. A critique proposed by Leppänen et al. is that MSLQ is subject to construct validity threat, in that MSLQ may measure what students think they do, rather than what they actually do [78].

Classroom Assessment Techniques (CATs) refer to a collection of strategies. RSQC^2 combined with Muddiest Point was used to bring elements of metacognition into courses. For example, RSQC^2 provides a structure for asking questions to encourage students to elaborate upon information they have learned in lectures. Muddiest point has students identify parts of a lesson they found unclear. For comparison sake, while MSLQ was found in 11 papers, only two papers reviewed used CATs, and CATs was the second highest measurement technique used. It should also be noted that due to the nature of CATs, one can use them both for teaching and assessment of the students [72].

Self-efficacy is known to improve student performance in CS 1 [36], and the Computer Programming Self-Efficacy Scale [65] was used to evaluate a number of techniques. Student focused techniques looked at self-efficacy as it related to peer instruction [66].

Self-Regulation Questionnaires (SRQ) were developed for assessing when students act autonomously while performing tasks, and is often customized to specific domains [73]. In Computer Science, Toma and Vahrenhold used SRQ assessment in their exploration of psychological factors that affect self-efficacy and emotions in collaborative work. They found labs to be a positive influence on students, especially when allowed to work together and collaborate [79].

The last three assessments were not widely used, but presented by Prather et al [70] as they were all used once across the papers surveyed. Student Perceptions of Classroom Knowledge Building (SPOCK) is meant to help develop collaborative learning tools within a classroom.

Questions are presented which assess student perception and encourage student reflection on their learning habits [75]. Epistemological Beliefs Questionnaire (EBQ) is an instrument used to measure both teacher and student beliefs about the nature of knowledge and its acquisition. The areas looked at are: (a) “Knowledge is simple rather than complex”(Simple Knowledge), (b) “Knowledge is handed down by authority rather than derived from reason” (Omniscient Authority), (c)“Knowledge is certain rather than tentative” (Certain Knowledge), (d) “The ability to learn is innate rather than acquired” (Innate Ability), and (e) “Learning is quick or not at all” (Quick Learning) [75]. The Achievement Goal Framework Questionnaire looks at building mastery-performance in a 2x2 framework: mastery-approach goal, performance-approach goal, mastery-avoidance goal, and performance-avoidance goal. The 2x2 framework provides a different approach to competence-based self-regulation [76].

2.3.2 Theories and Pedagogical Approaches

Prather et al. references 11 theories of metacognition that are used throughout the papers they evaluated. They group them into three groups. Motivational theories contain self-efficacy theory, which affects strategy use and coping mechanisms, mindset theory which focuses on strategy selection, goal oriented theory in which self-regulation focuses on beliefs and progress towards goals, temporal motivation theory which focuses on setting behaviors and reducing procrastination, self-determination theory focused on both intrinsic and extrinsic motivators, and control-value theory of achievement emotions which affects learning strategy planning and regulation. Cognitive theories contain cognitive development theory in which planning is not possible until later stages of development, and cognitive load theory in which learners need resources to use metacognitive knowledge. Social theories include model of school learning in which generalized skills affect time to complete tasks, social learning theory in which cognitive tasks are difficult to learn as they are not visible in others, and cognitive apprenticeship in which control is learned as part of skill development [70].

Not mentioned in their theory grouping, but mentioned as a theory representing metacognition and self-regulation is Revised Bloom's Taxonomy [80]. Bloom's Taxonomy [81] was originally designed for learning objectives, but was later revised to include metacognition and self-regulation as an added dimension learners can develop. In a work missed by Prather et al. due to the lack of explicit words they used in their initial search to find papers, Thompson et al. use the revised Taxonomy in relation to Computer Science, and the importance of developing cognitive skills related to programming. Within the Bloom's categories, Thomas et al. provide the following examples. *Remember* is defined as 'retrieving relevant knowledge from long-term memory', which when related to programming is identifying code and knowledge, and recalling material as presented. *Understand* is defined as constructing meaning from instructions. Thompson et al. use converting algorithms from plain English to code and vice versa as an example of understanding. Code tracing would fall into this category. *Apply* is using a procedure in a given situation, applying and writing of programs is applying their knowledge. *Analyse* is focused on breaking material into smaller parts. This means taking problems and breaking them into smaller parts, also program design between classes and methods.*Evaluate* code is going beyond just seeing how it works, but more about how it can work better than its current form. *Create* is focused on constructing code segments. While these are high level categories, Thompson et al. found the explicit use of the taxonomy forced them to evaluate their work, including exams, and that it helped them define their own cognitive process in addition to the students' process. They were able to use certain wording to invoke certain cognitive processes [82]. It should also be noted that create is one of the highest elements in Bloom's, but often one of the first things instructors ask of students in CS 1.

The real strength of Prather et al.'s work, outside of providing a high level overview of metacognitive and self-regulation research, is providing a series of pedagogical themes that show up in the more in depth research. The weakness of their paper is that these themes are more self-selected, instead of themes already presented in psychology. The themes presented by Prather et al. are reflective activity after assignment/exams, reflective activity as a structured assign-

ment, visualizations, in-process scaffolding, active learning, and awareness (of procrastination / time management). Three of the themes listed were evaluation of themes more than pedagogical themes: behavior surveys, standardized tests, and data collection. Though the last three could be used as a means to develop a pedagogy [70]. Each of these pedagogical approaches, while categorized differently, will be addressed in more detail in Sections 2.4.1, 2.4.2, 2.4.3, and 2.4.4.

In conclusion, Prather et al. highlighted the central themes of their survey [70].

- Metacognitive knowledge is difficult to achieve in domains about which the learner has little content knowledge.
- Metacognition and self-regulation are often directly related to self-efficacy, and self-efficacy is often directly related to performance. However, the direct link between cognitive control and performance is weak.
- Self-report measurements of cognitive control, such as the MSLQ, often measure what students think they do, rather than what they actually do.

However, they also found the definition of self-regulation and metacognition is loosely used within Computer Science, and they encourage the field to settle on a standardized definition. They recommend the use of multiple measurements, and the mapping of practices onto theories [70]. It is in this last aspect that their paper is weakest. While they provide the theories, it would have been stronger if they linked the pedagogical techniques with the explicit theory. Instead they list theories, but there is a decoupling of the techniques which is counter to the conclusion. Overall, the paper provides a very strong introduction to metacognition and self-regulation in the context of Computer Science.

One aspect in which there could have been improvement is the grouping of the pedagogical themes. While Prather et al. makes reference of specific techniques common in learning and memory research, they fail to address those techniques directly as themes. However, within psychology, there are techniques that are often grouped together.

2.4 Improving Learning Efficiency

There is a long history with memory research, dating back to the Seminal 1885 work of Ebbinghaus [83–85]. 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 efficient 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: (1) process material actively, (2) practiced retrieval, (3) distributed practice, and (4) use metamemory [85]. A related book, "Make It Stick" encourages readers to use psychology to improve retention of material, presenting eight different practices to improve memory: retrieving, spacing, interleaving, elaboration, generation, reflection, calibration, and mnemonic devices [86]. Of these eight, five of them are heavily repeated in cognitive science literature. These five are highlighted by Roediger et al. as spacing, interleaving, practiced retrieval, elaboration, and reflection [84].

While Schwartz et al. and Roediger et al. utilize different themes for improving memory efficiency, 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 metamemory has reflection and metacognition at its center. In the literature, Roediger's listing of the learning techniques is more popular, so while the next four sections will follow the Schwartz et al. wording, they will be grouped as Roediger's themes within Schwartz's broader categories. A downside to Schwartz et al. is that their groupings may be slightly too broad, though the general principles are sound and well supported by the literature. The works reviewed by Prather et al. [70] will be grouped at the end of each section as a way to relate what is currently being done in Computer Science in relation to learning efficiency.

2.4.1 Active Learning

Active Learning often involves asking students to elaborate on the topics they are learning in lecture. Elaboration is about asking questions about the topics and seeking answers to those questions. Another way stated, it is the "why" question. Why does something work the way it does, and then seeking the answer to the why question. It is also about developing a body of information based on course content in which the student is forced to seek answers not readily given in the course. Elaboration has been shown to help with both retention and understanding of material [87]. The downside of elaboration is that it slows down reading content, adding additional time requirements on the student to seek answers to the "why" From the student's point of view, it is hard to see the need to dive deeper when balancing multiple classes and time constraints if they are not presented with the benefits [84]. However, that does not have to be a limiting factor with elaboration. Schwartz et al. point out there are many aspects to active learning, including word-associations when pairing, preparing to teach someone the topic they just learned, whether they teach it or not, and others. The goal is that when students think deeply about topics, relationships and organization, they will learn more than taking a passive approach to education [85].

Within Computer Science, peer instruction [38,43–46,66] and pair programming [47–49,51] are both methods of active learning. Bagley and Chou found that deliberate collaboration was important for student metacognition and understanding. They found it was most important at the brainstorming stages, allowing students to intentionally brainstorm together is also a form of active learning. Females responded particularly well to the encouragement to collaborate either in groups or via a mentoring relationship [88]. 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, and their results show improved grades shifting a course with predominately C grades to a course with predominately B grades [89].

Visualizations as listed by Prather et al. is another form of active learning, or more explicitly, elaboration. They are taking content, representing it as visual representations, and asking students to dive into that information. Visualization can vary, but the most common one presented by Prather et al. is the use of visualization when looking at code. For example, when a student checks in their code, they are provided with a visual representation of the flow of the code, the layout and other information. Proponents of visualization, such as Yan et al. argue that setting up visualizations of students' code put the human element back into CS, as it encourages students to discuss their code in a unique manner, and also improves student problem solving processes [90].

2.4.2 Practiced Recall

Practiced retrieval, better known as testing, is often misunderstood as a means of evaluation of progress and knowledge in an area. While testing is most definitely used as a means of evaluation, the testing effect is a well documented means of enhancing recall and memory. Stated another way, testing by itself is a way to force memory retrieval which then enhances memory. Roediger provides an overview of ten different benefits of testing [91]:

- The testing effect: retrieval aids later retention
- Testing identifies gaps in knowledge
- Testing causes students to learn more from the next learning episode
- Testing produces better organization of knowledge
- Testing improves transfer of knowledge to new contexts
- Testing can facilitate retrieval of information that was not tested
- Testing improves meta-cognitive monitoring
- Testing prevents interference from prior material when learning new material

- Testing provides feedback to instructors
- Frequent testing encourages students to study

Of these ten benefits, 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 [84, 86, 92, 93]. Unfortunately with testing, it is the least intuitive of the practices. Students when surveyed often prefer to "restudy" or review material they have previously studied, such as going over notes, slides and rereading content. It is believed familiarity of content causes the false illusion of knowledge, and students often assume they know content before they actually know it [85, 94].

There is very little on testing as a means to improve student performance, even though tests are extensively used within the field. Contrary to the benefits Roediger presented, Watson et al. analyzed 25 predictors for programming performance, and found the only predictor was self-efficacy as predicted by MSLQ questionnaire. This leads Watson to encourage less testing [95]. However, this does not take into account the other nine benefits of testing, and the value of short frequent tests as compared to the midterm style tests used in Watson et al., which could indicate that proper testing may be useful for student metacognition.

2.4.3 Distributed Practice

As defined by Schwartz et al. "Distributed practice is learning that is spread out across relatively long periods of time rather than massed all at once." [85]. This is also called the spacing effect. The spacing effect is one of the oldest findings in experimental psychology dating back to 1885 with work done by Ebbinghaus [83–85]. There is an extensive body on the benefits of this research, and Cepeda et al. provide an updated overview of using spacing to enhance recall [96]. The spacing effect describes the benefit in memory retrieval of topics when those topics are studied with a gap between sessions. For example, when someone studies a little each night instead of hours before an exam. The process of forgetting, combined with forced recall of the topic enhances student long term memory of that topic. In contrast to spacing, most students

study using massed or blocked practiced, conventionally known as "cramming" [85]. However, massed practice is the most common, and it is not solely on students as course design can unintentionally encourage massed practice. For example, only releasing the study guide for the exam a couple days before the exam encourages students to study only after the guide is given. Not leaving time to go back to topics also encourages students to study in blocks and not return to previous material. However, the benefits between spacing and massed practice are a bit more nuanced than that, as short term recall is benefited by large block studying. This means students see immediate benefit, even if long term recall is not there. Schmidt and Bjork showed that with progressive testing, recall decreased when they studied in blocks as compared to increasing results when students practiced spacing [97]. This leads to the conclusion that massed practice is better for short term memory and spacing for long term memory. Furthermore, it has been noted 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. For example, someone studying an hour the night before the exam, as compared to someone studying 20 minutes a night for three nights. The individual who studies 20 minutes a night for three nights will often outperform the person who studies for an hour the night before the exam [98].

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 studies math by learning multiplication and then studies by learning division, as compared to a student who studies both topics at the same time intermixing what they are learning. It has been shown that interleaving of these topics improves recall in multiple domains, including mathematics [99, 100]. It is also believed a strength of interleaving is that students are better at discerning and figuring out the problem, causing fewer discrimination errors [84]. Using the math example, a student who practices the same type of problem with variation already knows the goal of the problem, causing them to skip a step in which they have to discern the goal of the problem. Interleaving the problem types means a student must first discern the goal of the problem, before working

on the problem. In an ideal world, students combine both interleaving and spacing for all study sessions, by interleaving topics while spacing out their study sessions.

There is very little on spacing within Computer Science education or interleaving. Simply put, Computer Science educators tend to follow an unspoken universal norm on the order of what they teach for CS 1 courses, without much thought in the restructure of that ordering. One study that looked at spacing within Computer Science was by Leppänen at al. studying pauses and spaces when students were working on programming projects. They found, contrary to their expectations, that the longer breaks students took, the worse they did in the course. Their study only looked at spacing of programming assignments, often week long assignments which students would spread out coding across the week [78]. It did not look at spacing or interleaving the course topics, or even intentional studying by use of breaking up study sessions. This leaves an area open for debate on the benefits or disadvantages of spacing or interleaving when it comes to the study and application of Computer Science.

2.4.4 Use Metamemory

Metamemory is the one category where Schwartz et al. does not easily coincide with the rest of the literature. Using metamemory is using metacognition and more importantly self-regulation as they relate to study habits. Judgements of learning (JOLs) play a major role in student study habits, or self-regulation. However, like every heuristic-based judgement, they are subject to systematic bias. For example, students tend to be overconfident, which may lead them to not studying a topic they need to study, and humans tend to overlook the fact they forget topics. Just as important as knowing when to study, is knowing how to study, meaning students know to apply the other techniques listed [85].

Of the four principles, this ends up being the most complicated one, and one could argue it is overly complicated. The reason why is that the parts that make up metamemory, metacognition and self-regulation, are really broad categories. This causes the question, "What major techniques improve self-regulation?" In Prather et al. they review a number of themes, but the

predominate theme is reflection [70]. Reflection is also a primary theme in the Roediger et al. listing [84].

Reflection or self-explanation often requires individuals to monitor and describe either out loud or internally their learning process. Going beyond class content, they are asked to question their own understanding of the content. How much do they really know? What do they know, and more importantly, what do they not know? What are ways they can improve? Similar to elaboration, this strategy requires the student to be an active learner, even when reading book material. Such practice is known to elicit improved recall and understanding [101]. Even more, reflection improves applying knowledge to very different problems, termed far-transfer of information between topics, and was studied within mathematics. Wong et al. gave students questions to ask while reading. Those in the questioning group performed better than those in the standard group when applying their knowledge to related domains that were not yet covered in studying [102].

The benefits of reflection after exams and assignments has been extensively studied in Computer Science Education, with noticeable benefits to student performance. Common areas are broken up into reflection activities as part of another assignment or exam, and reflection as separate assignment in the course [63,70,103–107]. Kann et al. confirmed that going beyond reflections with discussions about those reflections actually improves both their reflections and their future coding activities [108]. Beyond school, students who are encouraged to reflect on their learning, will have an advantage in their related Computer Science professional careers [109].

Overall, there are many benefits to reflection, and while not the silver-bullet [110], teaching and having students reflect is a very strong tool used to promote Metamemory, metacognition, and self-regulation for students.

2.5 Conclusion

Learning Computer Science can be a difficult endeavour, especially with the current expectations placed upon students. This literature review defined CS 1 as the primary introductory

course for most students entering the major with nearly a universal standard of topics taught, but an analysis of variables for performance showed that sometimes taking a CS 0 course first is better. A further analysis of variables and best practices, showed that even after thirty plus years, the failure rate in Computer Science remains high. As such, instructors need to turn towards models of learning that are specific applications applied to Computer Science of well known educational techniques and cognitive models. These models help instructors promote metacognition and self-regulated learning, both of which are tied to understanding Computer Science and problem solving. Last, this review detailed four major areas in memory research that are being applied to education, and how Computer Science currently uses those areas to improve performance.

Of the topics and techniques listed, there are two major areas that show gaps: distributed practices of topics taught, and practiced retrieval as a means to improve recall. Furthermore, most techniques have been applied as additions to the current CS 1 structure, commonly accepted as the standard structure to teach introductory programming. Instead, a challenge would be to take these learning practices and redesign a course around all of them as a unit, causing an introductory Computer Science course to focus on teaching students both what to learn *and how to learn*.

Chapter 3

Current Courses At Colorado State University

Fall 2016, Colorado State University launched redesigned CS 1 and CS 2 courses. These courses were redesigned to match the CS 1 and CS 2 recommendations by the CS2013 Ironman Draft while also fitting in with the department guidelines. After the first year, they found the changes called for the development of a CS 0 course, and over the last three years, they have made changes to better the courses. Section 3.1 highlights the changes originally made to CS 1 and CS 2 with an analysis of the initial success of those changes. Section 3.2 details the need for a CS 0 course that was then first implemented in Fall 2017, and some questions that developed based on those changes.

3.1 CS 1 and CS 2: Fall 2016-Spring 2017

Led by Chris Wilcox, CSU redesigned CS 1 in Fall 2016. They took the previous CS 160, CS 161, and CS 220 sequence which had discrete mathematics spread throughout, and redesigned the intro course sequence to follow the ACM guidelines where discrete math was mainly placed inside of its own course, which in turn restructured how CS 1 was taught as an introduction to programming and made CS 2 a data structures course. Over the summer of 2018, Wilcox and Lionelle analyzed the changes in particular to CS 1 and CS 2 [3].

3.1.1 CS 1 (CS 163 and CS 164) Performance

CS 1 was broken up into two sections with identical content, but targeted different audiences. Most notably, *CS 163: Java (CS1) No Prior Programming* targeted students with no prior programming experience, while *CS 164: Java (CS1) Prior Programming* targeted students with limited programming experience. Students were allowed to self select, though their advisor would encourage placement during a summer orientation discussion with the students. The reason for the split was to follow what they witnessed being done at other universities, most

notably to reduce the intimidation factor when students who have programmed before are answering questions in a room with students who are seeing the topics for the first time. After students completed CS 1, their next course in the sequence is *CS 165:Java (CS2) Data Structures and Algorithms*. The two student groups are combined, with the assumption they both have the minimal levels of programming needed for CS 165.

While the two courses were for different audiences, encouraging different types of discussion, the content of the courses were exactly the same including assignments, labs, website, and other course materials. For their study, the instructor remained fixed not only for CS 1, but also for CS 2. The instructor, Wilcox, was the designer of the new course sequence. The topics covered on a weekly basis can be seen in Table 3.1. They follow the chapter order in *Introduction to Java programming and data structures* by Liang [111], though they made ample use of Zybooks for java. Furthermore, Wilcox adopted techniques recommended by previous literature, including:

1. Adding Intrinsic and Extrinsic motivators by modifying assignments and labs [2, 21].
2. Emphasis on CS Culture not being the "coder in a cubical" [53].
3. Interesting Assignments, often showing off graphics or other more advanced topics [52].
4. Peer Instruction plays a major role in the course design [43–46].

When comparing the groups of students, Wilcox and Lionelle found that the CS 164 students significantly out performed the CS 163 students in quizzes and exams, while 163 students did better in labs, programs and assignments, but not with any statistical significance. Furthermore, Wilcox believed the nearly identical scores for labs, programs, and assignments was connected to the extensive use of help desk, an on-campus tutoring service, and potentially plagiarism [3]. Detailed in Table 3.2, the greatest difference was found in the course quizzes, where 164 students scored 10.52% higher than their CS 163 counter parts. Furthermore, 34% of the CS 163 students either withdrew from the course or earned a grade of D or F. Since CS 165 requires a C or above in CS 1 (CS 163 or CS 164), a D would be considered not passing the

Table 3.1: Schedule of topics for Fall 2016 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	Multi-Dimensional Arrays
Week 9	Classes and Objects
Week 10	Exceptions and File Input/Output
Week 11	Abstract Classes and Interfaces
Week 12	Recursion
Week 13	Collections and ArrayLists
Week 14	Sorting and Complexity
Week 15	Evolution of a Software Engineer
Week 16	Final Exam

course from the perspective of students moving forward. In contrast, 26% of CS 164 students either withdrew or earned a D or F grade. Grade comparisons can been scene in Table 3.3.

Not mentioned in the Wilcox and Lionelle paper is the affect of grade differences on the weighting of each category, a similar issue expressed with Alturki's research [2]. 70% of the students grade was made up exam style assessments: quizzes (20%), midterms (30%), and final exam (20%). These are the three areas that CS 164 students did better than their counterparts in, which means they had more experience recalling the information than those who did not have previous programming experience. This would also mean their performance with recall helped them have a higher overall grade. Simply stated, the correlation with students having higher overall all grades was introduced by the weighting of the exams.

Table 3.2: CS 163 vs CS 164 Grade Area Comparison.
Table modified with permission from [3].

Samples	CS 163 Average 200	CS 164 Average 94	CS 164- CS 163	T-Test Values
Programs	82.00	81.32	-0.68	t = 0.24, p = 0.79
Labs	90.85	89.76	-1.09	t = 1.30, p = 0.45
Peer	84.60	83.83	-0.68	t = 0.35, p = 0.67
Quizzes	72.38	82.89	10.52	t = -3.82, p = 0.01
Midterm 1	75.39	81.33	5.94	t = -2.44, p = 0.01
Midterm 2	69.36	74.82	5.47	t = -2.21, p = 0.02
Final Exam	68.32	74.99	6.67	t = -2.91, p = 0.01
Total	**77.07**	**80.31**	**3.24**	**t = -1.68, p = 0.04**

Table 3.3: CS 163 vs CS 164 Final Grade Comparison.
Table modified with permission from [3].

Grade	CS 163 Count (Percent)	CS 164 Count (Percent)
A	45 (20%)	38 (37%)
B	59 (26%)	27 (26%)
C	45 (20%)	13 (13%)
D	34 (15%)	10 (10%)
F	17 (8%)	6 (6%)
W	24 (11%)	10 (10%)

3.1.2 CS 2 (CS 165) Performance

While the differences in grades between those with more experience and those without experience is not surprising, the student performance in CS 165 was unexpected for the department. Most notably, by the end of CS 165 there was not a distinct difference between the students. This means no matter the background, in our program students often catch up with each other by the second semester with equivalent performance. This became a selling point the advisors would share with students who often felt intimidated joining the program with less programming experience, and they were encouraged to stick it out through CS 2 as they will see results.

The experiment was setup with two sections of CS 2, with Wilcox teaching both sections. Students opted what section they took, and there ended up being a random sampling of CS 163

Table 3.4: CS 163 students vs CS 164 students Grade Area Comparison in CS2.
Table modified with permission from [3].

Samples	CS 163 Average 86	CS 164 Average 67	CS 164- CS 163	T-Test Values
Programs	86.25	84.69	-1.56	t = 0.58, p = 0.56
Labs	92.51	90.14	-2.37	t = 1.05, p = 0.29
Peer	85.19	83.78	-1.41	t = 0.49, p = 0.62
Quizzes	85.97	87.43	1.46	t = -0.53, p = 0.59
Midterm 1	81.21	82.69	1.48	t = -0.79, p = 0.43
Midterm 2	81.43	84.72	3.29	t = 1.59, p = 0.11
Final Exam	84.10	86.07	1.97	t = -0.63, p = 0.52
Total	**84.46**	**85.00**	**0.54**	**t = -0.27, p = 0.79**

and CS 164 students in each section. Results were from the two sections combined without a bias of grades between the sections. Table 3.4 details the results of the course performance on different assignment categories. Most notably, by the time students finished the course the average difference between CS 163 students who took CS 165 and the CS 164 students who took CS 165, was 0.54%! Just as importantly, none of the differences were significantly different, though similar to CS 1, students in CS 163 showed slightly better results on labs, assignments, and peer instruction activities while CS 164 students showed better testing performance.

In addition to looking at all students, the study looked at performance based on the self-identified gender of the students. They found for CS 164 grades, women significantly out performed their male counterparts in almost all areas with their average final grade being 85.39% compared to 79.63%. This same group of women also out performed everyone in CS 2, with their average grade being 88.22% compared to 84.44% for the CS 164 males who took CS 2. In contrast, in CS 163, the women were slightly lower than their male counterparts, but not at a significant difference. They had an average of 75.51% final grade compared to the CS 163 males of 77.53%. In CS 2, the difference was even less with CS 163 women averaging 84.05% and their male counterparts averaging 84.56%. What is most notable about this difference is that for incoming students, Colorado State University assigns a selection index. This index is based off high school GPA and standardized test scores (ACT/SAT), and the university uses the number

to represent the college preparedness level of the student. For example, Computer Science is a competitive major, and requires a selection index of 107 just to declare the major as an incoming student, else a student will often get placed into undeclared seeking if they do not meet that index. The index scores of each group is as follows.

- CS 163 Women Average Selection Index: 122.84
- CS 163 Men Average Selection Index: 117.20
- CS 164 Women Average Selection Index: 117.67
- CS 164 Men Average Selection Index: 118.97

While such scores are notoriously limited in determining student performance [36], it is notable that the group that had the lowest average grade had the highest high school performance. This indicates that coding background is more important than other high school experience. Wilcox also pointed out the confidence level of the student aligned with their final groupings on performance. Most notably of the CS 164 students 67% felt very confident starting CS 164 as tested by a class survey at the beginning of the course, and 85% of the males felt very confidence. In the CS 163 groupings, only 38% of the women felt very confident, where as 50% felt very confident. The conclusions we can draw from this is that while the CS 163 women actually have the strongest academic background coming into the CS 1 sequence, they were both under confident and less likely to perform as well. They are also less likely to take CS 2 after taking CS 1 [3].

3.1.3 CS 1 Retention to CS 2

When looking at the sample numbers, CS 163 had 200 students, but only 86 went onto CS 2, where as, CS 164 had 94 students and 67 went onto CS 2. That means less than half (43%) of the students who took CS 163 chose to continue onto CS 2, while 71% of the students who took CS 164 went onto CS 2. When factoring in only 124 students (62%) were eligible in the CS

163 group to continue on and 76 (80%) of the 164 students were eligible to continue on, we still show a 19% loss in CS 163 students as compared to 9% loss in CS 164 students.

While we may believe every student should take CS 2, there are still plenty of students who may not need to take CS 2 based on their major. To better understand this discrepancy, only Computer Science (CS), Applied Computing Technology (ACT), and Undeclared Information Technology (USCS) majors were analyzed on who opted to continue onto CS 2. In all three cases, students had to take CS 2 (CS, ACT) as part of their degree, or they were seeking to get into the CS program, which meant CS 2 would be their recommendation while they worked on program entry requirements. It was notable to see that 100% of the CS 164 women who were CS or USCS majors went onto CS 2. However, there is also a factor of scale, as CS 164 only had 9 women. In contrast 66.67% of CS 163 women who were CS/majors went onto CS 2. Figure 3.1 shows the percent retention across different major groups and CS 163 and CS 164 groupings.

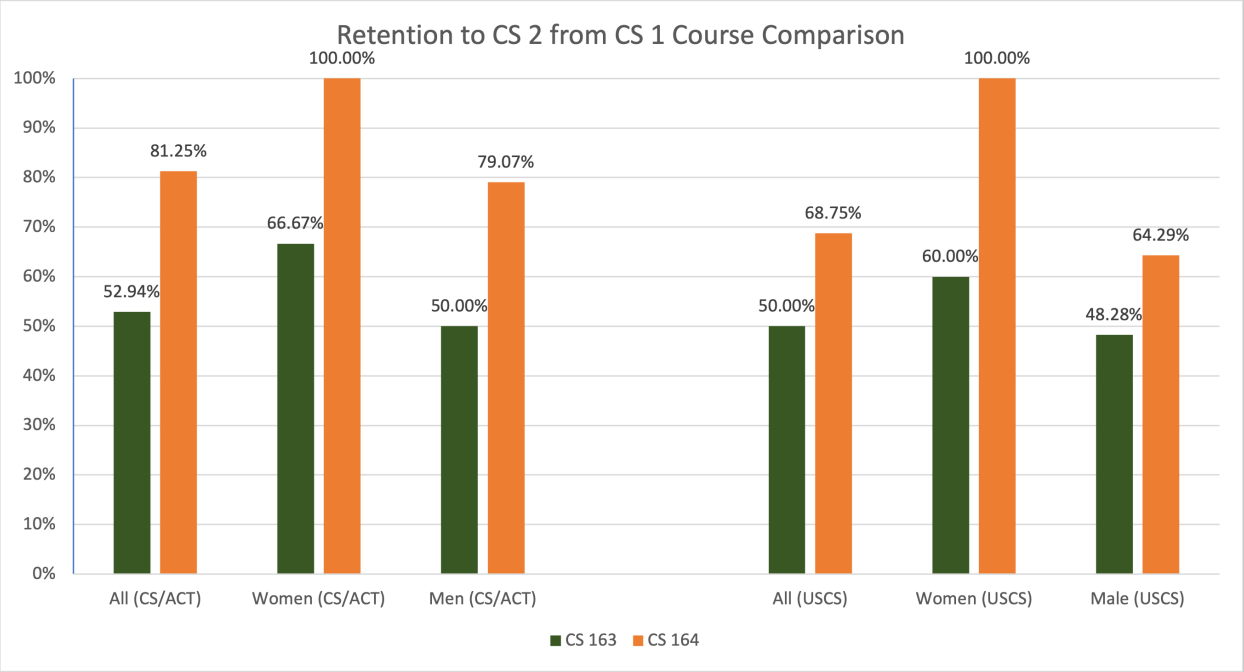


Figure 3.1: Percent retention based on majors declared at the start of CS 1, and how many students chose to take CS 2.

The concern with the retention difference is that while students don't show difference in performance in CS 2, it was theorized that was due to the intensive "culling" that happens with CS 1 students in CS 163. Simply, more students are not continuing, so only the better ones choose to move on. Additionally, given the performance of the CS 164 students, it was believed it would be advantageous to the department to add a CS 0 course to the curriculum. In Fall 2017, CS 150 was reintroduced to the curriculum.

3.2 Introducing CS 0 (CS 150)

The purpose of CS 0 (CS 150) at Colorado State University was two-fold. It was meant to be an introduction to our CS 1 course, taught in Java, for students who do not have the algorithmic / math background to jump directly into our CS 1 Course, and for other majors who just need a little bit of programming. The topics were selected to cover the first 8 weeks of our first CS 1 course but across 16 weeks. They can be found in order in Table 3.5. Any new course design was required to cover these same topics. Furthermore, it was hoped that creating the course would reduce the number of students who were choosing to not continue the program between CS 163 to CS 2, by creating a pipeline of university students into CS 164 which showed greater success at filtering students into CS 2. Namely, students who take CS 0 know what they are getting into, have chosen to continue once, and are more likely to continue to CS 2.

This design was popular, but one problem came up. While students were taking CS 150, the university shifted their CS 163 retention issue to CS 150, similar to the issue reported by Campbell [34]. In Figure 3.2 and Figure 3.3 it is possible to compare the enrollment changes. Most notably, when CS 0 was added in Fall 2017, the number of women in CS 1 in Fall 2017 and Spring 2018 dropped. It was believed that the percent women would go up in Spring 2018, but that was not the case. Furthermore, this drop continued in Fall 2019, in which the women (and men) from CS 0 Spring 2018 should have ideally registered for CS 1 in Fall 2019. At the same time, the grades in CS 1 did not go up. Instead the course averages remained the same and the DWF rate remained between 38-34%. This led to the conclusion more work needed to be done

Table 3.5: CS 0: Fall 2017 - Spring 2018 Design, a very traditional designs

Week	Java Topic
Week 1	Introduction, Computer Environments
Week 2	Primitives
Week 3	Primitives
Week 4	Exam and Primitives
Week 5	Conditionals
Week 6	Conditionals
Week 7	Strings
Week 8	Exam and Strings
Week 9	While Loops, Do-While Loops
Week 10	For Loops
Week 11	Arrays
Week 12	Exam and Arrays
Week 13	File I/O
Week 14	File I/O
Week 15	File I/O
Week 16	Final Exam

in CS 150, to make it both a course that recruits students and increases the performance for students who take CS 1.

3.2.1 CS 0 Redesign: Spiral Design

When looking at CS 0, the department had to ask themselves what was the primary limitation of students moving from CS 0 to CS 1. The issue came down to lack of interesting topics, as discussed in the interest based versions of CS 0 in Section 2.2.4. Additionally, in the State of Colorado, to be part of the General Education requirements for a university (called All University Core Curriculum at CSU), you also have to fall under the guaranteed transfer pathways, which allows transferring of General Education credits between universities. It was believed by getting into that group, it would attract more students, which could add more students taking CS 1 and CS 2.

This meant any redesign needed to follow three primary concepts:

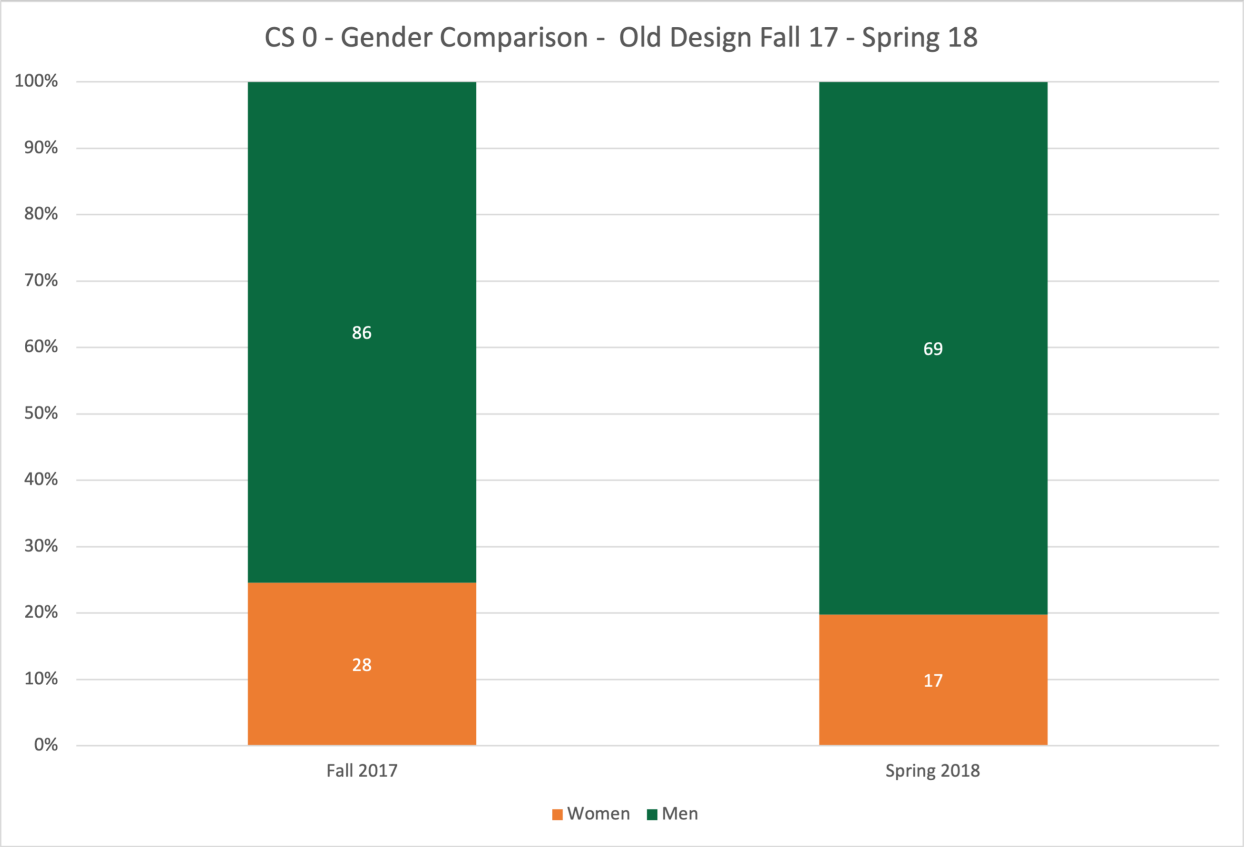


Figure 3.2: CS 0 (150) enrollment during Fall 2017 and Spring 2018, following a traditional CS 0 design.

1. Interest Based: CS 0 needed to be more based on inspiring students, and while attempts were made in the past design (often through movie references), that is not the same as designing around an interest topic.

2. Similar Concepts: Any redesign had to keep the same topics that were already setup based on department requirements. Basically, the first 8 weeks of CS 1, needed to be covered in CS 0.

3. GT-Pathways: Any design needed to fall under a GT-Pathway. While there are a variety of pathways, Different Ways of Thinking was selected as the most ideal fit.

GT-Pathway: Different Ways of Thinking:

While there are a number of different GT-Pathways, the goal was to fall into the Arts and Humanities within the university. Across the University categories, this was the best place, as

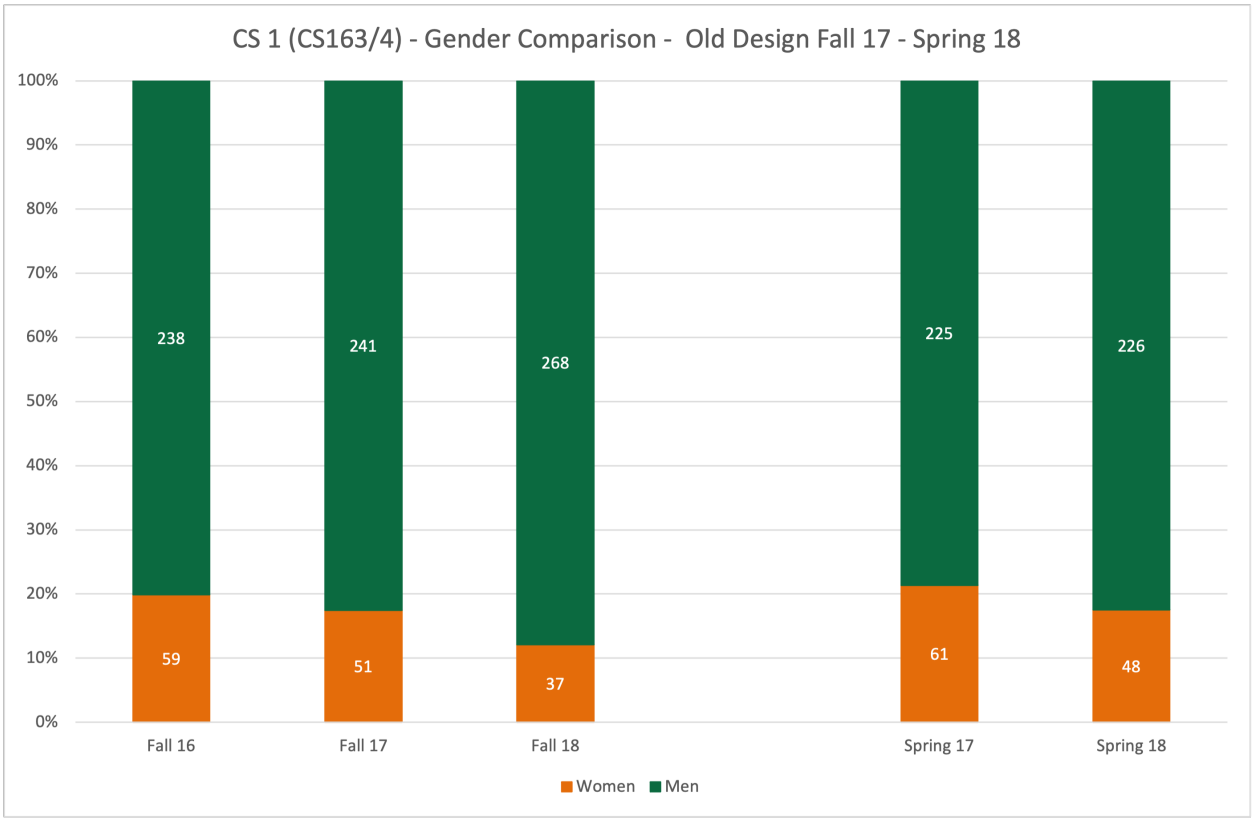


Figure 3.3: CS 1 (163 and 164) enrollment during Fall 2017, Spring 2018, and Fall 2018. Enrollment is affected by CS 0 enrollment, ideally going up Spring 2018 and Fall 2018, but enrollment actually decreased.

at the earlier levels, the program is focusing on teaching ways to look at problems, and how to think about problems. The GT-Pathway Different Ways of Thinking has the following learning objective requirements that had to be implemented in our CS 0.

1. Critical Thinking

   (a) Explain an Issue

   - Use information to describe a problem or issue and/or articulate a question related to the topic.

   (b) Utilize Context

   - Evaluate the relevance of context when presenting a position.
   - Identify assumptions.
   - Analyze one's own and others' assumptions.

(c) Understand Implications and Make Conclusions

- Establish a conclusion that is tied to the range of information presented.
- Reflect on implications and consequences of stated conclusion.

2. Diversity & Global Learning

(a) Build Self-Awareness

- Demonstrate how their own attitudes, behaviors, or beliefs compare or relate to those of other individuals, groups, communities, or cultures.

(b) Examine Perspectives

- Examine diverse perspectives when investigating social and behavioral topics within natural or human systems.

3. Written/Oral Communication

(a) Develop Content and Message

- Create and develop ideas within the context of the situation and the assigned task(s).

(b) Use Sources and Evidence

- Critically read, evaluate, apply, and synthesize evidence and/or sources in support of a claim.

(c) Use language appropriate to the audience

Additionally, internal to CSU, there is the requirement that 25% of the grade comes from written essay style assignments that include directed feedback on ways to improve the writing. To accomplish this task, topics listed in Table 3.6 were introduced, all of which had a basic writing component to them.

It was possible to include the Philosophy and Ethics topics across the other four topics, which left four major topics of discussion: History, Inclusive Design, Internet/Networking, and

Table 3.6: Culture based concepts added to CS 0

Topics	Description
History	CS history and key individuals with a focus on women's contributions to the field.
Inclusive Design	Moral and Ethical considerations when designing programs for a wide range of audiences
Philosophy & Ethics	Common philosophical and ethical questions that arise in computer science, such as what it means to be intelligent, syntax verses semantics, understanding language, and ethics problems such as the trolley problem.
Internet	History of the Internet, HTML, basic networking protocols and layers
Security	Basic computer security and data privacy. Ethics of data ownership.

Security. Between the coding topics and the culture topics, that is a lot to cover in a semester. As the previous model of the CS 0 course was to take a full sixteen weeks on the eight programming topics, it became imperative to look at not only the topics, but also structure the topics in a way where less instruction still encourages higher student retention.

Solution: Spiral Design

Psychology of Learning cites the importance of spacing topics to increase recall which is detailed in section 2.4. As CS 0 is an introductory level course, it was important to build into the course study techniques for students, using learning psychology to the student's advantage. This meant the pedagogy was based on splitting up the eight programming topics into smaller parts, and then reiterating over each topic, going more in depth on the topic with each iteration. This "spiral" allowed less time spent on topics but with greater recall on each iteration. These topics were then combined with the culture topics, with a focus on the coding assignments informing the student's understanding of the culture topic. The curriculum for the course is laid out in Table 3.7.

A traditional teaching model that covers the entirety of a topic before moving on, rarely builds in the iteration needed that is required by the spiral design. The spiral design allows for a

Table 3.7: Spiral Design: Spacing topics for better recall

Week	Culture Topic	Coding Topic
Week 1	History	int, double, boolean, char, System.in/out
Week 2	History	Methods, Basic if/else
Week 3	History	Strings
Week 4	Exam	Loops (for, while)
Week 5	Inclusive Design	Complex Boolean Logic
Week 6	Inclusive Design	String Operations, Methods Continued
Week 7	Inclusive Design	Loops Part 2
Week 8	Exam	Additional Review
Week 9	Internet / Networking	Other Primitive Data types, Binary
Week 10	DNS and Routing	Methods and Unit Testing
Week 11	Net Neutrality	Arrays
Week 12	Exam	Additional Review
Week 13	Operating Systems	File I/O
Week 14	Security Concerns	Recursion
Week 15	Review	Review
Week 16	Final Exam	Final Exam

reduction of time spent teaching coding topics, so there was room to introduce new topics. For example, when introducing primitives, books and most classes introduce all primitives along with a discussion of memory. The Spiral Design introduces int, double, boolean and char; the basics needed to build a program. Very basic memory is discussed, but an emphasis is placed on simply programming. After 8 weeks of practice with coding, and multiple discussions on memory, introducing the other primitives then went from an entire lecture, to a 15-20 minute discussion in week 9 as part of a discussion of memory management. As students had already seen and used primitives before the discussion, less time was spent on both their introduction, by introducing a smaller subset, and their follow-up discussion as students had a better understanding going into the discussion. It was also possible to group the course into 4 week units, each focusing on a different cultural topic, while also changing how the information is presented. For example, the first time students see code, it is often syntax focused, but the second time they see it, they get more visual representations of that code as they have already had practice with the syntax.

Spiral Method: Assignment Design

Students had multiple types of assignments which helped with evaluation. Many of these assignments were designed to teach students good study habits, and also link programs to real world problems. A focus was placed on providing students with meaningful assignments, more than simple and fun assignments. The assignments are broken into the following categories with examples. It should be noted that in the design, students spent two days in lecture and two days in lab at 50 minutes each. Lectures focused on culture topics with about 10 minutes towards coding, while labs focused on coding with about 10 minutes relating back to culture topics. Essentially 100 minutes each week were focused on coding, and 100 minutes focused on teaching culture topics. This is compared to the 180-200 minutes per week focused on coding in the previous course design.

- **Daily Interactive Textbook Readings**:

  Due before each class, students were to complete daily interactive readings in a programming text book. These readings were due and graded before the start of class, and often would take 10-20 minutes to complete.

- **Lab Assignments**

  Lab assignments were about 40 minutes of the 50 minute lab and due at the end of the lab. They would focus on a related programming topic, and the content of the lecture. For example, when computing and cryptography in World War II was discussed, the students programmed a very simple Caesar Cipher in lab with references to the Bombe machine.

- **Practical Programming Assignments**

  These were take home assignments due every two weeks. They were challenging assignments that were parts of a larger program. They were related to class content, and students had two parts to each practical assignment: an auto-graded coding assignment, followed by a reflective writing piece based on the class discussions and programming assignment. For example, students wrote code to help analyze pay differences across

genders and ethnic backgrounds in industry, and their writing focused on salary inequity based on the analysis they generated.

- **Discussion Assignments**

  Every other week, students had a discussion essay style post. The question was based on the cultural topics in class that week, and students were required to cite sources to support ideas. After posting, students were required to respond to other student posts.

- **Final Project: Data Analysis**

  For their final project, they would pick one data set to analyze from a number of provided data sets: temperature change data, major demographics across departments, glacial melt, and endangered species lists. They were encouraged to pick a set that (1) interested them, and (2) ideally related to their primary major. Students then worked in pairs, but not on the coding side. Instead, they were required to pick two different data sets, and each student had to write code to analyze that data set. They then had to write about both data sets, encouraging discussion with their partners to learn about the different analysis they did. Their grade was based on how well they could describe the data sets, with a section about what students shared in common and different in their code.

- **Exams**

  Four exams were required, all equally weighted. The first three exams contained questions on both coding and culture topics, and the final exam was only coding based. The final exam was used across multiple class iterations and designs (both the traditional method, and the new curriculum) as a means to gauge student coding competency at the end of the class.

Design-Conquer-Glue

Throughout the entire class, there was an emphasis on problem solving, including the occasional logic problem in class to teach algorithmic thinking. Throughout this process, the Divide-Conquer-Glue (DCG) problem solving strategy is encouraged as mentioned by [112].

DCG focuses on presenting a problem, dividing it into smaller parts, solving those parts and reassembling the parts to solve the problem. To teach DCG, It became necessary to present methods early on in the course, something the previous course model skipped entirely. Most assignments were tested on a method by method basis, designing them so each method was a different, but related problem to solve. Student work focused on the conquer part to start, and slowly into the divide and glue aspects of programming. With the culture discussion, the same methodology was used when trying to understand the topic. This focus led to a very specific "different way of thinking" when presenting and understanding problems. It was found the discussion easily crossed disciplines, including how to word and write essay papers for the students who were needing development in those skills.

3.2.2 Concepts Not Changed

Between the old design, and the new Spiral Design, there were a number of factors that were not changed. Most notably, the time of day and room remained the same, even though it was a different semester. Furthermore, both courses used zybooks for the coding questions, and most of the TAs were the same TAs between the semesters. In the end, the same final exam was used as a means of evaluating their coding proficiency when students completed the course, as department requirements wanted them at a certain coding level.

3.2.3 Evaluation Methods

While the course maintained the general education requirements for the university with the cultural topics, it was also a priority to maintain a set number of topics for our majors. This means students needed to perform just as well at coding tasks from the old design to the new design. As such, the evaluation was twofold.

- **Performance Comparison**

  Less time can be spent on programming with equivalent student performance based on final exam score comparisons. Additionally, performance in CS 1 was also evaluated.

- **Retention Goals**

  If the goal of the new design was to attract students, it is essential to make sure numbers of targeted student groups (women) are increasing relative to the CS 0 influence.

It was determined that if students were performing equivalent, it would be considered a success for the department, as equivalence means:

- 50% reduction in teaching coding topics
- Addition of culture / interest topics
- Similar results in coding topics / similar goals met even with reduced teaching time.

It would be a very lofty goal to reduce the amount of time teaching a topic, and expect increased results.

3.2.4 Retention Comparison: CS 0 Old Style vs. Spiral

As retention was the primary motivation for the CS 0 redesign, looking at retention is measure of success. Does increased enrollment in CS 0, show increasing enrollment in CS 1?

It was found that more students moved onto CS 1 with the Spiral Teaching model than with the old model with about a 10% increase in the number of female students who moved on. For Fall 17 (Old Model), there were 86 male students and 28 females students in CS 0. Of those students, 31 males and 9 females went onto CS 1 in the Spring semester. For Fall 18 (Spiral Model) there were 77 male students and 35 female students. Of those, 31 males students moved on, and 15 female students moved on. In both cases, the percentage of students out of the total students increased in the Spiral Model, with results shown in Table 3.8.

When looking at the numbers across multiple years, Fall CS 0 should be compared to Spring CS 1, and Spring CS 0 should be compared to Fall CS 1, even though Summer offerings of CS 1 reduce the transition numbers. Immediately after implementing the new CS 0, there was an increases in CS 1 - Spring 2019, repairing the loss of numbers that were seen when CS 0

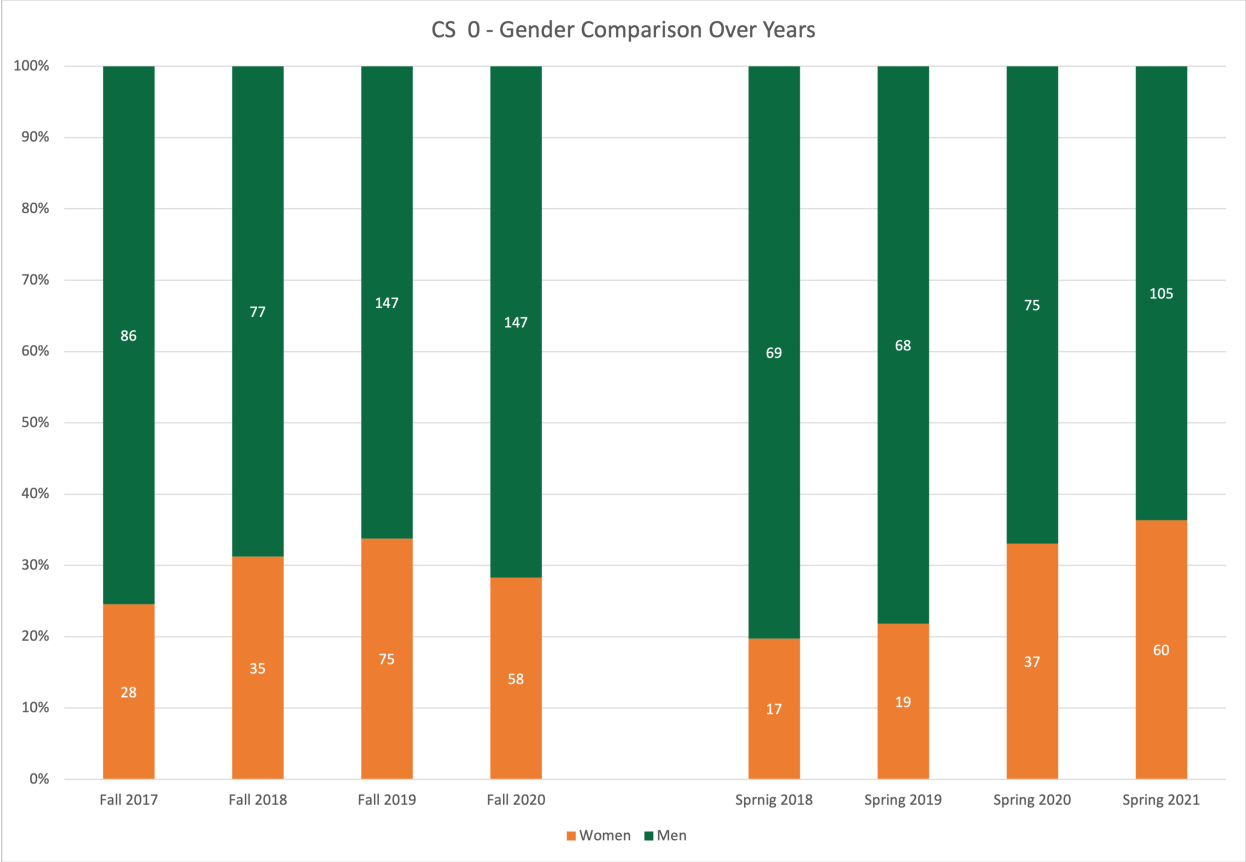


Figure 3.4: CS 0 (150) enrollment over the years. Increasing Spring enrollments are related to non-majors taking the course.

Table 3.8: Students transitions from CS 0 to CS 1

	Men Total	Women Total	CS 0 Men Taking CS1	CS 0 Women Taking CS1
CS 0 - Fall 17	86	28	31 (36%)	9 (32%)
CS 0 - Fall 18	77	35	31 (40%)	15 (42%)

was first implemented. When looking at Spring CS 1 numbers, a couple points to take into account. Spring 2020 numbers are lower than they were in practice. Due to COVID, a number of students were allowed to free drop the course, removing them from the number counts. Second, about 1/3 of the Fall students are students who do not have the math requirement to get into CS 1. As such, it is assumed at least 1/3 will move on, as CS 0 can substitute in for the math requirement. This makes the Fall to Spring numbers harder to narrow down on success, though we do see increased enrollments over Spring 18, the first semester paired with CS 0. What is

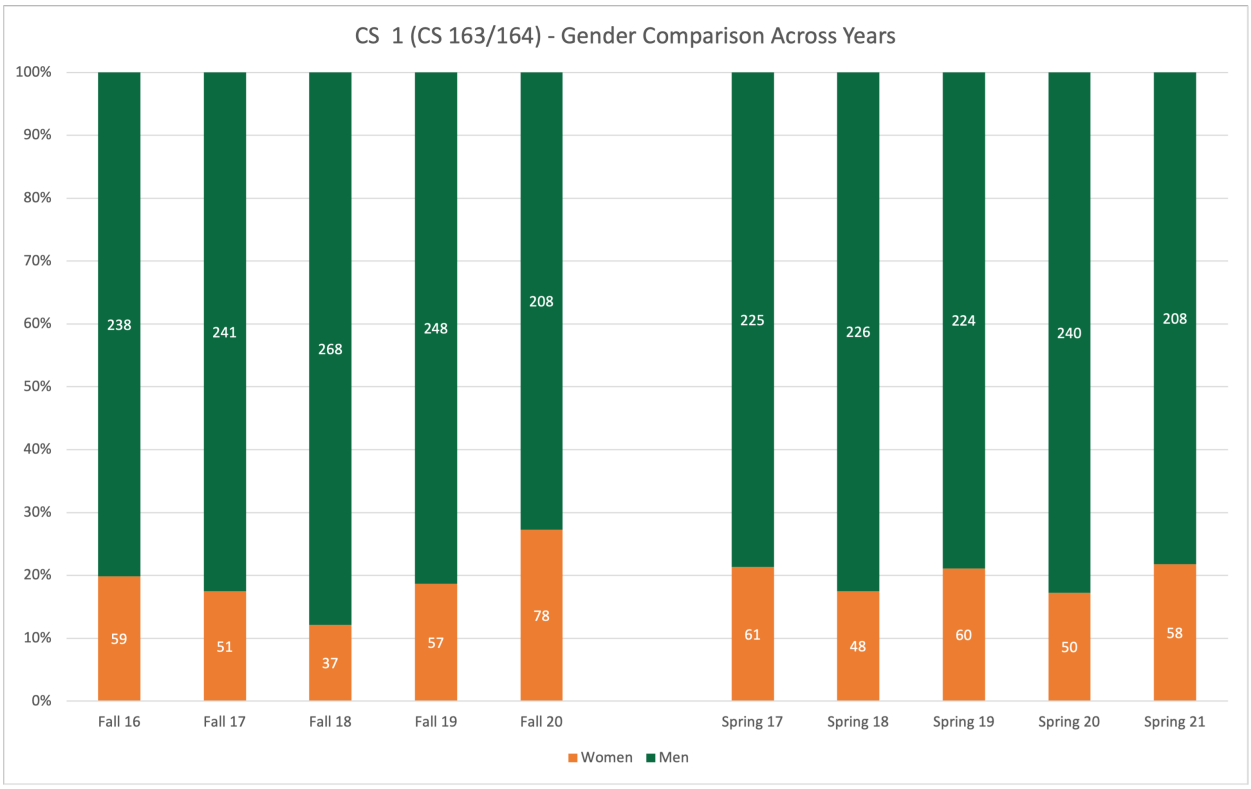


Figure 3.5: CS 1 (163 and 164) enrollment over the years. Most noticeable is the distinct increase in Fall enrollments related to an increase in CS 0 Spring enrollments.

more noticeable, is the CS 0 Spring to Fall CS 1, there is a direct increase both in CS 0 and CS 1, meaning more non-majors are opting to take CS 1 as very few CS majors take CS 0 in the Spring semester.

Overall, CS 0 continues to act as a means to recruit non-majors, and provide an introduction to CS students who lack the math background to go directly into CS 1. The question then focuses on performance. Do students continue to perform as well in CS 0 under the Spiral model?

3.2.5 Performance Comparison: CS 0 Old Style vs. Spiral

Final exams in the 1st year level courses (freshman level / 1xx level) in the department are computer based, taken in a controlled environment. Students are required to take the exams on department computers, with heavy system lock downs and multiple TAs patrolling the room. Exam answers or questions are not released and can be used from year-to-year. This established

a clear basis for comparison, requiring students to take the same final no matter which version of the course they took.

Based on exam results, it was found that students all had a similar level of proficiency. This was compared across Spring 18 (old-method), Fall 18 (spiral teaching) and Spring 19 (spiral teaching). With 99 students in the old-method and 200 students taking the spiral teaching based course. The average score for the final exam for the old-model of teaching is 69.8% with the median score being 73%. The average score for the final exam for the Spiral teaching model is 72.1% with the median score being 76%. This shows roughly a 3% difference in scores. Fall and Spring spiral teaching based semesters were within 1% of each other. Results can be seen in Figure 3.6. However, the results are not statistically significant as the p-value is 0.16. While this is not definitive in that the students are performing similar, it is a good indicator that students are testing the same at the end of either model of teaching. However, it should be noted that due to the reduction in teaching time allotted to specifically coding, students performing at a similar level is notable.

To add to the performance comparison, it was determined that looking at student performance in the following CS 1 course, would be a good way to measure students against other CS 0 type courses, as most all students in CS 164 have had a CS 0 course of some sort, and many have had a CS 1 variation just in a different language. This gave the opportunity to compare the spiral design students to the other students. It should also be noted that in the traditional design there was not a difference between students in CS 164.

Looking at Figure 3.7 one can see that CS 0 Students who learned via the spiral design out perform students who learned via more traditional models of teaching in **every** exam. While spiral teaching based students out performed other students between 3-6% on the exams, the most notable difference was their final score for the class, which includes all programming assignments as they performed 8.9% better than their counter parts. The final grade was significant with a p-value of 0.008 where $p < 0.05$. The median grade difference is more pronounced with a median grade of 87.6 compared to 78.3, giving a 9.5% increase in grades compared to the

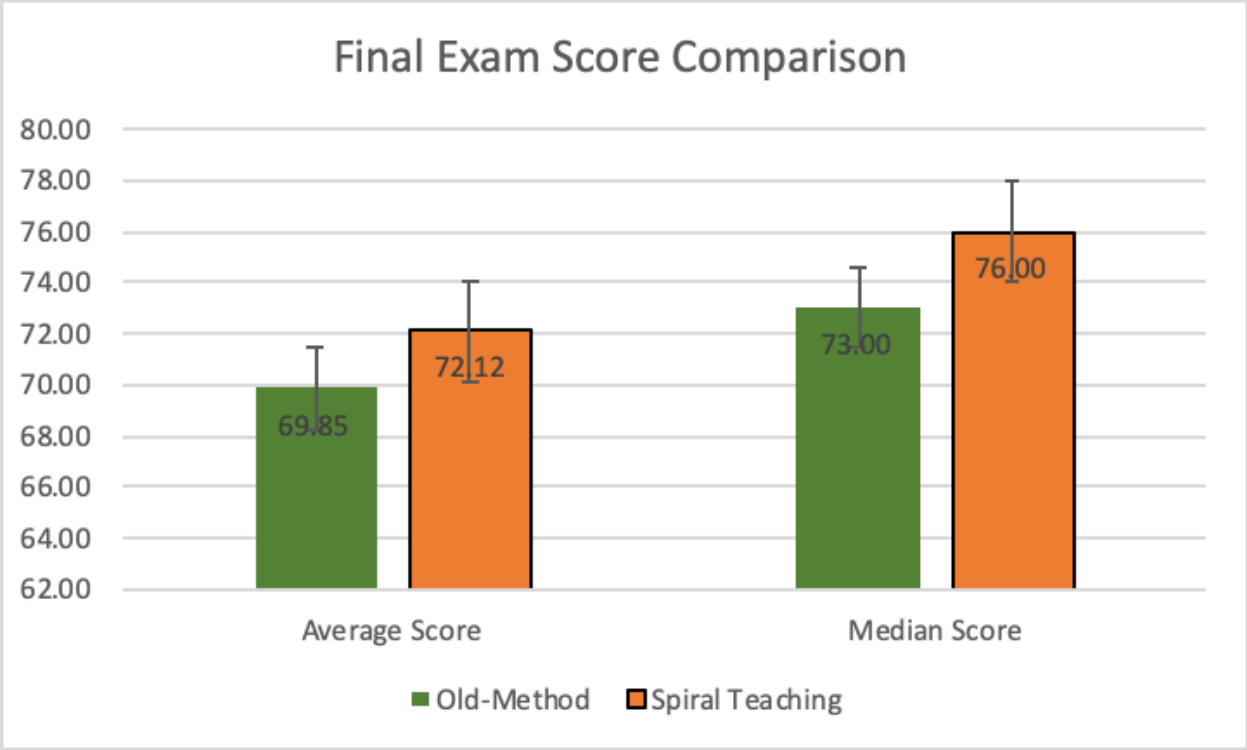


Figure 3.6: Comparison of Exam Scores in CS 0 between Old Model and Spiral

other students. This shows the students who learned via the spiral model are out performing the other students.

This increase in CS 1 scores is surprising relative to other students who also had CS 0 courses as traditional studies point out the style of CS 0 doesn't matter, just that they simply have a CS 0 course [4, 56]. Furthermore, our own research shows that usually by the second semester of taking a course, students even out [3].

The results from the Spiral Model lead to the primary question: *does the spiral model increase performance in the following course, showing lasting retention of material over the traditional structured model?*

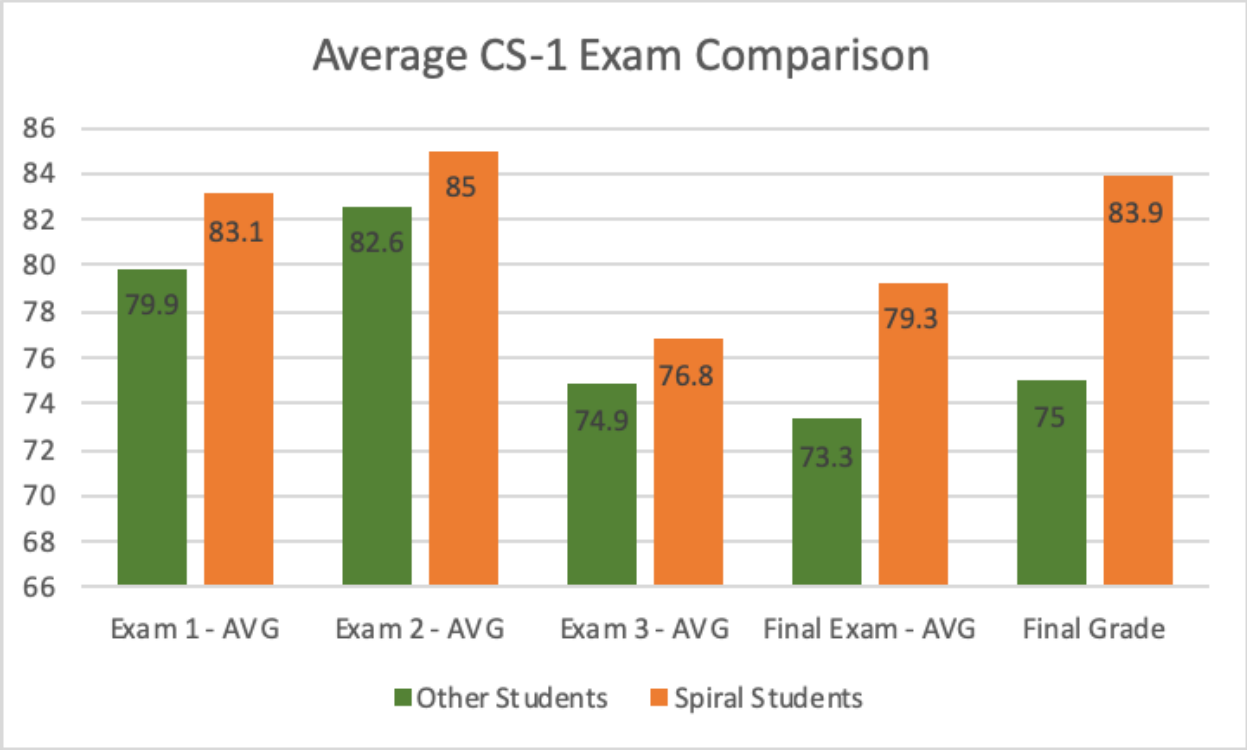


Figure 3.7: Comparing Students who took CS 0 with the Spiral Teaching Model with students who learned in other programs.

Chapter 4

CS 1: A Spiral Model Design and Evaluation

With the success of the Spiral Model of teaching, it leads to three salient questions that we seek to answer.

- Q1: Given that going to spiral allowed us to add content without reducing learning, what happens when CS 1 is designed around the Spiral Model without adding content? Will student performance in the current course increase?

- Q2: Given CS 2 is often the great equalizer in student performance no matter their programming background, will students who learned via the Spiral Method in CS 1 outperform students in CS 2 who learned with the traditional methods in CS 1?

- Q3: Any modern changes should not hurt inclusion and retention of students between courses, does the Spiral Method at least encourage equal, if not improved retention of students who come from underrepresented backgrounds to continue onto CS 2 as compared to the traditional method?

In this chapter, the experiment proposal is outlined, with detail given to the design of each of the course types. The goal is to teach both course types side-by-side in a single semester, and then measure performance of students in both CS 1 and CS 2 the following semester. Section 4.1 details the design of the traditional CS 1 as it is currently taught at CSU. Section 4.2 details at length the changes made with the Spiral Design. Section 4.3 details the process of evaluating the two different types of courses, including both the fixed and dependant variables in the process.

4.1 CS 163 Green: Traditional Design

The traditional design of CS 1, follows very closely the design done by Chris Wilcox in 2016. Over four years, there have been modifications in some of the content, moving some of it to

Table 4.1: Schedule of topics for Fall 2020 Course Syllabus. Changes from Fall 16 are bold.

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

CS2, and in return, moving some CS 2 content to CS 1. The most notable changes were a recent update to lab environments, and adding more interest content. The course was also adapted for a hybrid design due to the influence of COVID-19. For simplicity purposes, the traditional design will be termed **Green**.

Table 4.1 details the weekly schedule for the Green section. Similar to the Fall 2016 design, it goes roughly in the order of the Liang book [111], though new slides had been generated for the course and it made more use of Zybooks. Liang's book was marked as an optional book for that reason even though the order matched. The new slides that were generated were a joint effort between instructors to bring more active learning and peer instruction opportunities into the course, along with adding some "interesting topic" discussions that matched the departments' six concentrations.

In addition to reorganizing the topics, the presentation of the course was adapted over the years to make it easier for students to find topics, including making use of module layouts in the Canvas LMS.

4.1.1 Assignment and Exam Modifications

Over the years for the Green section, the assignments underwent modifications. Most notably, they adopted many of the zybook zylabs as part of the many small programs approach that was emphasized with the traditional design. Additionally, a student was hired in Summer 2018 as part of a Center for Learning and Teaching fellowship fund to add question banks for the quizzes. The student helped setup question banks, and mapped banks to learning outcomes and topics in the course.

Exams were scaled each unit (4 units, 4 weeks each) so that the first exam was worth less points with increasing points until the final exam. They also added a zybooks lab as a coding quiz; single submission attempt, as compared to most labs that had unlimited times to submit for grading. Review weeks were added to help facilitate studying for exams.

4.1.2 COVID-19 Modifications

In order to handle social distancing with a large class, the following changes were made with the course.

- Pre-recorded lecture videos

  The lectures were pre-recorded in shorter chunks, and students had full access to the lectures all semester.

- Reading Assignment Due Dates

  Assignments were modified to have flexible late windows to allow for students with other commitments.

- On-Campus Labs

  Labs were still run on campus, but limited to 12 students per room.

- On-Campus Help Sessions

  Students would attend an on-campus help session once a week led by the TAs. This would be an opportunity to participate in peer instruction activities and discussions.

- Online Help Desk

  In place of the extensive on-campus help desk, it was run online via Microsoft teams. Hours were slightly modified to include evening hours which were not included in the past.

Overall, while the hybrid modifications seem extensive, the course simply sought to follow best practices for building community while still maintaining social distancing practices.

4.2 CS 163 Gold: Spiral Design

The Spiral Design is a redesign from CS 1 basing the design first on the the five techniques to improve memory recall presented by Roediger [84]. As these techniques are designed around improving student recall, the course design seeks to build on these techniques as the foundation of the course. The belief is that while one can't memorize how to code, the recall of topics they have seen helps them construct their solution again, while reducing the intimidation factor of not knowing where to start on the problem. The five principles and how they were directly implemented into the design are as follows:

- Spacing

  The spiral design purposely defines "what not to teach, yet", so students can come back to the topic later going more in depth. Usually the spacing gap is 2-4 weeks before explicitly returning to a topic.

- 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 knowledge checks or "short weekly quizzes" that students were encouraged to retake and use both as way to read code, but also practice recalling information.

- Elaboration

  Within the spiral design, elaboration shows up in two areas. The first is the active learning approach of peer instruction and discussion. The second are the practical projects that cause students to look at what they are learning in the larger context of industrial style applications, and to think about the direction they would want to take in CS as the projects are themed.

- Reflection

  One to two times a unit, the spiral design encourages students to reflect on what they are learning,and how far they have progressed over the semester. This is done at the end of every practical project.

Each aspect of the design is detailed in the following sections.

4.2.1 Topic Ordering

Similar to CS 150, the topics were reordered to encourage recall. Unlike CS 150, 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 4.2. 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 Table 4.1 and Table 4.2 is that due to covering fewer topics, the Gold section was able to visit topics sooner, visiting loops before the first exam. However, in the long term, the Gold section took a bit longer to get to Interfaces and Abstract Classes seen in Week 13 compared to Week 11. The Gold section also had two additional weeks of review week 4 and week 15. These review weeks ended up as catch-up weeks for students.

Table 4.2: Schedule of topics for Gold Section of CS 1 (Spiral Design)

Week	Topic	Practical Project
Week 1	Fundamentals (computers, variables, simple types)	-
Week 2	Objects, Methods, Strings, Conditionals	-
Week 3	Loops, Code Tracing	Practical 1
Week 4	Review and Exam 1	Practical 1
Week 5	Data Types, Classes, Common Classes	Practical 2
Week 6	Logical Operators, More Loops, More Methods	Practical 2
Week 7	Arrays, File Input	Practical 3
Week 8	Review and Exam 2	Practical 3
Week 9	File Output, Exceptions, More Classes	Practical 4
Week 10	More Branching, 2D Arrays, Introduction to Recursion	Practical 4
Week 11	Inheritance, ArrayList, Unified Modeling Language	Practical 5
Week 12	Review and Exam 3	Practical 5
Week 13	Abstract Classes and Interfaces, Polymorphic Life	Practical 5
Week 14	Recursion, Searching and Sorting, Collections	Practical 5
Week 15	Review Week	Practical 5
Week 16	Final Exam	-

Covering a topic, students had three majors parts that were meant to be done in order: Reading, Lecture, and Knowledge Check. They then followed it up with a lab or practical project which started in the lab environment. Figure 4.1 shows an example of what students saw for each week in Canvas. 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.

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

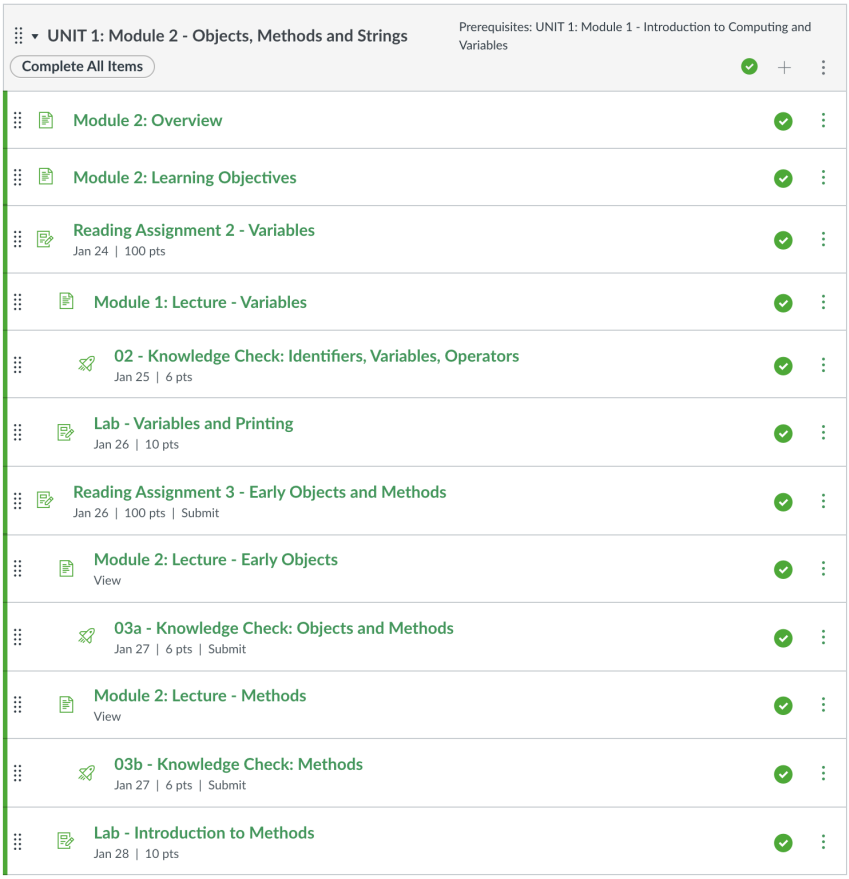


Figure 4.1: Example of a week of work in the Gold Section, Fall 2020.

4.2.2 Reading Assignments

The students were assigned reading assignments each week. For the Gold section, students had 2-3 reading assignments each week. It was found during the design of the Spiral Design that the book was a limiting factor as many books assumed prior knowledge when building chapters. For example, in the traditional zybooks, it assumed branching was covered in its entirety before methods, meaning they could level a discussion about switch statements calling different methods. This does not work in the spiral design as switch statements are not covered until the third pass with branching. Both the green section and gold section used the "one chapter" per reading assignment approach. For the green, they had an entire chapter a week, causing 15 chapters for the entire semester. Where as the Gold section had much smaller chapters but more of them. Overall, roughly the same amount of content was covered, just the length on each assignment varied. Figure 4.2 shows the two zybooks table of contents side by side. The right side is the book restructured for the Spiral design, and the left side is the traditional design, following the mostly traditional layout for the book optimized for the specific goals of CSU. For both sections, the readings were assigned before lectures and labs.

Overall when designing assignments careful attention had to be applied to what was actually taught before each section. It is easy to assume prior coverage of topics, and then find out they were not covered. This proved to be particularly difficult for file reading, as the sections in zybooks assumed knowledge about arrays and exception, and those sections assumed knowledge about files. In practice, the exception and file reading and writing sections are all taught in one chapter, but when attempts were made to break them up, there was an issue in how they were laid out. The only possible fix is to actually design a new book, so for this research the focus was getting the book as close as possible. TAs and instructors would attempt to fill in any knowledge gap with students, and then encourage them to go back and finish the chapters the following week.

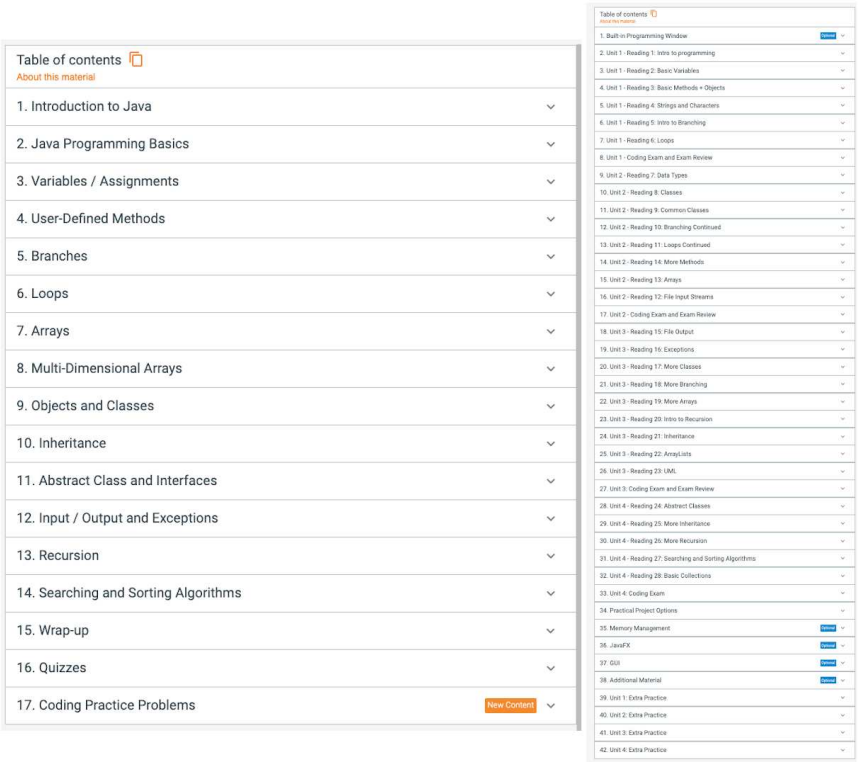


Figure 4.2: Zybooks was used for both classes, with noticeable restructuring for the Spiral Design (right side)

4.2.3 Lectures

In order to promote active learning, the lectures focused on 5-10 minutes of instruction followed up with peer activities and discussion. Activities were designed around going over knowledge check problems as teams, code reviews, and discussion topics. During a typical lecture, there may be 1-3 lecture portions, with similar peer instruction activities. During Units 1 and 4, the two major topics / lectures were planned for each week on Monday and Wednesday, with the third Friday lecture reserved for tool specific discussions. For Units 2 and 3, most weeks all three lectures were needed to cover topics. Furthermore, topics were intermixed with some "interesting topic" references. For example, talking about Ada Lovelace during the first lecture, or some of the history of who did what in programming throughout the years. They were almost always focused on diversity topics or CS for the social good, but they were never the primary topic of a lecture. In an ideal world, teaching assistants would be spread throughout the room,

to help facilitate discussion and code reviews. They are often called learning assistants at our university when used in such a situation.

The primary goal of the peer instruction focused lecture times is to promote active learning and **elaboration**, and to avoid the "sage on the stage". Between the Green and Gold sections, this same model was followed but with different topics in the lectures. Overall, the lectures were affected the most with the modifications needed for COVID-19 for both sections. Section 4.2.7 details the changes made to the Gold section.

4.2.4 Knowledge Checks

Using a question bank feature in Canvas LMS, 4-10 questions were developed for each of the topics covered, 3-5 for weekly quizzes and the remainder for exams. These question banks were then linked to learning outcomes for the courses, which matched every topic area. The outcomes were listed as follows and were aligned with the topics shown in Table 4.2. The naming convention matches a university naming convention of College->Department->Course(s)->YearCreated->Topic. In our case, we made the topic java and further divided into subsets of the language. These topic areas were also used for CS 150.

- NS.CS.ALL.2020.JAVA.ALGORITHMS

  Covered in the last portion of the class, with a focus on sorting and searching algorithms.

- NS.CS.ALL.2020.JAVA.ARRAYS

  Focus on arrays both single and multi-dimensional.

- NS.CS.ALL.2020.JAVA.BRANCHING

  If/else statements, complex conditional statements (and/or), switch statements, and conditional statements.

- NS.CS.ALL.2020.JAVA.COMMON

  Common Java SDK libraries, including String and Math APIs.

- NS.CS.ALL.2020.JAVA.FUNDAMENTALS

  Types, expressions, basic printing

- NS.CS.ALL.2020.JAVA.INHERITANCE

  Inheritance, Abstract Classes, Interfaces, Polymorphism

- NS.CS.ALL.2020.JAVA.INPUT_OUTPUT

  File reading and writing, exception handling, input/output streams.

- NS.CS.ALL.2020.JAVA.METHODS

  Methods both static and instance, parameters, and pass by value.

- NS.CS.ALL.2020.JAVA.OBJECTS

  Objects and classes.

- NS.CS.ALL.2020.JAVA.RECURSION

  Basic recursion.

- NS.CS.ALL.2020.JAVA.REPETITION

  The four types of loops in java (for, for-each, while, do-while), along with nesting loops and complexity of too many nested loops.

While topic areas are not required for a Spiral Design, it makes the layout of the knowledge checks easier, and allows an instructor to see growth in different topic areas. For example, the question bank *KQ: 01 Intro To Programming* had three questions to it, and mapped to *NS.CS.ALL.2020.JAVA.FUNDAMENTALS*. Then four weeks later, students started getting quizzed on questions from the question bank *KQ: 07b Data Types* which was also mapped to *NS.CS.ALL.2020.JAVA.FUNDAMENTALS*. Beyond the scope of this research, question bank mapping could be used to help explore mastery of topics and mastery grading styles. For this research, it was added simply as a means of tracking progress and making sure each topic was covered more than one time.

The questions from the knowledge checks focused heavily on reading code, and the labs and practicals were focused on writing code. Students had fill-in-the-blank (what does this code print out), multiple choice, matching, and true/false style questions. These questions matched the format of the exams, and where possible, matched up with the quiz banks developed for the Green section. However, due to the spiral model, it was impossible to duplicate all directly, and new questions had to be created.

Knowledge check quizzes are meant to be asked at least once a week, pulling from the topics discussed that week. The questions are also encouraged to be shown in lectures, tried in groups, and discussed, as the importance of knowledge checks was not getting the right answer but simply taking them. Students were allowed to take the knowledge checks as many times as they wanted, and a lecture was devoted on the first day of classes about using the knowledge checks as a means to study for the exams. This same lecture talked about the benefits of spacing and interleaving, along with practiced recall, and students were reminded throughout the semester that the best way to study was to go back to past knowledge checks and intermix the ones they take each night.

The primary goal of the knowledge checks was to encourage an environment that promotes **Practiced Recall**. While the Green section had practice exams students could take, the knowledge checks were much more in depth, and a major difference between the two types of teaching styles.

4.2.5 Labs and Practical Projects

Students had one to two labs a week for three weeks, and then would have the fourth week of a unit as 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. Labs were designed by a team of undergraduate students, and then the TAs for the course had the ability to modify the wording as the

semester progressed. The labs all focused on having around five parts for the students to complete. Each part was broken up into a separate method, and the methods were graded using the Zybooks auto-grader and unit testing. For the five methods, the typical pattern was as follows:

1. Method 1: Review - Review a past topic and how it ties into the lab.
2. Method 2: A simple case of the topic they were emphasizing in the lab.
3. Method 3&4: Methods that were often small, but tied into a larger program they were writing.
4. Method 5: A method that utilized the other methods to make a working program.

This pattern was followed so that labs were treated as a series of problems on the same subject being covered. As the semester progressed, the methods and work progressed in difficulty, especially with each pass of the same topic. An emphasis was placed on problems that involved the students having to figure out a larger problem, but the code itself was relatively simple to implement once the problem was solved. The labs were written like "tutorials" walking students through the process.

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 that they were able to work on. As to not make the practicals overly difficult, practicals 1-4 had three parts to them.

1. Code Tracing - Students were given a canvas quiz on the provided code or other concepts they would need to know about the code they were supposed to write for the practical (e.g. HTML for the HTML web-page generator). They could submit this up to three times, and they were encouraged to take the quiz in the lab where the practical was assigned. TAs were allowed to help explain questions on the quiz. 1/4 of the practical grade was awarded to the quiz.

2. Code - The actual coding portion of the assignment. 70-80% of the code was focused on getting the correct syntax (e.g. accessors and mutators), with the last 20-30% being focused on a unique, but essential problem the code needed to solve in order for it to run, like writing a substitution cipher. 1/2 of the practical grade was awarded for the code portion.

3. Reflection - Students were asked to reflect on the practical project, and also anything else they wished about the class. As the practicals were spaced out every two weeks, this gave an opportunity for them to reflect every two weeks. They were graded simply for completing a reflection, and not the content though the instructors gave feedback based on the content. 1/4 of the practical grade was assigned to the reflection.

Practical 5 was slightly different than practicals 1-4. The goal of practical 5 is to transition onto harder assignments they will be expected to solve in future courses. In Practical 5, they are provided with a UML diagram and full documented Javadoc of the finished assignment. They are then asked to redesign Practical 1 using all the Object Oriented Techniques they learned throughout the course. Overall, the practical assignments were designed to seem daunting until they apply the divide-conquer-glue methodology, focusing only on the one aspect they were working on each moment. Table 4.2 also shows when the various practicals were assigned.

The primary goal of the practical projects was to promote **elaboration** and **reflection**. Elaboration as students were able to see programming in context of social good and larger applications. Reflection as that was an aspect of the practicals in general, even if it was a small aspect of the overall course.

4.2.6 Exams

The exams were given every four weeks, in both the Green and Gold sections. The Exams were always cumulative, though the exams had to be redone since the topics were different. When possible, exam questions were duplicated. Exams are meant to be scaled similar to the rest of the spiral where depth is added every four weeks, the exams are worth more every four

Table 4.3: Exam weighting for each component.

Exam	Item	Points
Exam 1	Practice exam	20
Exam 1	Coding exam	5
Exam 1	Exam	75
Exam 2	Practice exam	10
Exam 2	Coding exam	5
Exam 2	Exam	85
Exam 3	Practice exam	5
Exam 3	Coding exam	10
Exam 3	Exam	85
Final Exam	Practice exam	5
Final Exam	Coding exam	10
Final Exam	Exam	100

weeks. This causes the first exam to be "low" stakes compared to the final exam. This also matched the scaling used in the Green section. Each exam had three parts.

- Practice exam: Assigned at the beginning of a unit, the students are encouraged to take it as many times as they can.
- Coding exam: A single submission attempt lab. They could use resources like their book, but they were asked to not use other students or the TAs.
- Exam: The exam itself was a Canvas exam with questions similar to the practice exam and knowledge checks. Students were proctored and timed.

Table 4.3 gives the points possible for every exam, exams 1-3 equate out to 100, but the final exam purposely had 115 points to give slightly more weight to it. More importantly, the points for the final exam needed to match the points for the Green section exactly as the same exam was used for both designs. The weighting of exams from less to more was done in previous years with our department, and not directly tied to the spiral model.

4.2.7 COVID-19 Modifications

Due to the size of the class, and social distancing guidelines, there were some noticeable changes to adapt for COVID-19 regulations. Most notably, the lectures were affected, as they had to be prerecorded, and students could watch them asynchronously. Students would then attend a help-session/lecture once a week that would provide additional content, primarily the discussion / peer-instruction elements. Additionally, to motivate watching the videos, knowledge checks were assigned after each video, since they were already broken up into topics. The changes are summarized as follows, and they purposely match up with many of the changes in the Green section.

- Pre-recorded lecture videos

  The lectures were pre-recorded in shorter chunks, and students had full access to the lectures all semester.

- Knowledge Checks

  One was assigned after every video lecture with a small amount of points.

- On-Campus Labs

  Labs were still run on campus, but limited to 12 students per room.

- On-Campus Help Sessions

  Students would attend an on-campus help session once a week led by the TAs. This would be an opportunity to participate in peer instruction activities and discussions.

- Online Help Desk

  In place of the extensive on-campus help desk, it was run online via Microsoft teams. Hours were slightly modified to include evening hours which were not included in the past.

Overall, while the hybrid modifications seem extensive, the course simply sought to follow best practices for building community while still maintaining social distancing practices. Ex-

cept for the change in knowledge checks, the modifications were the same for both Green and Gold sections.

4.3 Evaluating Outcomes

At the start of this chapter, three primary questions were defined. Two questions focused on performance, and the third question focuses on student retention and enjoyment. Based on that, the evaluation measures are predominately focused on student performance. Each subsection highlights key aspects of the evaluation process. Each subsection also addresses attempts to minimize bias.

4.3.1 Student Distribution and Sampling

The courses were divided so CS 163 section 001 was taught using the Spiral Model, and CS 163 section 002 used the traditional model. They were termed Gold and Green, respectively. Additionally, CS 164 which only had one section was taught using the Gold model, but due to only having one section will simply exist as third point of comparison. Section 001 (Gold) was scheduled for 3:00 PM in the afternoon, before changes for COVID-19, and Section 002 (Green) was targeted for 11:00 AM. While Poulsen et al. point out afternoon classes often perform better [113], the instructor's experience for CS 163 was that at Colorado State University the morning section often performed better than the afternoon. Furthermore, to balance out the morning and afternoon discrepancy, the labs were setup in an alternating manner so that odd labs matched up with section 001 (Gold) and the even labs were matched up with section 002 (Green). As such, students unknowingly select their instructional design based on the time that fits best in their schedule. No distinction was made at registration time, and even the instructor was not listed at registration time to prevent students biasing themselves to CS 163 or CS 164 based on their history with the instructors.

In order to hold the influence of the instructor fixed, the same instructor taught both CS 163 Green and Gold sections. The instructor was experienced with Green, and was just learning the

Gold methodology. As a modification to COVID-19, the instructor influence shows up more as part of the online interaction, as TAs ended up teaching the different on-campus help sections that replaced a lecture. The help section times were the same as the assigned lecture times, but only one day a week with the actual lectures being prerecorded.

TAs were provided the same slides for both help sessions, though coding content had to be modified for the Green section. The help-session focused more on peer instruction and discussion of interesting topics, and students were not required to attend due to wanting to allow student flexibility in the uncertainty of on-campus interaction.

Once students started class, they were provided with the disclaimer provided in Appendix A.1 detailing the research and giving students the opportunity to opt out of data collection.

Minimizing Threats to Validity

The most common threats when teaching with different methodologies are teacher familiarity, and sample groups biasing themselves at selection time. For this study, effort was made to bias the results towards the traditional methodology, so if the spiral method proved promising it would have to overcome that bias. The instructor selected had experience with the traditional methodology, but not the spiral, and more importantly, the same instructor was used for both CS 163 sections. Student selection was essentially random, due to students not knowing which methodology they were signing up for at the time of enrollment. With that said, different student populations may have self selected to different time slots, so a question to answer is how did students perform both by section but also by student categories across sections. For example, were more honor students in Green or Gold? What about returning students compared to incoming students? What were the major distributions?

It is believed the alternating labs contributed to randomizing students more, as labs are a major factor when determining course sign up. To also bias the data slightly towards the traditional method, the earliest lab in the day (L01, at 8:00 AM) was assigned to the Spiral method, as that lab has often been shown to be a low performing lab and the last lab students sign up for (often the students who procrastinate most). This was part of the motivation of selecting

odd labs for the Gold group, along with simplicity for the instructor to remember Section 001 line up with odd labs, and 002 lines up with even labs. Overall, when taking into account time selection bias, attempts were made to bias towards the traditional methodology, as that is the "baseline" the spiral model seeks to beat.

Attempts were made to group TAs to a certain style. Due to scheduling constraints, this proved impossible to do, so a number of TAs ended up teaching both Gold and Green labs and help-sessions. It is believed this had very little influence, as TAs often are focused on providing help, rather than to talk about different teaching techniques. They were also requested to not talk about "what is happening in the other section". Twenty of the Twenty-three TAs were all taught using the traditional method, which biased their experience towards the traditional manner of teaching.

4.3.2 CS 1 Performance Comparison

Due to the nature of the spiral reordering topics, there is only one point where the Green and Gold sections line up for a comparison on performance, the final exam. As such, the final exam makes the primary point of comparing students between the two sections. Most notably, given a semester of traditional or a semester of spiral, do students perform better on their final assessment at the end of the course?

The final exam for both courses pulled from the same question banks that were linked back to topics presented in Section 4.2.4. The question banks were exam only question banks, whose answers have been kept hidden over the years, as many of the questions were used in past semesters in the traditional method. More questions were added to add more depth to the exam both in the number of possible questions that could show up for each question, and more questions the students were asked to answer. Overall, the final exam consisted of 25 questions, with point values ranging from 3-5 points. Many questions had multiple parts to them. Each of the 25 question banks ranged from 2-3 questions deep creating at least 300 different exam combinations for students to take. Some exemplar questions are provided in Appendix A.2.

The theory is that if the Gold section students performed better on the final exam compared to the Green section students, then they were able to learn and retain more than the Green section.

Minimizing Threats to Validity

The most notable threat is the nature of using an exam as the sole assessment for performance. Unfortunately, due to the different nature of the courses, this is the only unifying point between Green and Gold layouts. The strength of this assessment is that it matches what was done in CS 150's evaluation as shown in Section 3.2.5, and due to the cumulative nature of the exam, it is an indicator that shows what was expected of students no matter the course design. This threat is minimized by looking at student performance in CS 2, detailed in Section 4.3.3.

The second major concern is if students really had comparable exams due to the random nature of the question selection. However, by linking each question to a question bank, that was then further linked to an outcome/topic, the instructors made sure the questions were on similar topics. After that, the questions were compared against each other within the question bank to make sure they were the same type of questions. For example, all questions in the bank "sort theory" were fill-in-the-blank, just the type of sort changed or the inputs were changed. Students would end up with a single question from the bank, but they would end up with different sorts they had to use. For the most part the variation in the questions were focused on varying inputs to the code, but the code remaining the same. This helped keep variation to a minimum, but have enough variation that students couldn't share what was on the exam.

The last threat is linked to the time of day students were asked to take the exam. As the exam was proctored remotely, students were given six days for them to pick a time to take the exam, and could opt for their own time anytime within those six days.

4.3.3 CS 2 Performance Comparison

The second assessment of performance is similar to the CS 1 evaluation performed by Wilcox and Lionelle [3]. To evaluate the difference in teaching methodologies, it is important to look

at the following course which is CS 165 (CS2). CS 2 was modified over the years to include a "review exam" during the third week of classes. Based on the results of the exam, the instructor would recommend that students retake CS 1, take a booster course along with CS 2, or no recommendation either way if they scored 60 or above. This exam is the final exam in CS 1 with small variations, giving a seven week gap between when students in CS 1 took the final (finals week of Fall semester), and when they take final again after some review (week 3 of the Spring semester).

The first review exam provides the ability to ask two major questions. Do Green or Gold continue to perform better after CS 1, to see if the same results remained over the winter break? Secondly, how do students perform based on CS 1 performance. For example, do students earn relatively the same score, or is there drop in performance due to loss of material over the break?

More importantly than performance on a single exam, how do students perform overall in CS 2? Detailed in Section 3.1 students who took CS 1, no matter their prior programming experience, performed equally when they took CS 2. For this study, the dimension of teaching methodology is added to these results. Do different teaching methodologies of CS 1, cause student performance to differ in CS 2? A full analysis will look at the different assignment grouping as done in the previous study, but also attempt to see student growth across the groupings. This will help determine if one group starts off stronger, but equalizes out over time, or if the group that starts stronger remains stronger throughout the course. Overall, there are three major groupings to look at, CS 163 - Green Students, CS 163-Gold Students, and CS 164 Students. Furthermore, when exploring threats to validity, Students in the Booster Course group was added as a fourth group.

For teaching CS 2 in the Spring, all students signed up for the same section taught by an instructor who does not have experience teaching the spiral method, nor was it the instructor who taught CS 1 in the Fall. This allowed the students to "start over" with a fresh face who did not have a vested interest in the CS 1 methodology utilized.

Minimizing Threats to Validity

When it comes to looking at the follow up course, a concern for validity of the data is the amount of scaffolding built into the course to help students fill in missing knowledge. One concern is the booster class students are encouraged to take based on their performance on the review exam. This course has shown promising results, which means a nonperforming student may easily shift into the performing category. Furthermore, the booster class teaches students how to study, using the same five principles that are the foundation of the spiral design. While this threat cannot be minimized, and attempts to remove the booster class would cause ethical concerns, it can be documented. Students who take the booster class will be grouped together after the first exam, so we can see how they perform relative to their original cohort. Furthermore, the review exam is before the initial booster class, so the results on the review exam can provide a foundation before additional support is put into place.

Overall, the influence of the instructor finding support for students struggling is a variable that should not be controlled but only observed, and that should have a minimizing affect on the student population differences drawing the Gold and Green background students closer together by the end of CS 2. The question will be, is that enough based on any initial differences that show up going into the course.

4.3.4 Retention and Student Attitude

If performance improves, but retention of students between courses drops, then the changes made are more about filtering (a.k.a. weeding out students) than about improving performance. As such, it is important to look at the number of students between CS 1 and CS 2, and see how many chose to continue on between each teaching methodology. The retention should be broken up by major grouping, to see how many students actually wanted to continue, but chose not to, and those who needed only the one course. A success will be if at the bare minimum the same percentage of students went on between the methodologies with an increase in performance.

Furthermore, previous research shows that students who have higher confidence and self-efficacy tend to be easier to retain in programs, along with higher performance. Does the Spiral design help improve confidence compared to the traditional design? To help answer that question, the students were presented with a pre and post survey. The questions for the survey can be found in Section A.3. Students in both sections (and CS 164) were asked the exact same questions. Additionally, students were asked the basic MSLQs as a baseline to see their opinion of their self-regulation and efficacy at the end of the semester. The MSLQ survey is also conducted in CS 2, so while it is expected there is not much difference in CS 2, their attitude about their own learning can be tracked throughout the entire year.

Minimizing Threats to Validity

The biggest threat to validity is student opinion is often not how students are actually performing. To minimize this threat, it would be worth looking at how students performed based on their answers. Does one group of students have a better understanding of their capabilities than the other group? With that said, the primary focus of the survey is to make sure students have a positive outlook towards their learning, as previous research shows that outlook helps with retention. This question is then tempered by looking at the retention numbers between the courses. While instructors often have an influence on retention, in the CS 163 case the instructor was the same, so this should not be a threat to validity.

The biggest threat to understanding student retention is COVID-19. Simply, this is an unknown influence as there isn't a good baseline to compare against. To minimize this threat, the retention numbers should be only compared to each other for the same system, as compared to previous semesters. This keeps the comparisons the same, with both sections having been influenced by COVID-19.

4.3.5 Conclusions

Using measures for both performance and retention, it should be possible to determine if the course design suggested with the Spiral Design is a better design for students. Overall,

Table 4.4: The variables of the study, and if they were fixed or different between experimental groups.

Design Variable	Difference
Instructor	Fixed
Lecture Modality	Fixed (online)
Help-Sessions	Fixed
Interactive Text Book Usage	Fixed
IDE Usage	Fixed
Final Exam CS 1	Fixed
CS 2 Section and Content	Fixed
Surveys	Fixed
Topic Ordering	Modified
Reading Assignments	Modified based on topics
Labs	Modified based on topics
Practice Exams	In both, Topics modified
Exams 1-3	In both, Same timing, topics modified
Practical Projects w/ Reflections	Only in Gold
Knowledge Checks	Only in Gold

Table 4.4 shows both the fixed and changed variables between the designs, so it is possible to examine the actual influence of changes. The largest changes were the order of topics and the use of the knowledge checks throughout the semester. The study itself by necessity is a two semester study, starting in the Fall and ending with the Spring session. While the original design was done before COVID-19, the unexpected pandemic caused modifications that were also taken into account by making sure both courses were modified in similar manners.

To summarize the methodology, two sections were setup for CS 163, a Green section that follows the traditional course design, and a Gold section that follows the Spiral Model. Both sections were taught by the same teacher, and the only modifications made are ones unique to the design. Students will be evaluated based on performance at the final exam for CS 1. Furthermore, students will be evaluated throughout their CS 2 experience on performance. While performance is the primary question, surveys have been developed to help determine student growth and attitudes towards the different types of methodologies used. It is believed that while Computer Science is hard, teaching students to learn and building the course off learning techniques, students will perform better, not only in the current course but in the following course.

Chapter 5

CS 1 - Results

In order to test the different teaching methodologies students were grouped into three different sections. Green students were taught by using the traditional pedagogy. Gold students were students with no-prior programming experience taught using the Spiral Design. CS164-Gold are students with prior programming experience taught using the Spiral Design. The different ordering of topics makes it difficult to evaluate student performance in CS 1 throughout the semester. Thus, in order to evaluate CS 1 performance, we focus on the cumulative final exam, and the number of students who chose to continue to CS 2. Success is defined by students taught with the Spiral methodology (Gold) out performing students taught with the Traditional methodology (Green). Furthermore, Gold needs to have at least the same, if not improved retention compared to Green.

Section 5.1 gives a quick overview of the demographics of each group to lay a baseline of who was in each group. Section 5.2 details the results on the final exam, showing that Gold students outperformed the Green students. Section 5.3 goes through the course outcomes, and how students performed in each outcome area. The mapping was accomplished by built in tools to the LMS system (Canvas) used. Section 5.4 examines the survey presented in Appendix A.3 which is a reflection of student beliefs in CS, and their belief in themselves as they progress through the course. Section 5.5 looks at the affects the Spiral model has on retention by looking at students who could continue onto CS 2.

5.1 CS 1 - Demographics

The Gold, Green, and CS164-Gold groups were compared to see if there was a significant difference between the student population, and if there was a difference if that difference leaned towards Green, biasing Green with the better students.

Table 5.1: Breakdown by number of credits when students started CS 163/4. Every 30 credits, students are the next grade level representing amount of experience with college courses.

	CS164-Gold (N=106)	**Gold (N=86)**	**Green (N=91)**
Freshman	42%	48%	38%
Sophomore	36%	32%	29%
Junior	9%	8%	16%
Senior	8%	5%	13%
Senior - Post	0%	0%	1%
Second BA	6%	6%	2%

Table 5.1 shows the percentage distribution of students across different student levels. Every level at Colorado State University is split by 30 credits, so freshmen have 30 credits or less, and sophomores have 31-60 credits, and so on. Senior-Post are students who have a bachelors degree but are not admitted to any program. Second BA students are students with a previous bachelors degree and seeking a second bachelor's. While it is difficult to tell if a student is better, it can be assumed that more credits completed often means more college experience, and more time to have developed mature study skills. 32% of students in the Green section had at least 60 credits already completed on their transcript, as compared to 19% of the Gold students. While students were not intentionally biased towards more experienced scholars, the Green section had more students who had more college experience by nearly double. With that said, running a simple chi-squares test the p-value is .23. The result is not significant at $p < .10$, meaning there is not really a significant difference in the distribution of students across the courses.

Table 5.2: Average GPA of on-campus sections of Green, Gold, and CS164-Gold

	Average GPA	**Std Dev**
CS164-Gold	3.19	1.04
Gold	3.07	1.01
Green	3.13	0.93

Looking at the average GPA of students in Table 5.2, the Green group has a slightly higher GPA. Students for whom it was their first semester at CSU were removed due to not having access to admissions records and thus previous GPA scores. Even with that said, any student

with a college GPA was calculated, showing the Green section to have a higher GPA than Gold. Running a T-Test on the Green and Gold students' GPAs across the test, the p-value is .27, and the result is not significant at $p < .10$. While there was a marginal difference in GPAs, the overall set was not noticeably different.

Looking at both GPA and student skill level helps answer the question, "what if one group of students is just a better group of students". While it is impossible to say either way, the fact students had more college experience in the Green section, with a marginally higher GPA, means if there was any bias on the student groups it was towards the Green group.

Table 5.3: Number of Men and Women in the course.

	Men	**Women**	**Total**
CS164-Gold	82	24	106
Gold	66	20	86
Green	58	33	91

Table 5.3 shows the number of students based on gender in each group. The Green group had more women than any other group, both percentage and total. This often leads to higher retention of women, biasing retention of women to the Green group. Overall, the demographics of the sections show minimal differences, with the Green section leaning towards older students and more of them are women.

5.2 CS 1 - Final Exam Results

Table 5.4 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. Using the median grade, 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.

Table 5.4: CS 1 Final Exam Grades Between Sections. All Gold groups did about 9% better than their Green counterparts.

Teaching Methodology	Average	Median	N
Green (Traditional)	53.5%	56.4%	91
Gold (Spiral)	59.5%	65.5%	86
CS164-Gold	60.1%	65.7%	106
Gold Combined	59.8%	65.7%	192

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 $p < .10$. While the results were significant, the Cohen's d showed a smaller influence due to the variability in mean grades and standard deviation. Both Gold groups showed a Cohen's $d = 0.27$ relative to Green. To each other, Gold and CS164-Gold only shows a Cohen's $d = 0.02$, which strengthens the results that while 0.27 is small, it is a noticeable difference from the Gold groups.

Gold and CS164-Gold only differing by .2% is a change from the 2018 paper by Wilcox and Lionelle, which showed that by the end of CS 1 students with prior programming experience (CS 164) outperformed students without prior programming experience by 6% [3]. 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 experience 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%.

5.3 CS 1 - Final Outcome Results

In Section 4.2.4, eleven learning outcomes were defined for both GREEN and GOLD exams using the Canvas outcomes feature. Questions for the Final Exam were linked to one of the 11 outcomes, and based on how a student scored on a problem, they were awarded both points for the exam and points towards showing mastery in an outcome. While the outcomes are originally designed for standards based grading, and demonstrating mastery of topics through standards, they allowed us to link questions to learning outcomes for the course providing an idea of how well students understood each topic.

Outcome points ranged from 0-5, with 3 being the recommended value for meeting mastery of that outcome target. Furthermore, every question was given a difficultly on how hard it would be to show a 3 in that category. For example, easier questions required students earned 100% on the question to earn a 3. More difficult questions may only ask students to earn 80% to show mastery, and any points on that question above the 80% helps a student to exceed mastery by scoring higher than a 3. We used the canvas default setting of "65/35 Decaying Average" for calculating outcome scores. The decaying average looks across all mastery scores earned on a particular topic. The most recent score receives 65% of the overall score, and the average of the remainder count as 35%. For example, if a student scored 4 out of 5 parts of a question correctly, they would have earned a 3 on their mastery score for that outcome since they scored at least 80% on the problem. However, they then would have previous questions on the same outcome taken into account, and assuming they had scored 3, 2, 1, 3 respectively on previous questions, their total mastery score would be:

$$(.65 * 3) + (.35 * ((3 + 2 + 1 + 3)/4) = 4.2$$

At the same time, if they had done worse on the question, for example only earning 1 mastery point, they would end up with not yet meeting expectations with a score of 2.9. Scores are not rounded. While these mastery scores don't map directly to student performance, as the grading scheme doesn't match up exactly with standards based grading, they have the ability to

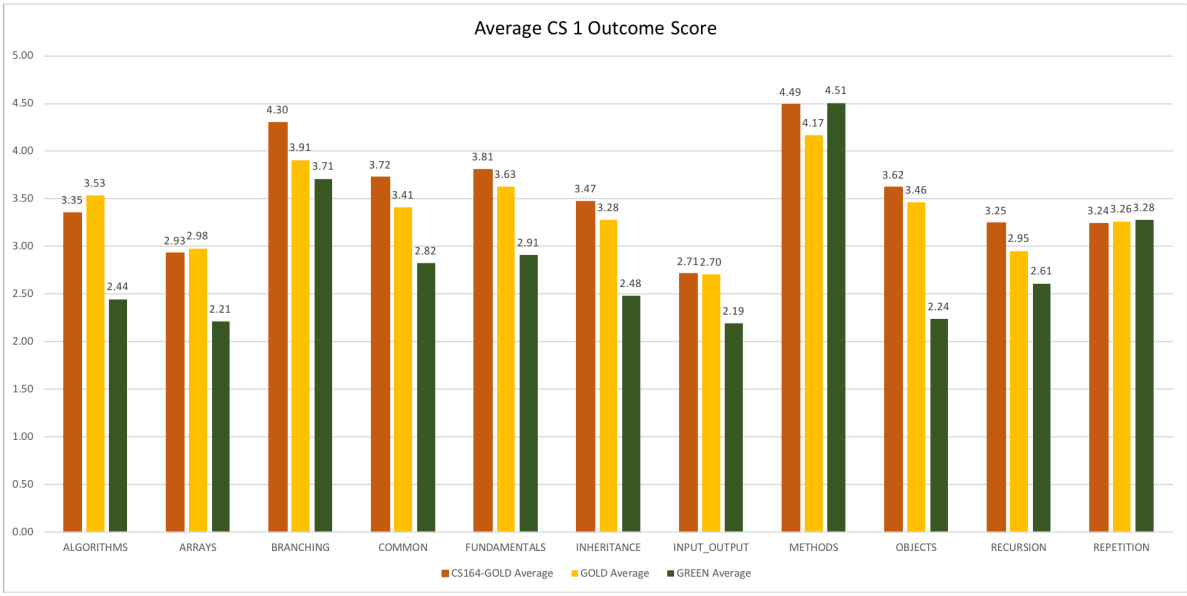


Figure 5.1: Outcome Averages across the outcome categories for each section. In all cases but Methods and Repetition, Gold outperforms Green.

give us a clue as to students' understanding of certain topics on the exams, especially when the questions are coming from rotating question banks.

Figure 5.1 shows the average outcome score for each outcome across the three different student classifications. In nearly every outcome, Gold outperforms Green except for Methods and Repetition. The high score in methods for all courses is a reflection of the emphasis of early methods in the department, and the teaching of "Divide-Glue-Conquer" as a means to approach problems. However, for the Green section, the standard deviation was relatively high, meaning some students really understood methods bringing up the average score while others struggled. Table 5.5 shows each outcome average with standard deviation. CS164-Gold showed the highest consistency of scores with the lowest standard deviation, while Green showed the highest inconsistency of scores with the highest standard deviation.

The largest difference in scores shows up in the students' understanding of objects. CS164-Gold and Gold show a much higher understanding of objects than their Green counterparts. They also show a higher grasp of inheritance, which is notable as the Gold sections spent less time on inheritance than the Green section due to when the topics was introduced. In the Green

Table 5.5: Outcomes Scores based on final exam results for each section. Greater that 3 means students on average met expectations. Green only met expectations in Branching, Methods, and Repetition. Gold and CS164-Gold only missed meeting expectations in Arrays and Input_Output, often areas with overlapping questions.

	CS164-GOLD		GOLD		GREEN	
	Average	Std Div	Average	Std Div	Average	Std Div
ALGORITHMS	3.35	1.61	3.53	1.49	2.44	2.08
ARRAYS	2.93	1.49	2.98	1.48	2.21	1.61
BRANCHING	4.30	0.73	3.91	0.96	3.71	1.04
COMMON	3.72	0.93	3.41	1.08	2.82	1.31
FUNDAMENTALS	3.81	0.76	3.63	0.83	2.91	0.92
INHERITANCE	3.47	1.38	3.28	1.49	2.48	1.41
INPUT_OUTPUT	2.71	1.08	2.70	1.04	2.19	1.38
METHODS	4.49	0.70	4.17	0.96	4.51	0.94
OBJECTS	3.62	0.95	3.46	1.05	2.24	0.98
RECURSION	3.25	1.51	2.95	1.47	2.61	1.82
REPETITION	3.24	1.54	3.26	1.58	3.28	2.28

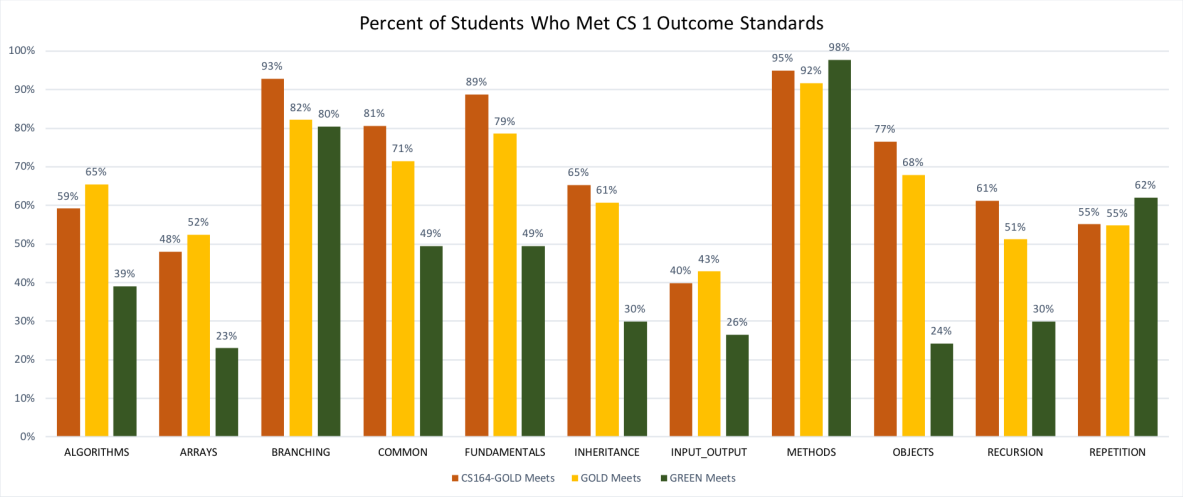


Figure 5.2: Percent of Students who meet the outcome standards

section, week 10 introduced inheritance with two entire weeks focused on the topic. The Spiral Model introduced inheritance in weeks 11 and 13. Although it also had two weeks on the topic, the Spiral Model had a gap and exam between. Both sections continued to use inheritance after introduction, giving Green an additional week on the topic in addition to the focused time.

Since we can look at outcomes as meets or does not meet, we also looked at the number of students who met those outcomes. Figure 5.2 shows each group based on the percent of stu-

dents who meet the outcome standards. The difference between Gold and Green becomes even more profound, as 44% more Gold students demonstrated mastery of objects over Green students, a rather important concept for Object Oriented Programming. Furthermore, 31% more Gold students showed mastery of Inheritance over Green students, along with 30% more in Fundamentals, 29% more in Arrays, 26% more in Recursion, and 26% more in Algorithms, which was one of the few topics only introduced once in the Spiral design. The only areas in which more Green students meet a standard as compared to Gold are Methods and Repetition, which matches the results from the average score.

In an ideal world, all students would master 100% of the topics, but this is rarely the case. Furthermore, as the course wasn't built around standards based grading, students were not required to master every topic for their grade. Instead, they could do well in most all topics, and that would help them with a good overall grade. At the same time, seeing how students master each topic helps determine the overall strength of the students in the section before they move onto CS 2.

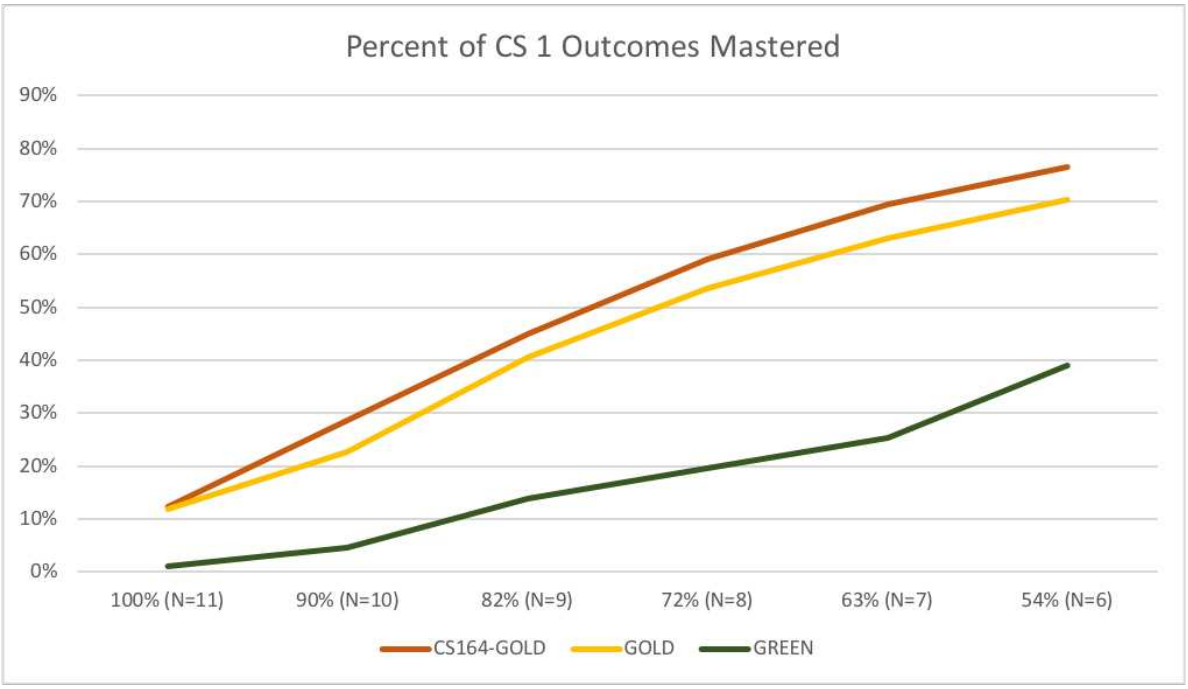


Figure 5.3: Percent of students who mastered N number of outcomes

Figure 5.3 shows the number of students who mastered all the 11 outcomes, students who mastered 10 of 11 (90%) of the outcomes, students who mastered 9 of 11 (82%) of the outcomes and so on until 54% (n=6). Only 1% of Green students mastered all eleven outcomes, where 12% of Gold students mastered all outcomes. 23% of Gold students demonstrated a 90% mastery of the outcomes as compared to only 5% of Green students. Moving out to only needing to master 54% of the topics, 70% of the Gold students at least mastered half the topics while only 39% of the Green students mastered half the topics. Table 5.6 details the percent of students who mastered topics from N=11 to N=7. Furthermore, it also provides the unique students for each outcome, as mastering 11 topics also means mastering 10, 9, 8, 7, and 6 topics. The unique count removes that factor, showing how many mastered only that exact number of topics instead of the ever increasing number shown in the chart. Looking at the unique students in each topic area, both sections leaned towards 9 or greater topics mastered, or at least 82% of topics mastered. While most students in Green demonstrated mastery of less than half the topics.

Table 5.6: Percent of students who mastered at least N number of outcomes out of 11 possible outcomes. The Unique percent is the number of students who only mastered that many outcomes and no additional outcomes.

	N=11	N=10, (Unique)	N=9, (Unique)	N=8, (Unique)	N=7, (Unique)	N=6, (Unique)
CS164-GOLD	12%	29%, (16%)	45%, (16%)	59%, (14%)	69%, (10%)	77%, (7%)
GOLD	12%	23%, (11%)	40%, (18%)	54%, (13%)	63%, (10%)	70%, (7%)
GREEN	1%	5%, (3%)	14%, (9%)	20%, (6%)	25%, (6%)	39%, (14%)

While there may be future work to better align outcomes to truly document student performance, when looking at Gold relative to Green, the students taught with the Spiral Method (Gold, CS164-Gold) continuously outperform Green section students. A possible future study could be to explore the relationship to outcomes in depth, and implement fully standards based grading where students earn "A" letter grades only if they demonstrate mastery of at least 90% of the topics. At the same time, students would need the opportunity to return to topics to prove mastery before the end of the semester. It is also worth noting the Final Exam was the primary factor in determining "outcome mastery" as compared to a wide range of assignments. The fi-

nal exam was already documented as being a relatively hard exam with the median score of the Green section being 56.4% making the exam score a reflection of the topics mastered.

5.4 CS 1 -Student Self-Evaluation

At the start of the semester, students were given a survey gauging their opinions about their interest and ability to learn computer science. Additionally, the same survey was given at the end of the semester with the same questions, along with the MSLQs as a means to gain their opinions about their study skills. While self reflection is notoriously inaccurate, such gauges can help enlighten us as to how students view themselves as it relates to computer science, both before and after the course. This section analyzes the survey paying particular attention to women as a dominate underrepresented population. As the survey was opt-in for both courses, the number of students who took the survey (242) was 17% less (-41) than those who took the course (283).

The first four questions were duplicated for all students at the beginning of the semester and at the end. These same questions have often been used in Colorado State University CS 0 and CS 1 courses, though this evaluation is only for this semester as past semesters were not designed to have the data used outside of the department. The four primary questions for the survey were based on a rating system from 1-10 with 10 being the highest. They were prompted with the question, "Based on a scale where 1 is the lowest and 10 is the highest, how would you rate the following statement?", and then a statement they were asked to rate. Two questions were focused on students' interest in the field, and two were focused on students' confidence in themselves to be successful in the field. The statements are as follows.

- I know that I like Computer Science.

  This question will be presented by LIKE throughout the results.

- I think Computer Science is interesting.

  This question will be presented by INTERESTING throughout the results.

- I feel confident in my ability to learn Computer Science.

  This question will be presented by LEARN throughout the results.

- I feel confident in my problem-solving ability.

  This question will be presented by PROBLEM SOLVING throughout the results.

Figure 5.4, Figure 5.5, Figure 5.6, and Figure 5.7 illustrate the average scores of what the student self-evaluated for each question. The curve for the student samples was a normal bell-curve distribution. Groups that would start out higher often remained higher throughout the semester, no matter the teaching methodology. CS164-Green students rated themselves higher in every category, and continued to rate themselves higher even if there was a drop in their rating. Gold students were more interested in Computer Science from the start, and remained more interested at the end. Green students were more confident about their skills at the start, and remained more confident in the end. The latter could be attributed to the slightly higher maturity of the Green students, as Green had 33% of their students who started with 60 credits or more before taking CS 1, and Gold only had 19%, meaning Gold had more students for whom this was their first college course.

Comparing each group against themselves, CS164-Gold had a very small change throughout the semester, where as Gold showed a major increase in their interest in Computer Science, but also a noticeable drop in their confidence to be successful at Computer Science. For example, CS164-Gold increased their like of Computer Science by .25 points, but also had a decrease in their confidence to learn by .06 points. At the same time, Gold showed an increase by .88 in their interest, but a decrease in their confidence to learn by .62 points. Green showed similar differences as Gold but with less dramatic extremes. Their largest increase was also like of Computer Science with a .55 increase, but with their confidence in problem solving being decreased the most by .29. While these changes look extreme, it is worth noting that students could only select whole numbers between 1 and 10, as such an increase of .88 is less than an entire point of increase.

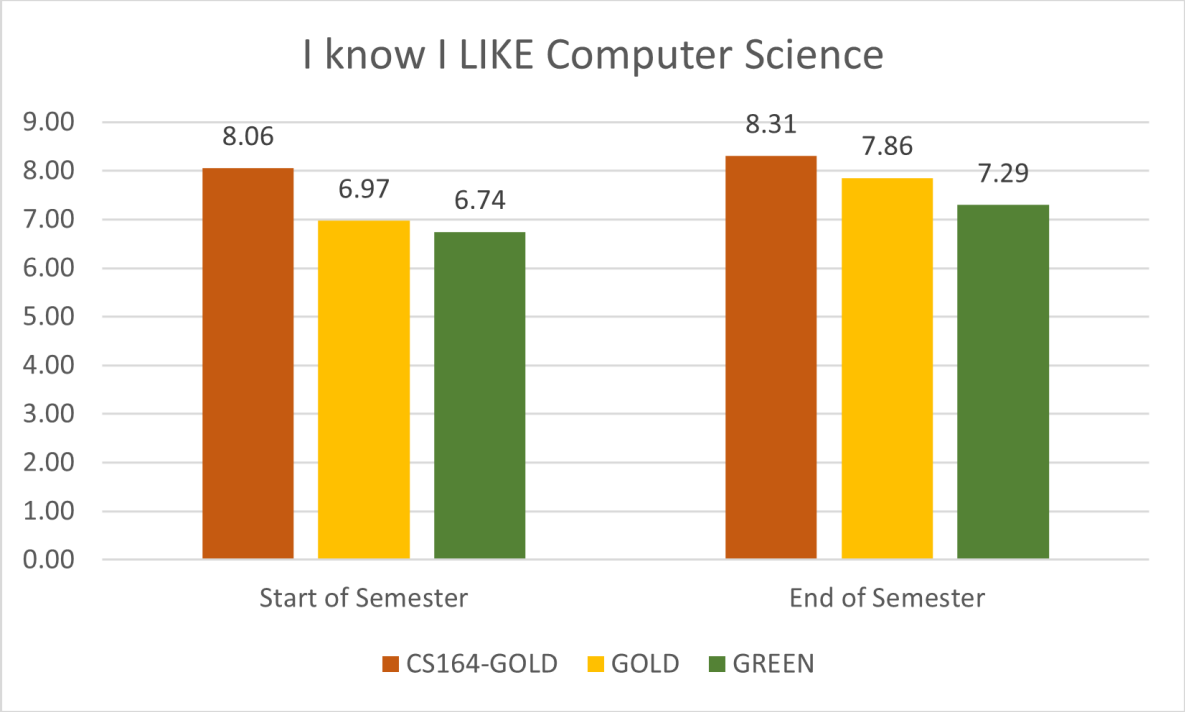


Figure 5.4: The average score across students both at the start of the semester and end of semester on how much students felt they liked Computer Science.

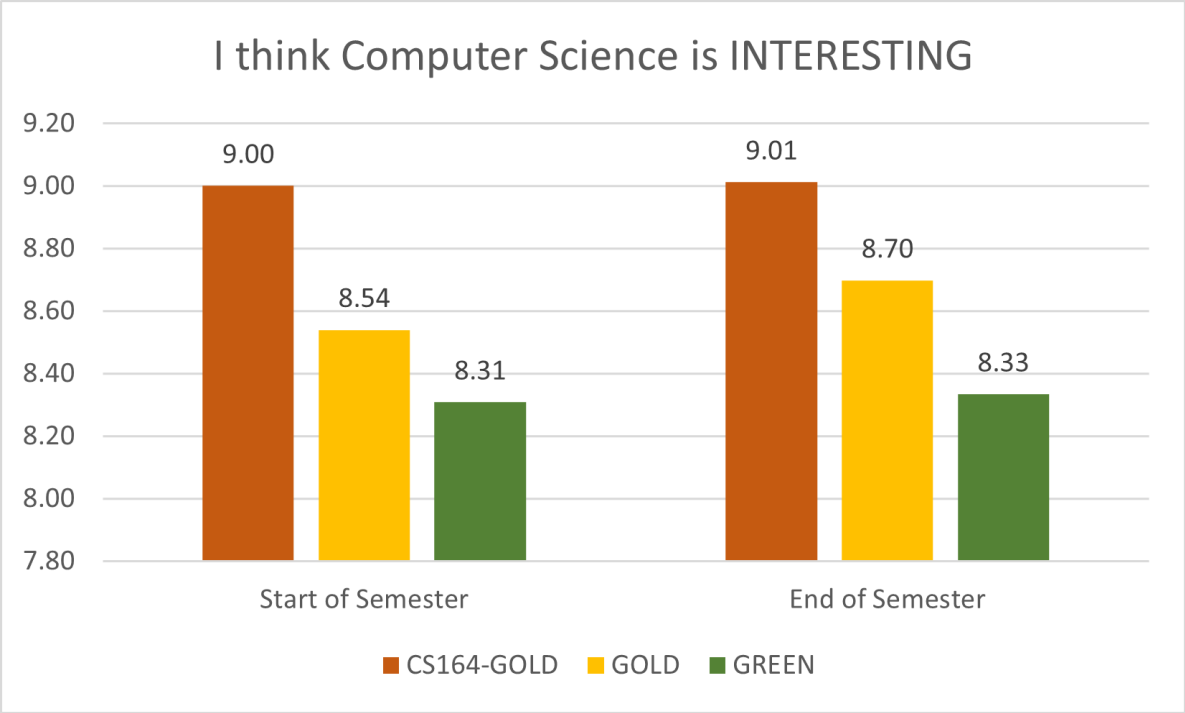


Figure 5.5: The average score across students both at the start of the semester and end of semester on how students felt how interesting Computer Science was as a topic.

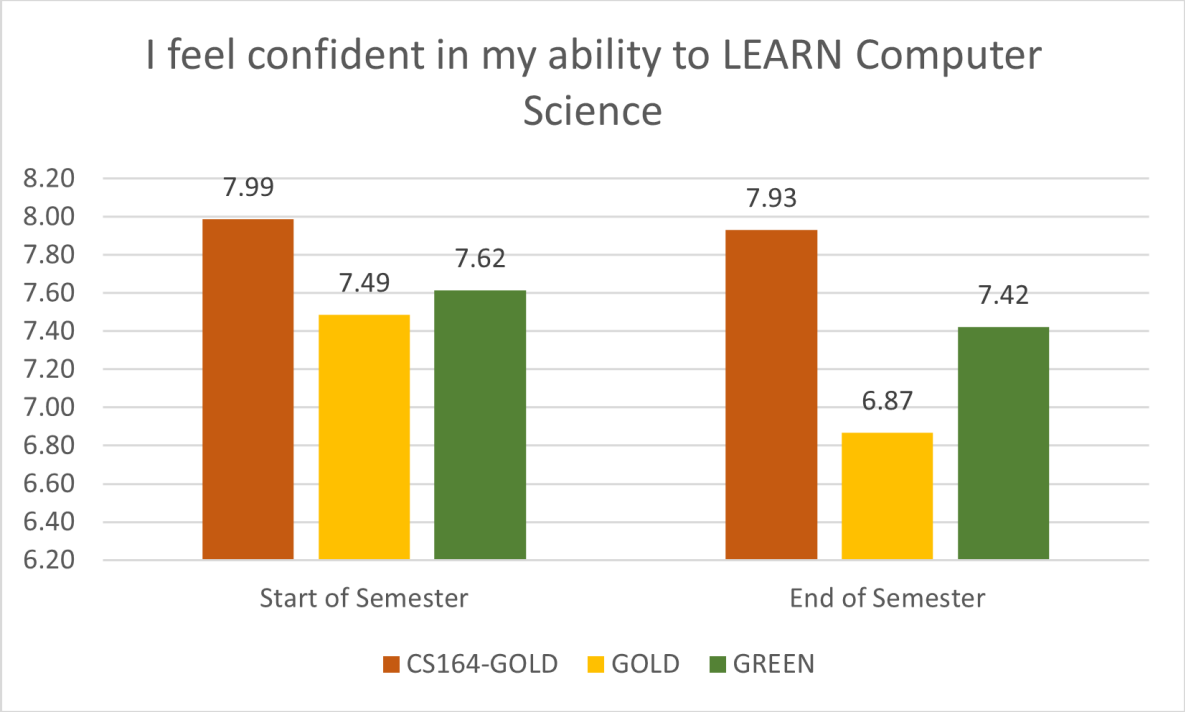


Figure 5.6: The average scores on what students thought about their ability to learn Computer Science.

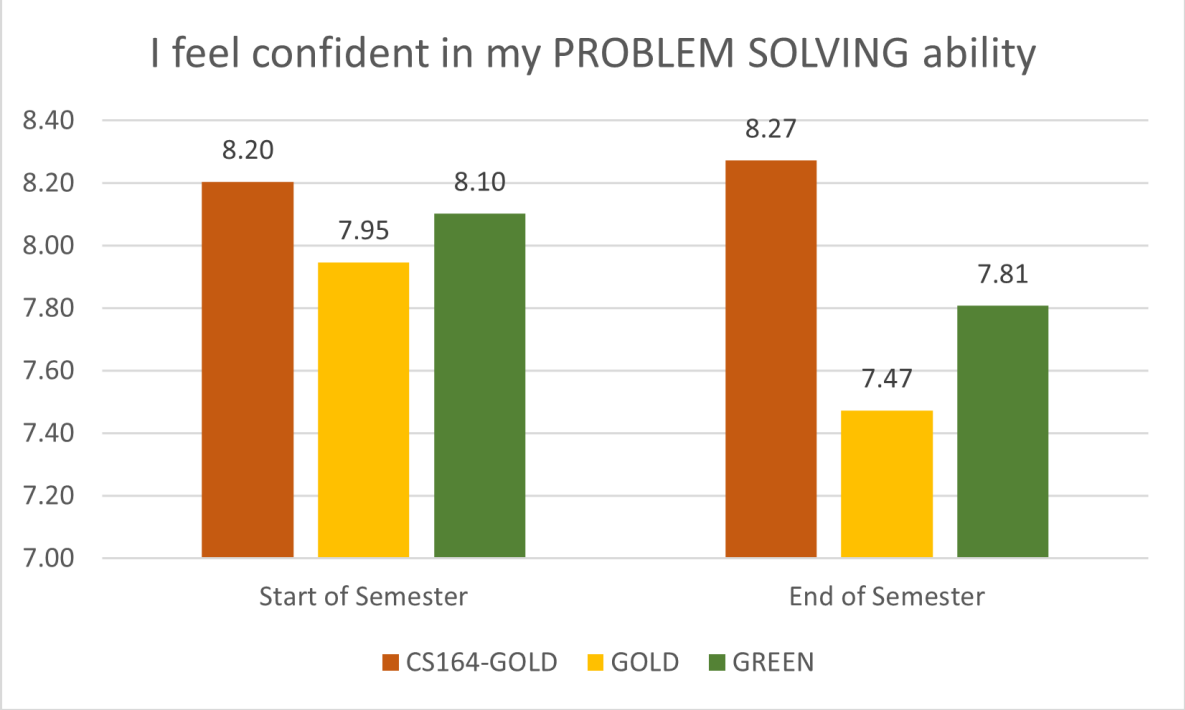


Figure 5.7: The average scores on how students rated their problem solving ability in general.

Table 5.7: The average student rating across interest and confidence questions, both start of semester and end of semester.

	CS164-GOLD			GOLD			GREEN		
	F	M	All	F	M	All	F	M	All
LIKE Semester Start	8.05	8.06	8.06	5.94	7.27	6.97	6.10	7.17	6.74
LIKE Semester End	8.27	8.32	8.31	8.06	7.80	7.86	6.94	7.53	7.29
INTERESTING Start	9.14	8.95	9.00	7.94	8.71	8.54	7.77	8.66	8.31
INTERESTING End	9.09	8.98	9.01	8.59	8.73	8.70	8.23	8.40	8.33
LEARN Start	7.91	8.02	7.99	6.29	7.83	7.49	6.94	8.06	7.62
LEARN End	7.68	8.02	7.93	6.76	6.90	6.87	7.10	7.64	7.42
PROBLEM SOLVING Start	8.14	8.23	8.20	6.82	8.27	7.95	7.55	8.47	8.10
PROBLEM SOLVING End	8.23	8.29	8.27	7.00	7.61	7.47	7.45	8.04	7.81

Table 5.7 shows the average for every question start and end of the semester, along with the averages broken up by gender. Looking at the gender differences, women reacted very differently to the spiral model than men did. Where men often had a large decrease in their confidence, women had an increase in their confidence, even more so than the Green methodology. This is particularly due to males being much more confident than women at the start of the semester, but by the end of the semester they are reporting similar confidence levels, creating a noticeable decrease for males, but an increase for women.

Figure 5.8 shows the difference between start and end of the semester for each of the four questions, across the teaching methodologies. Women in the Gold group showed the highest increase across all the categories and was the only group that did not have a decrease. With that said, the only question that showed a significant difference using a T-Test was LIKE, and the amount change was significant for all of them at $p < 0.10$ with CS164-Gold showing .06, Gold at .01, and Green at .02. While changes are documented or the other questions a significant difference was not discovered. Table 5.8 shows the differences along with the p-values found between start and end of the semester scores. As such, even if a difference was documented, there may not have been a significance documented between the start and end of the semester student responses.

Table 5.8: Difference between start and end of semester for each survey question. P-value calculated by looking at student responses to questions between start and end of the semester.

	CS164-GOLD		GOLD		GREEN	
Women	Avg	p-value	Avg	p-value	Avg	p-value
Difference Like	0.23	**0.06**	2.12	**0.01**	0.84	**0.02**
Difference Interesting	-0.05	0.31	0.65	0.16	0.45	0.45
Difference Learn	-0.23	0.40	0.47	0.26	0.16	0.41
Difference Problem Solving	0.09	0.18	0.18	0.40	-0.10	0.42
Men						
Difference Like	0.26	0.16	0.53	0.81	0.36	0.20
Difference Interesting	0.03	0.46	0.02	0.47	-0.26	0.24
Difference Learn	0.00	0.5	-0.93	**0.01**	-0.43	0.11
Difference Problem Solving	0.06	0.26	-0.66	**0.02**	-0.43	0.24

Figure 5.9 shows the differences between responses for men from the start to end of the semester. Men in the Gold group had the largest increase of interest in Computer Science, but also the largest decrease in confidence. At the same time, they are still more confident than the women in the Gold group. Men just simply started much higher in their confidence so while

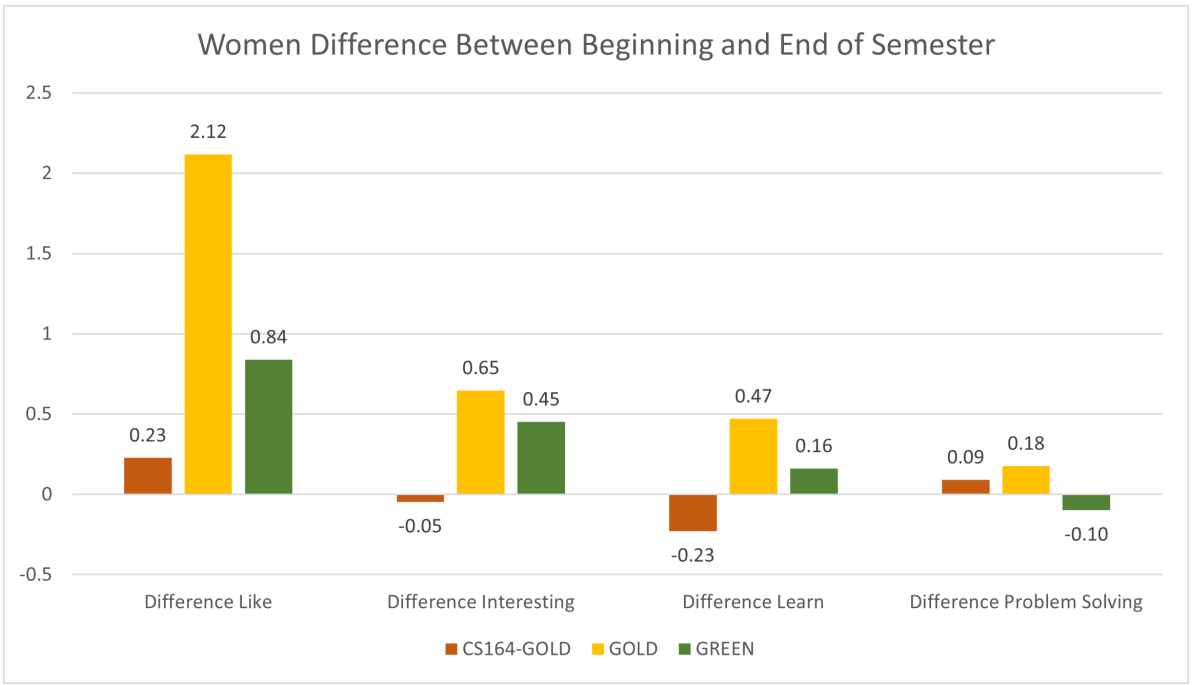


Figure 5.8: The difference between start and end of the semester for women for each survey question by teaching methodology. Gold showed the greatest increase across all categories for women.

the Gold women increased, the Gold men decreased but still remained higher in their average responses.

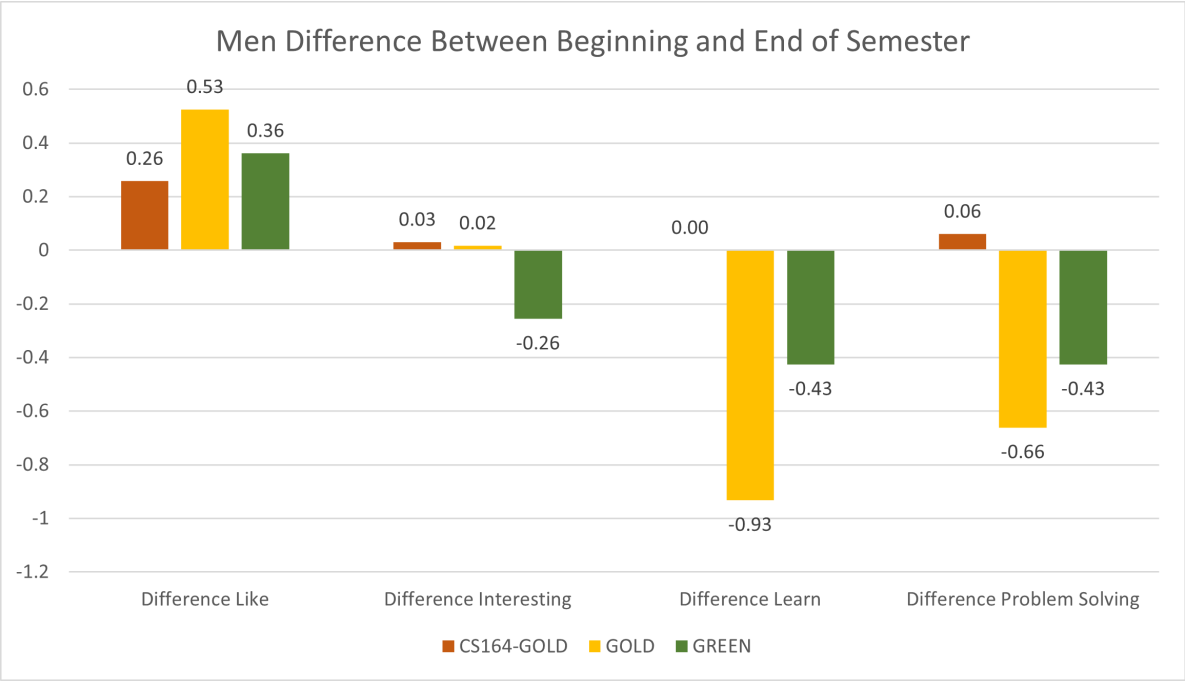


Figure 5.9: The difference between start and end of the semester for men for each survey question by teaching methodology. Gold showed the greatest increase in their like of CS, but also the greatest decrease in their confidence.

Looking at both men and women, the Gold group showed both the most improvement, especially for women, and also greatest decrease in male confidence. This decrease actually brought male and female opinion closer to each other, though the concern remains that a decrease in confidence may also inadvertently push males away from continuing to take future computer science courses.

5.4.1 Motivated Strategies for Learning Questionnaire (MSLQ)

At the end of the semester, students were asked to fill out the Motivated Strategies for Learning Questionnaire (MSLQ) developed by Pintrich et al [71]. The following contains an analysis similar to the one presented in the manual provided by Pintrich [77], with one notable change, while the manual would correlate to the final class score, we focused our correlation on the final

exam score as that was a shared score between all three sections. The MSLQ breaks up the questions students answered into five categories. The first three categories: Self-Efficacy, Intrinsic Value, and Test Anxiety; relate to students' Motivational Beliefs. The last two categories: Cognitive Strategy Use and Self-Regulation relate to a student's Self-Regulated Learning Strategies. Each question is broken up in the tables presented and can be mapped to the question wording in Appendix A.3.

Table 5.9: Documents the average score for questions related to determining student self-efficacy. CS164-Gold was highest, Green second, and Gold lowest.

Self-Efficacy	CS164-GOLD		GOLD		GREEN		Correlation to Final Exam
	Avg	Std Div	Avg	Std Div	Avg	Std Div	
Q2	5.08	1.29	4.53	1.65	4.82	1.41	0.26
Q7	5.87	1.02	5.21	1.40	5.49	1.37	0.36
Q10	5.18	1.47	4.89	1.62	4.92	1.52	0.34
Q11	4.69	1.38	4.64	1.54	5.05	1.37	0.14
Q13	5.31	1.49	5.03	1.45	5.10	1.35	0.34
Q15	5.12	1.72	4.95	1.57	5.04	1.41	0.42
Q20	3.54	1.48	3.91	1.52	3.95	1.29	0.00
Q22	4.36	1.48	3.87	1.70	4.44	1.42	0.14
Q23	5.88	1.19	5.49	1.42	5.36	1.32	0.30
Scale	**5.00**	**1.39**	**4.72**	**1.54**	**4.91**	**1.38**	**0.26**

Table 5.9 shows the average scores plus standard deviations for Self-Efficacy questions. The overall category has the highest positive correlation to the final exam score, but still a weak correlation at .26. Question 15, "I think I will receive a good grade in this class.", showed the highest correlation among all the questions, showing a strong positive correlation. However, given the time of when the survey was administered, most students had a good idea of their grade already before taking the survey. Q7, Q10, Q13 all showed moderate correlation with the final exam score. Question 20, "My study skills are excellent compared with others in this class.", has no-correlation with the final grade. It is worth noting that the group that scored the highest grades actually rated themselves the lowest on their study skills, and the group that scored the lowest grades rated themselves highest.

Table 5.10: Questions related to students' opinion about the Intrinsic Value of education and their learning. Both gold groups ended up with higher averages than Green, though only marginally. Does not show a strong correlation to final exam performance.

Intrinsic Value	CS164-GOLD		GOLD		GREEN		Correlation to Final Exam
	Avg	Std Div	Avg	Std Div	Avg	Std Div	
Q1	5.42	1.26	5.03	1.26	5.13	1.26	0.23
Q5	6.25	1.20	6.08	1.22	5.99	1.31	0.22
Q6	5.98	1.19	5.91	1.19	5.51	1.55	0.35
Q9	6.15	1.26	5.72	1.52	5.71	1.55	0.31
Q12	4.61	1.50	4.68	1.37	4.65	1.42	-0.02
Q17	5.86	1.13	5.82	1.17	5.92	1.19	0.12
Q18	6.33	1.06	6.12	1.22	6.00	1.24	0.31
Q21	6.07	1.21	6.08	1.20	5.74	1.39	0.30
Q25	6.40	0.96	5.92	1.44	5.99	1.29	0.28
Scale	**5.90**	**1.20**	**5.71**	**1.29**	**5.63**	**1.36**	**0.23**

Intrinsic Value measures the student's belief that their education and what they are learning has value to themselves and their future. Table 5.10 breaks down the average rating for each question related to intrinsic value. Q6, Q9, Q18, and Q21 all show a moderate positive relationship with the final exam score and for all four questions both Gold groups scored themselves higher. Question 6, "I like what I am learning in this class." showed the highest correlation, and all four of the highest correlation questions focused on students like of the course or feeling that specific topics will be useful. Question 12, "I often choose paper topics I will learn something from even if they require more work." showed a negative correlation. This could be because the course is not a paper topic based course, and may be poorly formatted for this particular subject. Removing the question brought the average for the question category from .23 to .27, making it the highest correlated category next to Self-Efficacy.

Test Anxiety is a known problem in academia, especially prevalent in STEM based disciplines that often rely on testing as the primary form to measure student outcomes. The MSLQ has four questions related to test anxiety shown in Table 5.11, due to their positive wording, a negative correlation is expected. For example, the higher someone rates their anxiety, the lower they tend to do on testing. The idea of remembering facts often relates to STEM disciplines, and Question 3, "I am so nervous during a test that I cannot remember facts I have learned." shows

Table 5.11: Shows average rating from 1-7 related to Test Anxiety students feel. Both Gold sections are lower than Green, indicating less anxiety in testing. The correlation to the final is a negative coefficient as expected, but not high enough to represent a strong correlation.

Test Anxiety	CS164-GOLD		GOLD		GREEN		Correlation to Final Exam
	Average	Std Div	Average	Std Div	Average	Std Div	
Q3	4.01	2.01	4.37	1.68	4.31	1.61	-0.26
Q14	4.57	1.85	4.82	1.73	4.71	1.68	-0.15
Q24	5.16	1.90	5.16	1.71	5.60	1.42	-0.19
Q27	4.36	1.99	4.29	1.72	4.40	1.51	-0.19
Scale	**4.52**	**1.94**	**4.66**	**1.71**	**4.75**	**1.56**	**-0.20**

the highest negative correlation with exam performance. Green had the higher self ratings out of the groups, showing they tended to feel more anxious about exams.

Table 5.12: Contains questions related to various student techniques. The specific techniques were not directly taught, and Green believed they employed more techniques than the rest of the other groups. There is no correlation with how students felt they employ study techniques and final exam outcomes.

Cognitive Strategy Use	CS164-GOLD		GOLD		GREEN		Correlation to Final Exam
	Avg	Std Div	Avg	Std Div	Avg	Std Div	
Q30	5.64	1.15	5.59	1.07	5.69	1.14	0.06
Q31	5.38	1.33	5.36	1.26	5.38	1.26	0.05
Q33	4.70	1.89	4.22	1.54	4.51	1.59	0.20
Q35	5.39	1.41	5.27	1.31	5.12	1.40	0.11
Q36	6.01	1.08	5.78	1.11	5.85	1.08	0.12
Q38	5.29	1.55	5.68	1.18	5.71	1.14	-0.11
Q39	3.63	1.93	4.24	1.75	4.12	1.85	-0.02
Q42	4.35	1.86	4.55	1.68	4.85	1.65	-0.07
Q44	5.99	1.20	5.79	1.17	6.08	1.21	0.25
Q47	5.69	1.28	5.66	0.93	5.82	1.09	0.10
Q53	3.70	1.88	4.25	1.79	4.12	1.67	-0.14
Q54	3.51	1.82	3.67	2.04	3.49	1.79	-0.08
Q56	5.79	1.04	5.68	1.19	5.90	1.06	0.18
Scale	**5.01**	**1.49**	**5.06**	**1.39**	**5.12**	**1.38**	**0.05**

Table 5.12 shows average student opinions about the cognitive strategies they use for learning. The strategies suggested in the question set were not necessarily strategies suggested by the course, nor did they line up with the course design. However, most strategies recommended are

commonly employed as elaboration strategies. Question 44, "I use what I have learned from old homework assignments and the textbook to do new assignments." had the highest correlation with the exam, though at a weak positive correlation of .20. A number of strategies had a negative correlation with the exam grades, with Question 53, "When I read material for this class, I say the words over and over to myself to help me remember.", having the highest negative correlation with exam performance. Overall, Green ranked themselves higher than Gold in most areas, but the student self evaluation of their cognitive strategies had negligible correlation with the final exam grade at .05.

Table 5.13: Questions related to student's opinion of ability to self-regulate. Green scored themselves higher than any other group, and the way the questions are answered does not show a correlation to the final exam.

Self-Regulation	CS164-GOLD		GOLD		GREEN		Correlation to Final Exam
	Avg	Std Div	Avg	Std Div	Avg	Std Div	
Q32	4.89	1.41	4.66	1.52	5.01	1.49	0.10
Q34	5.07	1.48	4.91	1.32	5.38	1.16	0.19
Q40	4.29	1.75	4.39	1.59	4.05	1.61	0.00
Q41	5.26	1.45	5.38	1.50	5.58	1.24	0.06
Q43	4.94	1.65	5.03	1.23	5.17	1.49	0.05
Q45	4.32	1.86	3.78	1.77	3.88	1.64	0.26
Q46	4.51	1.86	4.43	1.59	4.40	1.62	0.07
Q52	4.75	1.66	4.87	1.59	4.97	1.39	0.02
Q55	5.55	1.42	5.38	1.60	5.85	1.30	0.13
Scale	**4.84**	**1.61**	**4.76**	**1.52**	**4.92**	**1.44**	**0.10**

Self-Regulation questions focus on student time management and focusing on how they either add additional opportunities to study, or that they continue to study even if they are disengaged or uninterested in the subject matter. Table 5.13 shows the questions related to self-regulation. Question 45, "I often find that I have been reading for class but don't know what it is all about." is a negative question, causing the answers to be inverted when scoring. It also showed the highest correlation of the group at .26. If students understood what they were reading, they did better. This may also be connected to the reading being an interactive

programming textbook. Overall, Green ranked themselves higher at Self Regulation, and the category of questions had negligible influence to exam outcomes.

Looking across the question categories, the Gold groups ranked themselves higher on their Motivational Beliefs compared to Green. The group groupings within Motivational Beliefs also showed higher correlation to exam results, meaning student motivation, while only moderate, may have had an affect on performance. However, looking at Self-Regulated Learning Strategies student answers varied greatly, and the questions had negligible correlation with exam results. Gold groups were more critical of their self-regulated strategies often ranking themselves lower than Green.

5.5 CS 1 - Retention and Attrition

For most CS 1 courses, there is often the understanding that students who take the course are taking their first CS course due to lack of exposure at the K-12 level. Students often quickly learn that computer science and liking computers are very different topics, so there will be some acceptable loss. At the same time, we want to minimize the number of students who choose to not continue onto the following course, especially so for students who have to overcome social barriers to their participation in computer sciences (underrepresented students, such as women). Another way to look at it, any student who makes the conscious decision to continue in computer science, should continue onto later computer science courses with the only barrier being performance. Similarly, Any changes one makes to an introduction CS course, must be evaluated against the number of students who could and chose to continue onto CS 2.

Table 5.14: 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%

Due to the way the courses were curved by the instructor, 84% of the students who took Green, Gold, or CS164-Gold all passed the course with a C or above, the minimum grade needed to take CS 2. Table 5.14 shows the percentage of students who took CS 1 and chose to continue onto CS 2 of the 84%. 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. For women, the Gold section proved to have a much higher retention rate than the Green, by nearly 19.2%. 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.

As there are many degrees that only need CS 1, but not CS 2, we only look at men and women who were enrolled in a technology major that requires CS 2 for their degree. The technology majors reviewed are as follows:

- Computer Engineering
- Computer Science
- Data Science
- Undeclared Technology Interest (seeking CS)

Gold students have a noticeably 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%. Simply, students who start out interested in technology and programming, remain interested when using the Spiral method of teaching as compared to the traditional.

Overall, the Spiral design was not designed to improve retention of students, but simply improve performance with the hope that with improved understanding more students would seek to take CS 2. These results show that not only does the Spiral design increase the number of students who chose to move onto the next course, it has a particularly strong influence on encouraging women to take CS 2.

Chapter 6

CS 2 - Results

Since the overarching goal is 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 along with CS164-Gold.

For the following analysis 12 students were removed from the reporting results. The first exam in CS 2 is meant to help determine interventions for students. Those who do poorly on the exam are invited to take a "booster" course that teaches the same techniques for studying that are the foundation of Spiral Design. While the results for the booster students are highly interesting due to the success of the booster course over three semesters, the results are outside the scope of this dissertation. It was also noted that the majority of the students were from the Green grouping, meaning leaving them in the results would have reduced Green performance, especially on the review exam.

Going into this analysis it is worth noting Wilcox and Lionelle showed that students without prior programming experience would catch up to students with prior programming by the end of CS 2 [3]. For comparison, Green and Gold students are students Without prior programming, and CS164-Gold are students with prior programming. A guideline for success would be as follows

- Gold outperforms Green on the review exam. As the review exam measures the exact same topics as CS 1, increased performance would indicate greater long term recall of topics.
- Gold continues to outperform Green throughout the course. This would indicate that while students often "catch-up" to each other by the end of CS 2, the influence of the

Spiral design remained with the students, showing one pedagogy is better at increasing student performance than the other pedagogy.

- Gold students outperforms Green across assignment categories. This would indicate that the Spiral design is not only beneficial in the taking of exams, but also programming and other assignment types presented.

Section 6.1 details the results of the review exam, and reevaluates the learning outcomes for the review exam between each of the groups. Section 6.2 details the topic exams in CS 2, comparing understanding of new topics between the groups. Section 6.3 details both the assignment types and the final grades for students in CS 2.

6.1 CS 2 - Review Exam

CS 2 students are given a review exam at the end of week 2. The exam is essentially the CS 1 cumulative 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 before they are asked to retake it. Furthermore, the first two weeks of CS 2 are 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. A slightly larger difference than was recorded at the end of CS 1 as shown in Section 5.2. 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 a p-value of .24 at $p < .05$. Both Gold and CS164-Gold had significantly higher results than Green, with average, median, differences, and significant p-values marked in bold in Table 6.1.

6.1.1 CS 2 Review Exam - Outcomes/Topic Analyses

Similar to CS 1, the topics for the exam were tracked. They followed the same method as shown in Section 5.3, including the decaying average method to calculate the score for the individual student. The averages for student outcomes are shown in Table 6.2. CS164-Gold met

Table 6.1: Median score for the review exam. Green was outperformed by both CS164-Gold and Gold significantly. Bold values are significant at p <.05.

	Average	Median	Difference (p-value)		
			CS164-GOLD	GOLD	GREEN
CS164-GOLD	79%	76.1%	-	2.9% (0.25)	**12.9% (<0.001)**
GOLD	76%	73.2%	2.9% (0.25)	-	**10% (0.03)**
GREEN	66 %	63.2%	**12.9% (<0.001)**	**10% (0.03)**	-

outcome goals (avg > 3) for every topic except Arrays, Input_Output, and Repetition. All three topics often shared questions which could be a factor of the reduced scores. Gold showed similar results, but also did not meet the recursion outcome, showing only 2.86 of the 3 needed to meet the outcome; meeting 7 of the 11 outcomes. Green met outcomes in Branching, Common, Fundamentals, and Methods; meeting only 4 of the 11 outcomes.

Overall, both CS164-Gold and Gold significantly outperformed Green both on the overall exam score, and on the individual outcomes of the exam.

Table 6.2: CS 2 Review Exam (retake of CS 1 Final Exam) outcome mapping by student category. T-Test was used to calculate P value, and bolded values are significant at p <.10.

	CS164-GOLD		GOLD		GREEN		T-Test p-Value		
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Green vs Gold	Green vs CS164-Gold	Gold vs CS164-Gold
ALGORITHMS	3.51	2.06	3.00	2.15	2.92	2.07	0.42	**0.04**	0.11
ARRAYS	2.77	1.56	2.78	1.52	2.27	1.52	**0.04**	**0.02**	0.49
BRANCHING	4.52	0.67	4.27	0.71	4.05	0.74	**0.05**	**<0.001**	**0.03**
COMMON	4.02	0.88	3.70	1.34	3.20	1.15	**0.01**	**<0.001**	**0.06**
FUNDAMENTALS	3.83	0.74	3.71	0.83	3.02	0.80	**<0.001**	**<0.001**	0.21
INHERITANCE	3.73	1.34	3.40	1.50	2.70	1.49	**0.01**	**<0.001**	0.12
INPUT_OUTPUT	2.51	1.19	2.66	1.54	2.06	1.25	**0.01**	**0.01**	0.28
METHODS	4.67	0.77	4.44	0.99	4.58	0.86	0.19	0.27	**0.09**
OBJECTS	3.58	0.94	3.59	1.05	2.71	1.06	**<0.001**	**<0.001**	0.48
RECURSION	3.23	1.86	2.86	1.90	2.70	1.90	0.33	**0.04**	0.16
REPETITION	2.68	2.45	3.20	2.10	2.33	2.36	**0.02**	0.18	0.13

6.2 CS 2 - Topic Exams

Figure 6.1 shows topic exams for the students grouped by their CS 1 teaching methodology. After the review exam, students take an exam on every topic introduced in the course. The topic groups (modules) are as follows. While they are not designed to be cumulative, in programming most concepts end up being cumulative as concepts build on each other.

- Review Exam / Object Oriented Programming
- Recursion
- Testing / Lists / Generics
- Stacks/Queues/Priority Queues
- Expression Trees/ Binary Trees / Comparable
- B+ trees / Hashing
- Cumulative Final Exam

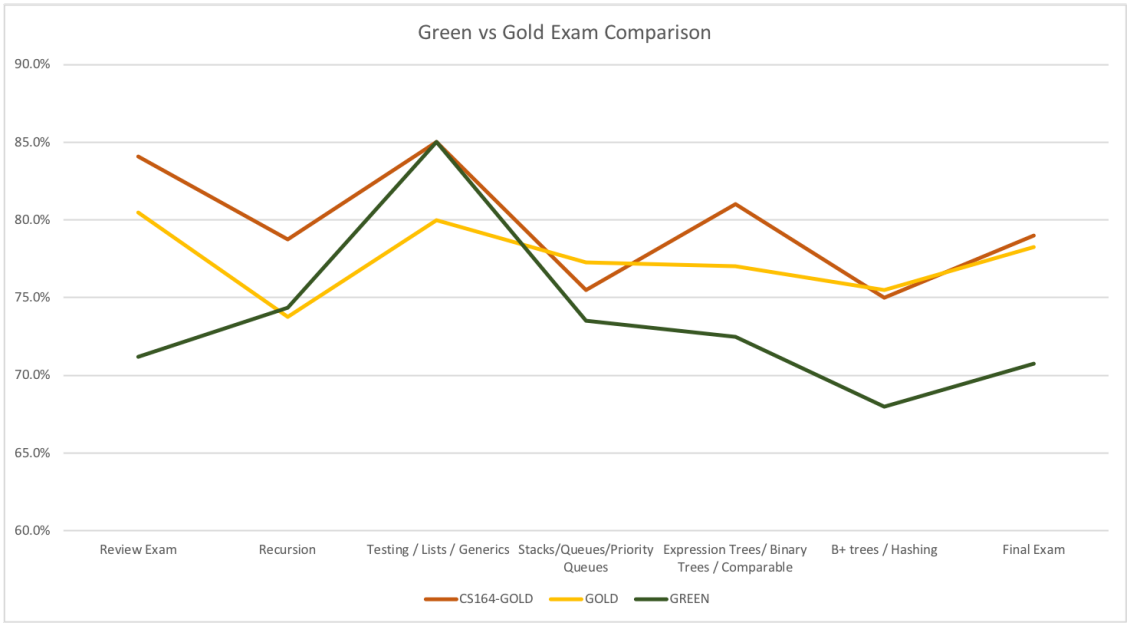


Figure 6.1: Module Exam Grades, Final exam for CS 2, broken up by CS 1 teaching methodology.

The initial review exam includes the 8% curve that the instructor applied so it is consistent across exam comparisons. This is separate from the results reported in Section 6.1 which were presented with raw values. There is not a change in the significance since the curve for the review exam was applied equally to all students.

For most students, there is often a drop on recursion due to it predominately being covered in a discrete structures course in our department, though since the Green group did so poorly on the review exam, they show a marginal increase. Testing, lists, and generics; and Recursion are the two modules that Green did better than Gold, showing a 5% and 0.6% increase respectively. However, the results were not significant at $p < .10$. Similarly, the results were significant in all other modules in which Gold did better than Green at $p < .10$. Gold did better than CS164-Gold on both Stacks and B+ Trees and Hashing, showing a 1.8% and 5% increase respectively. By the final exam, both Gold sections showed a difference of 7.5%-8.3% increase over the Green students, but only 0.8% difference from each other. Table 6.3 details the median grades, and presents both the difference between topic median grades, and the p-value calculated using a T-Test.

Table 6.3: Comparison of Median grades on Topic/Module Exams for CS 165. Difference calculations are bold when grades were significantly different at p <.10. In every case in which Gold did better than Green, the results were significant at p <.10, but Green was not significantly better than Gold due to high variation between test results.

	Module Exam Median Grades			**Difference (p value)**		
	CS164-GOLD	GOLD	GREEN	GREEN vs GOLD	GREEN vs CS164-GOLD	CS164-GOLD vs GOLD
Review Exam	84.1%	81.2%	71.2%	**10.0% (0.03)**	**12.9% (<0.001)**	2.9% (0.25)
Recursion	78.8%	73.1%	72.5%	-0.6% (0.38)	**4.4% (0.06)**	5.0% (0.14)
Testing / Lists / Generics	85%	80%	85%	-5.0% (0.47)	0.0% (0.41)	**5.0% (0.01)**
Stacks / Queues / Priority Queues	75.5%	77.3%	73.5%	**3.8% (0.07)**	**2.0% (0.01)**	-1.8% (0.12)
Expression Trees / Binary Trees / Comparable	81%	77%	72.5%	**4.5% (0.06)**	**8.5% (0.01)**	4.0% (0.10)
B+ Trees / Hashing	75%	75.5%	68%	**7.5% (0.07)**	**7.0% (<0.001)**	**-0.5% (0.01)**
Final Exam	79%	78.3%	70.8%	**7.5% (0.07)**	**8.3% (0.01)**	0.8% (0.14)

6.3 CS 2 - Assignment Categories and Final Grades

CS 2 broke up their assignments into the following categories, and weighted each category by the listed percent. By looking at the categories, it is possible to determine how students perform on different types of actions. Programming Homework and Labs both directly related to programming skill. Exams and quizzes, while connected to programming, are focused more on practiced recall and success at tracing code.

- Programming Homework (12%)

  Programming homework assignments tend to be harder coding assignments meant to be done outside of the lab environment.

- Labs (13%)

  Labs are meant to be done during lab time, are smaller, but are graded based on completion of the lab.

- Quizzes (5%)

  Quizzes are assigned at the beginning of the module, and students may retake them as many times as they want until the module exams (usually two week windows)

- Interactive Readings w / zybooks (10%)

  Are readings meant to be done before class sessions, focusing on both concepts and code.

- Module Exams (40%)

  Taken at the end of every module, and detailed in Section 6.2. They may only be taken once

- Final Exam (20%)

  Final cumulative exam.

Figure 6.2 shows the grades based on the assignment grouping for the course. In all categories CS164-Gold is higher than Green, and Gold is higher than Green in all categories but

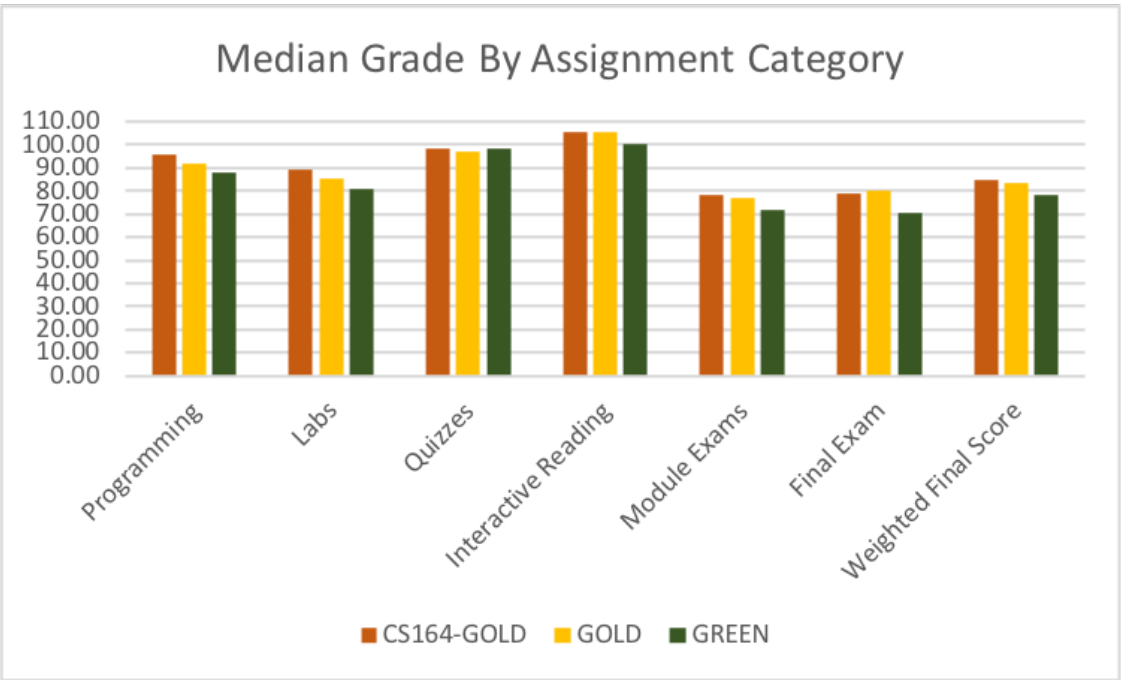


Figure 6.2: CS 2 grades broken down by average percent in each assignment category, compared across CS 1 groupings

Quizzes in which Green was .95% higher. By the end of the course, the weighted (and curved) final scores of the course were Green at 78.16 for the median score, Gold at 83.37 (a 5.22% increase from Green), and CS164-Gold at 84.82 a (6.66% increase from Green).

Table 6.4: Median grades by assignment type in CS 2. Bolded values show significant difference at p <0.10.

	Median Grade for Category			**Difference (p value)**		
	GREEN	GOLD	CS164-GOLD	GREEN vs GOLD	GREEN vs CS164-GOLD	GOLD vs CS164-GOLD
Programming	87.71%	91.67%	95.83%	**3.96% (0.04)**	**8.12% (0.01)**	4.16% (0.15)
Labs	80.96%	85.38%	89.23%	**4.42% (0.08)**	**8.27% (0.04)**	3.85% (0.91)
Quizzes	98.10%	97.14%	98.57%	-0.95% (0.17)	**0.47% (0.01)**	**1.43% (0.01)**
Interactive Reading	100.35%	105.45%	105.52%	5.10% (0.20)	**5.17% (0.01)**	0.07% (0.11)
Module Exams	71.60%	76.81%	78.27%	**5.21% (0.08)**	**6.67% (0.01)**	1.46% (0.11)
Final Exam	70.75%	80.00%	79.00%	**9.25% (0.04)**	**8.25% (0.01)**	-1.00% (0.23)
Weighted Final Score	78.16%	83.37%	84.82%	**5.22% (0.07)**	**6.66% (0.01)**	1.45% (0.14)

Overall, the Gold students outperform the Green students, and Green students are not "catching up" by the end of the course. Table 6.5 details the final grades for the CS 2 students. There is

a noticeable difference between Green and the Gold groups. Looking at C or higher, as C grades are required to move onto the next course in the curriculum, only 65% of Green students passed CS 2. Gold students had 80% pass while CS164-Gold had 90% pass the course. While there is a difference between Gold and CS164-Gold, the percentage is influenced by the group sizes, as there was only a 4 student difference between CS164-Gold and Gold, with 7 respective to 11 who ended up with a D or lower. At the same time Green had 32 students who did not pass. With that said, the Cohen D between sections is .37 a relatively small influence, but a much larger influence compared to the .02 relation between the Gold sections.

Table 6.5: CS 165 final grade distribution based on CS 1 Section. 90% of CS164-Gold passed with a C or above, 80% of Gold, and only 65% of Green. ANOVA between groups shows a p-value of .0007.

	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%
D	1	1%	2	4%	6	7%
F	6	8%	3	7%	17	18%
W	0	0%	4	9%	9	10%

Looking across CS 2, both CS164-Gold and Gold outperformed Green. Combining the various tests throughout the course, and the continued significance, this shows a significant difference in the student skill level coming into, and throughout the course.

Chapter 7

Discussion and Conclusions

In this chapter, we analyze and comment significant findings from the data presented in Chapter 5 and Chapter 6, while also adding to the discussion of the Spiral Design. Section 7.1 provides an analysis of the results with a focus on how the results between the different courses are related, and potential influencing factors. Section 7.2 provides a repeatable methodology for building a course around the spiral design, along with teacher guidelines when implementing the design. Additionally, further research questions are explored that came from the Spiral Design and how those questions can benefit the future CS education community. Finally, we summarize the paper's research questions and conclusions in Section 7.3.

7.1 Results Discussion

The primary goal of the Spiral Design was to see if restructuring around the principles of memory and recall would show benefit to students' performance in the restructured course, and increased recall in the following course. In Chapter 5, we evaluated the final exam for CS 1. The choice to evaluate the final exam came because once the course was restructured with the spiral pedagogy, that was the only point which aligned between the two sections. In addition to the exam, student performance at the topic/outcome level was tracked using the exam as a means to track them.

The exam itself proved to be a difficult exam, with the highest median grade being 65.7% and the highest average being lower at 60.1% for students taught using the spiral methodology who had prior programming experience going into CS 1 (CS164-GOLD). The no-prior programming students taught using the spiral pedagogy (GOLD) scored 65.5% for their median grade, and 59.5% average, while the students taught using the traditional pedagogy (GREEN) scored only 56.4% median and 53.5% average. It was found that the difference between both Gold sec-

tions and Green was significant, while the difference between the two Gold sections was not significant.

Two major points stand out in these results. The first is the obvious that Gold is scoring higher on the exam than Green. However, there is a rather large standard deviation for both groups, causing the Cohen's *d* to show only a small influence. While students struggled with the COVID-19 pandemic, it was witnessed across courses that some students struggled more with the online environment than others; causing increased DWF rates department wide. With that said, the significant difference between Green and Gold, and the Cohen's *d* showing some influence helps solidify that the new pedagogy is having an affect on student performance within the current course. Furthermore, most research design decisions focused on biasing the data towards the Green section. This may actually be reducing the overall influence of the Spiral Design, since it had a higher hurdle to overcome compared to the traditional design to show positive influence in student outcomes.

The second major point comes from the historical work done by Wilcox and Lionelle [3]. By the end of CS 1, before the spiral pedagogy, students with prior programming significantly outperformed students with no-prior programming experience. However, once the Spiral Design was implemented, students with no-prior programming ended up performing equivalent to those with prior programming. This was an unexpected outcome as past experience showed it usually took two semesters before those with no-prior experience caught up to those with prior experience. The caveat to this influence is that the prior programming experience students often started higher on previous exams, and it wasn't until the end of the semester that the Gold students began performing equivalently. Also, the number of CS164-Gold students going onto CS 2 compared to Gold is a noticeable difference, especially for the women. As such, while the two groups are equivalent in performance at the end of the semester, the slight differences between the groups add up across the semester. Also, past experience is a major influence in students choosing to continue onto more CS courses.

Diving more into the exam, Section 5.3 details the individual learning outcomes/topics expected of the students in CS 1. While there are a number of ways mastery of topics was presented, one of the more telling ways is Table 5.6. Only the top 1% of Green students demonstrated mastery in all 11 learning outcomes and only 5% showed mastery in 10 of the 11 (i.e. 90% of the course content). For both Gold groups 12% of the students showed mastery of all 11 topics. In a perfect world we would want 100% mastery of all topics before students move onto future courses, but such an expectation was not designed into the course, nor were the repetition opportunities available. With that said, only 1% showing mastery is abysmally low. Even at only 90% mastery of topics, the Gold sections show mastery at 29% for CS164-Gold and 23% for Gold, a drastic difference from the 5% of Green. This supports that Gold is doing far better on their overall understanding of the topics.

While the quantitative analysis of student performance is straight forward, the student survey presents a few curious and unanswered questions. Looking at the four pre and post questions, students in the Gold sections ended up liking CS a lot more by the end of the semester than what they thought at the beginning of semester. However, they either lost confidence in their own ability to program, or they gained a better understanding of their ability answering the questions more realistically. Especially for the male students who had a drop in their opinion about their ability to learn CS, it is possible they simply didn't have a good enough grasp of the topic to answer that question with validity. A common saying in the department is that Computer Science is not liking computers, and that saying stems from the experience of students coming into CS because they like computers causing them to drop when they find the emphasis on problem solving. When looking at it by gender breakdown, the increase in women's interest is noticeable compared to the males. This may also be a major contributor to the difference between women in Gold compared to women in Green who choose to go onto CS 2, a 19% difference in the number of women in Gold who choose to go on is a noticeable increase. However, like does not always mean increased scores, or does it?

The MSLQ answers reinforced the idea that opinion does not equate to performance, and that Green students often ranked themselves higher on their study skills compared to Gold even though Green performance was lower. The Gold groups would rank themselves higher in their motivational beliefs, which did have a small correlation to performance. This ties into the two questions on liking Computer Science in the pre and post question survey. Gold was higher in their interest, and that also seemed to translate in their self-reflection to their motivation in performing well. Based on these results, increasing student interest has the ability to affect both student's desire to take the following course and their performance in the current course. Though this study showed interesting results in this direction, a more comprehensive study would be beneficial; focusing on increasing student motivation and seeing if there is a direct relationship to retention and grades.

The students who chose to continue onto CS 2 were all placed into the same course with the same instructor. Their labs varied only in self selection of times, which means labs also had a variety of students from all three groups in them. In the work by Wilcox and Lionelle, it was found by the *end* of the semester that no-prior and prior programming students were equivalent, though arguably that could be because only the best of no-prior programming took CS 2 given the high attrition rates in the traditional design. In this case, the Gold and CS164-Gold students are already showing equivalency to each other, and Gold is already showing higher retention of students than the traditional no-prior programming course (Green). Going into the analysis that change does matter in two ways. One, it is important that Gold and CS164-Gold stay mostly equivalent between each other, and two, Green may be only sending on the "best" of their students to take CS 2. However, with that said, the size of the Green section was noticeable enough that the number of Green students is still more than the Gold students. Furthermore, Green students include students who took the traditional methodology in other semesters and who may have delayed taking CS 165, which increases their numbers.

One of the stronger tests in CS 2 is the review exam. Students are given the same final exam as CS 1, but due to the question banks, they would end up with mostly different questions. Run-

Table 7.1: Topic exam correlation with the final grade.

Topic Exam	Final Score
Review Exam (Curved)	0.58
Recursion	0.68
Testing / Lists / Generics	0.74
Stacks / Queues / Priority Queues	0.84
Expression Trees / Binary Trees / Comparable	0.81
B+ trees / Hashing	0.85
Final Exam	0.91

ning a correlation, it was found that the review exam has a 0.58 influence on the final grade (see Table 7.1), so it really was focused on a review of their knowledge from CS 1 helping answer the question "of how much was recalled". Both Gold and CS164-Gold performed similarly, and they both outperformed Green significantly. By showing similar results to the Final exam in CS 1, and increasing just slightly in the difference between median and average grades, it shows increased recall of the topic matter after the winter break. If we look at the Cohen's *d* as measurement of the influence of the recall, we have large affect size of 0.89 between CS164-Gold and Green, and a medium affect of 0.69 between Gold and Green. This means that not only are the grades significantly different, there is a noticeable influence affecting the groups. Combined, the p-values and Cohen's *d* present strong evidence that the Spiral teaching methodology is helping students retain the information, and present it again on the exam. It is also important to note that while CS164-Gold is slightly ahead of Gold, the results are not significant, and the Cohen's *d* is negligible (.02).

Looking at the correlations it is also worth noting that the exams with the highest correlation to the final grade, outside the final exam, are also the two exams that Gold scored the highest grades, even above CS164-Gold. Essentially, Gold was able to demonstrate a strong understanding in both those two areas, which had an .84-.85 correlation to the final grade. This could contribute to the higher grade at the end.

Having grades slightly above, but not significantly above across the course really added up for the overall grade at the end of the course. Gold ended up with a fair percentage of B grades

while CS164-Gold ended up with the most A grades out of everyone. There was also a significant difference between Green and Gold, but not Gold and CS164-Gold. It is a clear reminder that even though the difference is small throughout the course, it continued to add up, and seeking even small differences can have an affect on whether students pass a course or not.

A common critique of the using practiced recall is that while it can help students perform better on exams, are we truly enabling and improving their programming ability? While the Spiral Design includes far more than practiced recall, it is worth noting that as shown in Table 6.4 both Gold groups outperformed Green significantly in programming and labs. While the Cohen's *d* only showed a small influence (0.31 Green vs. Gold, and 0.52 vs CS164-Gold), that small influence added up. It is also noticeable that Green had a lot more variation with nearly double the standard deviation in grades (26 compared to 13). The labs showed the same range as the programming assignments, though the grade differences were a bit higher. Through these results we are able to show that the Spiral Design does indeed help improve programming ability, though it would be a good future research project to see which aspects of the design influence programming capability the most. For example, is it the practiced recall or the labs that intentionally practice interleaving? Since this experiment looked at the overall design, the smaller components are impossible to break out from the overarching design. Arguably, breaking the components out would not have as strong an influence, as probably there is a some contribution between all the techniques that are showing higher performance in CS 2, including programming performance.

Overall, there is enough data to strongly indicate that the spiral pedagogy is having an influence on students and their performance, but there are still some questions on exactly how strong that influence is from each of the components of the design. This provides a potential area for future research, as attempts at adopting individual components could be measured. For example, just adding the opportunity for practiced recall, as compared to spacing out the actual topics of the course or forcing the labs to include interleaving of topics to encourage discernment.

7.2 Design Considerations

Throughout two years of using the Spiral Design, first in CS 0 and then in CS 1, there are a number of insights which developed that are causing us to slightly modify the design. This section details both the process used in the design, feedback about the design, and suggestions for modification. Additionally, I dive into future research ideas that came from the implementation of the Spiral Design. This section can be used both as a guideline of what to do as well as considerations for future research.

7.2.1 Converting to a Spiral Design

When looking at a course to convert to a Spiral Design, the first step was looking at the current topic list of the course. This list is often a bit more expansive than the learning outcomes, but in good course design it may overlap. For CS 1, it was the 15 week schedule as topics. For CS 0, it was a focus on the learning outcomes from the CS 0 course.

After the topics have been designated, we then broke the topics up into sub-topics, aiming to develop at least two, ideally three subtopics. Table 7.2 shows the breakdown of the Conditions / Logic topic. In the traditional model these topics are all taught within one week (sometimes switch statements are delayed), and then students move onto new topics with the expectation to know or return to the topics themselves to gain clarity.

Table 7.2: Example of breaking down conditions into sub-topics. Each sub-topic is a natural breaking point.

Topic	**Sub-Topic**	**Dependency**
Conditions / Logic	condition statements simple primitive comparing, no Boolean operators	Booleans
Conditions / Logic	Boolean operators (&&, ||, etc)	condition statements
Conditions / Logic	Conditional Expressions (ternary statements)	condition statements
Conditions / Logic	Switch Statements	condition statements

The choice made for the spiral method was to break each topic up with a three week space between topics using three iterations by combining Conditional Expressions and Switch statements. These were combined simply due to room in the overall 4 units. The goal was to give a few weeks of experience using simple expressions before adding complex expressions, and then adding switch and conditional expressions as additional tools. There is repeated emphasis that often when we return to a topic, we are gaining more tools to make our code easier.

However, even this simple example ran into issues within the books for the course. Most books lump boolean operators with the boolean primitive. However, the boolean primitive needs to be introduced when working with condition statements. As such, we were forced to introduce boolean primitive only in lectures and labs and not the initial reading. It was rather surprising the number of times small structure considerations come up in the books for programming that assume or teach content simultaneously making it hard to split topics. For example, when files are introduced both reading and writing are often introduced immediately and then more in depth of each. An ideal split would have been to separate reading and writing so students can practice reading files for a few weeks, before writing files. While we were able to make this split, there was some overlap in the interactive reading assignments. Additionally, it would have been ideal to have exceptions separate from file reading, but most books introduce them together due to the necessity of needing exception catching in Java. While these are more obvious cases, the less obvious case is the assumption that students know both do-while and while loops at the same time and they often use them as examples to show the different types of loops. In the Spiral Design, we focus on one subtopic, like a while loop, and then use the comparison a few weeks later when do-while loops are introduced.

When breaking topics into subtopics, careful consideration needs to be taken into account with how that division really is within the resources. Probably the greatest limiting factor in the Spiral Design is that we are often not used to thinking about certain topics separately. Instead, we treat them as one giant lump of information and looking at scaffolding that information across multiple weeks is very different from what most people were taught over the years.

When designing labs for the Spiral Design, careful consideration needs to be placed into asking too much of students right away. As such, once topics were designed, we focused on first building the book (or zybook in our case) that matched as closely as possible, and then focused heavily on the labs. With the Spiral Design, students are introduced to topics in what is essentially a rapid fire sequence, and in order to balance that, it was found the labs when first started needed to provide the design considerations and layout. Essentially, we created a template to follow in which students were filling in each method like they were answering a math problem. This let the students focus on the simple side of the topic while also seeing and experiencing future topics, such as design questions and how methods truly interact.

When designing lectures and assignments in general, this same idea of giving students a glimpse of the future was utilized. Provide examples based only on what they know, but in context of how they can be applied in the future. For example, with conditionals / if-statements, we used nested if-statements to figure out if a value was within a range of two numbers.

```
if(x < 100) {
    if(x > 0) {
        System.out.println("X is greater than 0, and less 100");
    }
}
```

When we introduced complex conditionals, the code became much simpler for something the students had already been working on the past few weeks.

```
if(x < 100 && x > 0) {
    System.out.println("X is greater than 0, and less 100");
}
```

This intentional design choice helped students see connections between the various units and the return to topics.

While this topic division + spacing of topics worked well for CS 0 and CS 1, there would be other ways to breakup the content of any course that may look very different. For example, in

CS 2, one could also break the course topics down into subtopics, but instead of it being simple stacks, and more complex stacks, it could be introduction to stacks and then implementing stacks after they have had a few weeks of using them via commercial (SDK) libraries. Such a design would still force interleaving of topics, spacing of the main topic, and include an aspect of student discernment of which tool is correct for which situation and then eventually learning the details of how that tool is implemented. At its heart the steps are still the same:

- Define

  Define course topics / objectives.

- Divide

  Break topics into subtopics, ideally with clear lines.

- Dependencies

  Determine dependency between topics.

- Restructure

  Restructure order based on subtopics, returning to a topic every 3-4 weeks by way of adding an additional sub-topic of that topic. Another way to put it, determine what not to teach, so you can come back to the topic in 3-4 weeks.

- Build

  Build assignments based on students first focusing on concepts, then building into more complex connections later in the course.

- Intentional Design

  Design lecture and content that "hints" at the next subtopic to be covered, so that students are prepared when the subtopic brings them back to the primary topic.

The hardest part about implementing this design is that the tools, such as the book, don't align. The other difficulty is taking the time to critically reflect on which topics are essential

right away and which ones students can wait on so the instructor has built in topics to return to.

7.2.2 Teaching Feedback and Suggestions

When implementing the Spiral Design and working with students there is one suggestion that stands above the rest, **encourage a growth mindset**. The entire Spiral Design focuses on students coming back to topics and that they should not punish themselves if they don't get it right away; that everyone is learning. However, students come into the course already taught that they must master a topic before moving on and that if they don't have it mastered right away they must be failing. The first four weeks consisted of reminders every few days that they can do it, that they should focus on a growth mindset, and that we will indeed come back to the topic. Needless to say, working against preconceived notions of immediate mastery (no matter how false they actually are) is a difficult barrier to overcome. When running the Spiral Design, we spend a lot of time asking both the TAs and instructors to encourage students and often that encouragement made all the difference.

The second major suggestion is for instructors to use technology to their advantage. In order to get students to read before coming to lab, we setup an enforced rule that the reading had to be at least submitted, even if they could go back to redo it, before they could open up the lab assignment. This one change in CS 0 nearly doubled our initial scores in lab, and reduced the amount of questions. Partially, this is due to how the course is structured, and not specific to the Spiral Design. Our courses were designed that the interactive reading participation problems were to be done before lecture. Students then attended a lecture with heavy peer instruction and then they went to lab the following day to apply what they learned. This model requires a lot of reminding on the part of the instructor, as like most classes, students struggled on completing the reading.

Related to technology, while we designed the Spiral version of CS 1 to have a Knowledge Check after every lecture, even though they were asked only 1-3 questions, students found the

number of items due every week to be overwhelming. We have since reduced the number of knowledge checks to once a week, with 4 questions. This effectively gives them one or two questions per topic covered that week and causes the knowledge checks to have more variation between attempts. We also moved to requiring a 3/4 of the knowledge checks to open up the next module (week/labs/etc), as a way to keep students motivated and working at pace with a course that has flexible due dates for some assignments.

The last major suggestion, double check all examples. It was very easy to have a lab or class example have a topic we had not yet covered in the course. With our own experience, we are very used to doing the shorter path in programming. Since the Spiral Design is not teaching all the tools, it does not mean the students covered all the topics. To the instructor, they think nothing of having += in their code. However, students don't cover += right away as that isn't covered until week 5, after they have been using the longer form of 'val = val + something' for a while. Careful consideration and attention to detail needs to happen to make sure there isn't topic creep confusing students.

Overall, there aren't very many pedagogical concerns to consider when implementing the Spiral Design. The most common essential we kept coming back to is motivating students and reminding them when we will return to a topic. Most of the students have that "aha moment" as it starts making sense the second or third pass through a topic. Simply put, it is a course design that makes the course feel easier, not harder even though more topics are introduced. In order to accomplish that feat, the learning curve feels steeper at first as compared to a traditional design.

7.2.3 Future Research: Designing Across the Curriculum

The inspiration for the Spiral Design originally stems from the Spiral Curriculum Design, a process largely attributed to Jerome S. Bruner [114]. The theory is that concepts are repeated throughout a curriculum with deepening levels of complexity at every pass. While some college departments have implemented this curriculum design [115], it is largely unknown within

a university domain as a curriculum design, though it has been suggested for K-12 education [116]. A future area of research would be to look at the ACM/IEEE Core CS Curricula standards and lay them out using a spiral design across the core curriculum.

The ACM Body of Knowledge topics provide an overview of what they encourage CS programs to have in Tier 1, Tier 2, and Elective categories for topics. These categories are not defined by courses intentionally, and there is even encouragement within the document to come up with new and creative courses that go across the standards. What traditionally happens is that standards are grouped into a single course or maybe across two courses in a CS curriculum. For example, at CSU the Algorithms and Complexity knowledge area has most all of its topics covered in CS 320 (Algorithms Theory and Practice), both from the familiarity level through the usage level of understanding of the learning outcome topic. While some topics are introduced in CS 165 (CS 2: Data Structures), and some topics are reinforced in CS 220 (Discrete Structures), there is no specific alignment across the topics.

In order to implement a Spiral Design, the lower division courses would focus more on the familiarity topics with some assessment and usage mixed instead of focusing on bringing students to mastery levels of "foundational concepts". For example, CS 1 would incorporate best, worst, and expected cases of using algorithms; while CS 220 would then focus on assessments of those uses while introducing graphs to students. CS 320 would then focus on having students apply the techniques with intentional relationships back to when students last covered the topics in the previous courses. It could then apply assessments of graph techniques and some usage.

Another example would be Data Structures. Many departments, including our own, treat data structures as the second course in the sequence in which students must learn the foundation of all data structures to be used in future courses. This is asking students to completely master the topic of data structures before moving onto other courses, yet we know students don't immediately master topics with practice. I would argue that practice should be spread across multiple semesters and years. A way to accomplish this would be to swap the ordering of

courses, so that Data Structures is later in the sequence, then a CS 2 course would be a software engineering course. That software engineering course would naturally build upon the concepts of CS 1 while introducing students to the use of data structures. They would have the opportunity to be introduced to algorithms and Big-O complexity, while diving deeper into software design. They would be introduced to the data structures without being asked to program those data structures in depth. Instead, they would gain knowledge of when to assess the best usages of which data structure. This would then take the more difficult topics in CS 2, such as recursion, and allow those topics to continue to be spread across multiple courses throughout the core curriculum until mastery of the topic is expected of students.

While there were aspirations of developing a full spiral curricula for this dissertation, the realism of the problem prevented such aspirations. When looking into it, many of the topics throughout the core were not carefully defined, which would mean the first pass would be focusing on aligning the current courses with the ACM/IEEE core knowledge areas. Those knowledge areas would then ideally need to be linked to more concrete outcomes that follows closer to Bloom's taxonomy [81], which would then allow for dispersal of outcomes across new course designs. Such would involve numerous committees, and arguably multiple years, to develop a full curricula that would be very different than how people currently visualize the CS sequence. It would have to focus on how to teach students to think and look at problems while building examples across courses for students to create mental links back to previous topics.

With that said, this would be a good area of future research and development, as a number of programs could benefit from a spiral curricula. Careful consideration will be needed in how such curricula would affect transfer students, and also accessibility of the program for non-majors (students only taking one or two courses). Simply, just because a prerequisite says a student was introduced to the topic, it should not be assumed they have that introduction, and resources would need to be developed to support the curriculum design.

A simpler approach may simply be to take key paired courses (CS 220 and CS 320 for example) and restructure them so in CS 220 they follow individual spiral curriculum techniques,

purposely having students introduced, and then reintroduced to topics both within the course and between the courses. Another simpler starting point is to carefully evaluate the CS 2 and CS 3 sequence in most departments, and ask, "Is it better to use something before understanding the fundamentals of how it works, or is it better to understand the fundamentals of how it works before using it?" This is another open research question in our field.

7.2.4 Future Research: Standards Based Grading

In the book, "Grading for equity: What it is, why it matters, and how it can transform schools and classrooms." Feldman encourages a critical look at the grading process, and how our current 0-100% scale is not a very equitable or even mathematically accurate means of assessing students. The argument is that student grades should focus on formative assessments, assignments that students can and should redo until the assignments are correct, and summative assessments, assignments that students are demonstrating what they have learned in the course. Feldman then recommends using a scale from 0-4, matching the GPA scale based on the outcomes / standards for each assignment [117].

Standards based grading has been around for a number of years, and it focuses on the idea that student grades should be based on them meeting the standards of the course. It is a very growth mindset focused grading system, in which students are given opportunities to show that they either meet or exceed the standards, and if they don't they have to fix areas they don't meet. Using Feldman's example, students would be graded on the mode of the standards. For example, if the student had a majority of 4s in all topic areas (he suggested grading each standard as 4 exceeds, 3 meets, 2 almost meets, and 1 needs improvement), they would receive an A in the course, a majority of 3s would equate to a B in the course, and so on. Another way to implement standards grading is based on the number of standards students met, so if they meet 90% of the standards they have an A in the course. The biggest complaint about this model at a college level is that students need to have time to go back to a topic, and frequently are difficult to build into a course that is only a semester long teaching an entire content area.

The Spiral Design compliments standards based grading as the course designer has already built in points to go back to topics. With every pass of the spiral, in a standards based grading environment, students are given the opportunity to demonstrate mastery in that topic area. Additionally, since programming is cumulative, that demonstration can continue to happen throughout the semester if it is planned. Eventually, the formative assessments would become summative as the end of the semester approaches.

While formative and summative grading is possible, and already currently being explored, true standards based grading is currently limited by the tools used in Computer Science, especially tools used in introductory level Computer Science courses. Most notably, auto-graders often only report the number of passed and failed tests. Those that integrate directly with an LTI often report it as a point score (including partial points). Standards based grading often uses whole numbers, and the tool would need to report points directly as meeting outcomes on an individual test basis. Such a tool does not currently exist. For standards based grading to be feasible for large class sizes, like those shown in lower division CS courses, such tools would need to be developed.

The development of auto-graders that can report by standards and outcomes being met, an analysis of the accuracy of that reporting, and standards based grading in Computer Science courses are all areas of future research. The Spiral Design naturally works with the idea of standards based grading, as can be seen with the outcome performance, but without these tools in place it will still be limited.

7.2.5 Future Research: Software Application - Interactive Quizzes

When working with knowledge checks, the limitations of the quizzing system in Canvas, or most learning management systems, quickly became apparent. These quizzes end up focusing on reading code because all we are able to ask are questions that involve reading code and not interactive coding sessions. While questions can be carefully worded to help improve a students' coding ability, there is something to be said about asking students to recall small bits of

code to write, test, and see the output. There are websites, such as codingbat.com [118], that help with interactive coding, but they are based on a traditional design that may have assumptions about student knowledge. Additionally, those practice sites are not built into the course design in which students must work on knowledge checks in order to open up the next module.

A tool that would be ideal, not only for the Spiral Design, but for other courses would be a quizzing based tool that can interleave both code reading and code writing questions. Furthermore, the tool would ideally be focused on topics and standards based grading so that students will have questions on previous material interjected into the mix of current questions. A short list of specifications could be as follows:

- LMS Quiz Tool Interaction

  Similar to the new quizzes, the tool would ideally need to interact with LMS systems as another tool to represent quizzes.

- Outcome Alignment

  Questions should be able to be aligned with outcomes/topics for reporting of grades.

- Outcome Mapping

  By having outcomes mapped to each other, the tool could interleave past topics into current topics forcing the student to have a refresher of a past topic (potentially from a past course) to help them understand the newer topic.

- Question Banks

  Based both on outcomes and difficulty (pass in the spiral), questions should be able to be assigned to question banks where questions are randomly pulled from the banks.

- Variable Input Questions

  Some questions should have the ability for variable inputs, allows a limited number of auto-generated questions. A unique feature more specific to software engineering, would be applying mutations to questions in a manner that students would be able to detect and fix code broken by the mutation.

- Coding Questions

  Allow an interactive limited IDE, and the ability for students to have immediate feedback once they submit the question for grading. These questions should ideally be limited to sub-parts of an entire program, and not require all the wiring that needs to happen for some languages. For example, they write a method that simply loops through an input String, and returns every other character.

- Code Question Mutation

  Ideally, once a code question has been created, parameters for expected input and output could be put into place, allowing the same question to be used without much change to the original testing protocol. Such a system could be "question", "input range for each parameter", "output range".

- Analytic Report Generation

  Ideally students would be able to quickly see information on their study habits, how they are doing on topics / outcomes, and suggestions for improvements in how they study. This would also be useful for instructors, especially aggregate reports between courses that could report areas of weakness of the incoming cohort with suggestions for intervention.

- Intervention Suggestions

  If a student continues to struggle in a topic area, since the tool is mapped to outcomes, it may also be possible to map suggestions for intervention (videos, additional readings, study group times, etc) to those outcomes. At certain intervals, such interventions can be recommended to the student based on their performance.

- Sharing of Question Banks / Student Question Generation

  Ability for students or other instructors to generate their own questions, and share questions to others in the course and across question banks. Such a system would also need curation and confirmation on a course by course basis.

This type of tool would help students practice spacing, interleaving, and practiced recall while focusing more on problem solving over memorization. It also could be developed in iterations, initially focusing on a quizzing tool that also allowed coding questions at first, and build up from there to a full fledged study tool based on the same principles used in the Spiral Design.

7.3 Conclusion

In Chapter 4, three guidelines for success were presented.

> Q1: Given that going to spiral allowed us to add content without reducing learning, what happens when CS 1 is designed around the Spiral Model without adding content? Will student performance in the current course increase?

Yes. Students who learned via the Spiral Design outperformed their peers in the traditional design on the final exam by 9%. This is a significant increase in performance, showing an increase in performance in the current course. Furthermore, when tracking at the individual outcome level, Green students failed to meet the majority of the outcomes, while Gold students met all but three of the major outcome/topic categories tracked. Lastly, when the same exam was presented to students in CS 2, as a review of CS 1 content, Gold students outperformed Green students by 10%. The spiral pedagogy showed moderate influence on Gold vs. Green with a Cohen's $d = .69$, and a high influence on CS164-Gold vs Green with a Cohen's $d = .89$.

> Q2: Given CS 2 is often the great equalizer in student performance no matter their programming background, will students who learned via the Spiral Method in CS 1 outperform students in CS 2 who learned with the traditional methods in CS 1?

Yes. Gold students outperformed Green students in the following CS 2 course in multiple ways. They start by retaking the CS 1 final exam for review, showing an increase in retention after the winter break, increasing the gap between Green and Gold to 10%. They also outperformed on

all assignment categories showing not only increased capability in testing, but also in the programming skills needed to be successful in CS 2. Lastly, by the end of the course 15% more Gold students passed with a C or above than Green students, with Green showing 65% passing with a C or above and Gold showing 80% of their students passed. Simply, not only do Gold students start off higher than Green, the Green students never truly catch up to Gold in performance.

> Q3: Any modern changes should not hurt inclusion and retention of students between courses, does the Spiral Method at least encourage equal, if not improved retention of students who come from underrepresented backgrounds to continue onto CS 2 as compared to the traditional method?

Yes. We looked at the number of students who passed CS 1 in each group, and who chose to go onto CS 2. Gold showed a noticeable increase in women who chose to go to CS 2, increasing by 19.2%. Overall, the Spiral Method continues to encourage students to move onto CS 2, meaning the changes didn't affect retention per the condition for Q3. Furthermore, it actually had the opposite affect, increasing student retention, so we see less student attrition between the two courses.

With all the benefits of the Spiral Design, there is little reason to not implement this type of design for CS 1 and to look at how it can affect the content of courses. At the same time, such an implementation is not a trivial task. The design works against the traditional way our courses are taught, making resources difficult to find. Furthermore, it isn't at first the most common way we think when we design a course. While most course design looks at the topics and how they layout, the Spiral Design adds the stage of looking at the topics by breaking them up into additional cycles. Making us look at how we can integrate the spiral across topics. This type of implementation may take training and additional time spent on resource development.

In addition to the affects of the Spiral Design, additional research topics were discussed related to the Spiral Design and outcome tracking of students. These topics included looking at how to apply the design to other courses, how to apply the design across a curriculum, the implementation of standards based grading which the design naturally compliments, and tools

that could be implemented to help promote the same techniques used in the Spiral Design. These topics help solidify the benefits of the design; both to the teaching and future research communities.

Looking back at the Spiral Design, it may be possible to argue that it is simply good design to take learning outcomes and design a course across multiple iterations of those learning outcomes. With each pass students develop greater mastery of the topic as they go deeper into the topic. Looking at the design in this manner is to simply state that it is a good iterative design practice focused on measuring student outcome success.

Bibliography

[1] Brett A Becker and Thomas Fitzpatrick. What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1011–1017, 2019.

[2] Raad A. Alturki. Measuring and improving student performance in an introductory programming course. *Informatics in Education*, 15(2):183–204, 2016.

[3] Chris Wilcox and Albert Lionelle. Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course. pages 80–85, 2018.

[4] Zoë J Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, and Bruce DeBruhl. Mixed approaches to cs0: Exploring topic and pedagogy variance after six years of cs0. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 20–25, 2018.

[5] Jens Bennedsen and Michael E. Caspersen. Failure rates in introductory programming. *SIGCSE Bull.*, 39(2):32–36, June 2007.

[6] 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. Association for Computing Machinery.

[7] 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.

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

[9] Richard Bornat and Saeed Dehnadi. Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 53–61, 2008.

[10] Andrew Luxton-Reilly. Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 284–289, 2016.

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

[12] 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. *Psychol. Bull.*, 130(2):261–288, 2004.

[13] 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 *Proc. 2016 ACM Conf. Int. Comput. Educ. Res.*, pages 211–220, New York, NY, USA, aug 2016. Association for Computing Machinery, Inc.

[14] Daniel Zingaro. Peer Instruction Contributes to Self-Efficacy in CS1. In *Proc. 45th ACM Tech. Symp. Comput. Sci. Educ.*, pages 373–378, New York, New York, USA, 2014. Association for Computing Machinery.

[15] Ashish Amresh, Adam R. Carberry, and John Femiani. Evaluating the effectiveness of flipped classrooms for teaching CS1. In *Proc. - Front. Educ. Conf. FIE*, pages 733–735, 2013.

[16] Laura Toma and Jan Vahrenhold. Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs - A case study. In *ICER 2018 - Proc. 2018 ACM Conf. Int. Comput. Educ. Res.*, volume 10, pages 1–10, New York, NY, USA, 2018. ACM.

[17] 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. *J. Educ. Comput. Res.*, 19(4):367–381, dec 1998.

[18] Gerry Geitz, Desirée Joosten Ten Brinke, and Paul A. Kirschner. Changing learning behaviour: Self-efficacy and goal orientation in PBL groups in higher education. *Int. J. Educ. Res.*, 75:146–158, jan 2016.

[19] 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.

[20] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming a review and discussion. *Computer science education*, 13(2):137–172, 2003.

[21] Uolevi Nikula, Orlena Gotel, and Jussi Kasurinen. A motivation guided holistic rehabilitation of the first programming course. *ACM Trans. Comput. Educ.*, 11(4), November 2011.

[22] Ironman Draft. Computer science curricula 2013. *ACM and IEEE Computer Society, Incorporated: New York, NY, USA*, 2013.

[23] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. In *Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 204–223. ACM New York, NY, USA, 2007.

[24] Michael De Raadt, Richard Watson, and Mark Toleman. Language trends in introductory programming courses. In *Proceedings of the 2002 Informing Science+ Information Technology Education Joint Conference (InSITE 2002)*, pages 229–337. Informing Science Institute, 2002.

[25] Michael de Raadt, Richard Watson, and Mark Toleman. Introductory programming: what's happening today and will there be any students to teach tomorrow? In *Proceedings of the 6th Australasian Computing Education Conference: Computing Eduction 2004 (ACE 2004)*, volume 30, pages 277–281. Australian Computer Society Inc., 2004.

[26] Ernie Giangrande Jr. Cs1 programming language options. *Journal of Computing Sciences in Colleges*, 22(3):153–160, 2007.

[27] Robert Michael Siegfried, Daniel Greco, Nicholas Miceli, and Jason Siegfried. Whatever happened to richard reid's list of first programming languages? *Information Systems Education Journal*, 10(4):24, 2012.

[28] Neil CC Brown, Sue Sentance, Tom Crick, and Simon Humphreys. Restart: The resurgence of computer science in uk schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):1–22, 2014.

[29] Katrina Falkner, Rebecca Vivian, and Nickolas Falkner. The australian digital technologies curriculum: challenge and opportunity. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, pages 3–12, 2014.

[30] Tim Bell. Establishing a nationwide cs curriculum in new zealand high schools. *Communications of the ACM*, 57(2):28–30, 2014.

[31] Michael E Caspersen and Palle Nowack. Computational thinking and practice: A generic approach to computing in danish high schools. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 137–143. Citeseer, 2013.

[32] John Konvalina, Stanley A. Wileman, and Larry J. Stephens. Math proficiency: A key to success for computer science students. *Commun. ACM*, 26(5):377–382, May 1983.

[33] Nelishia Pillay and Vikash R. Jugoo. An investigation into student characteristics affecting novice programming performance. *SIGCSE Bull.*, 37(4):107–110, December 2005.

[34] Patricia F. Campbell. The effect of a preliminary programming and problem solving course on performance in a traditional programming course for computer science majors. In *Proceedings of the Fifteenth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '84, pages 56–64, New York, NY, USA, 1984. Association for Computing Machinery.

[35] Charles Dierbach, Blair Taylor, Harry Zhou, and Iliana Zimand. Experiences with a cs0 course targeted for cs1 success. *SIGCSE Bull.*, 37(1):317–320, February 2005.

[36] Veronica A Lotkowski, Steven B Robbins, and Richard J Noeth. The Role of Academic and Non-Academic Factors in Improving College Retention. ACT Policy Report. *Am. Coll. Test. ACT Inc*, 2004.

[37] Brenda Cantwell Wilson. A study of factors promoting success in computer science including gender differences. *Computer Science Education*, 12(1-2):141–164, 2002.

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

[39] Jane E Caldwell. Clickers in the large classroom: Current research and best-practice tips. *CBE—Life Sciences Education*, 6(1):9–20, 2007.

[40] Thomas Eberlein, Jack Kampmeier, Vicky Minderhout, Richard S Moog, Terry Platt, Pratibha Varma-Nelson, and Harold B White. Pedagogies of engagement in science. *Biochemistry and molecular biology education*, 36(4):262–273, 2008.

[41] Scott Freeman, Eileen O'Connor, John W Parks, Matthew Cunningham, David Hurley, David Haak, Clarissa Dirks, and Mary Pat Wenderoth. Prescribed active learning increases performance in introductory biology. *CBE—Life Sciences Education*, 6(2):132–139, 2007.

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

[43] 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. Association for Computing Machinery.

[44] 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. Association for Computing Machinery.

[45] 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. Association for Computing Machinery.

[46] 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. Association for Computing Machinery.

[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] 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.

[49] Brian Hanks. Student performance in cs1 with distributed pair programming. *ACM SIGCSE Bulletin*, 37(3):316–320, 2005.

[50] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research*, pages 19–26, 2014.

[51] 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.

[52] Malcolm W Corney, Donna M Teague, and Richard N Thomas. Engaging students in programming. In *Conferences in Research and Practice in Information Technology, Vol. 103. Tony Clear and John Hamer, Eds.*, volume 103, pages 63–72. Australian Computer Society, Inc., 2010.

[53] Stuart Zweben and Betsy Bizot. Cra taulbee survey. computing research news, 31 (5), may 2019, 2018.

[54] Allan Fisher and Jane Margolis. Unlocking the clubhouse: the carnegie mellon experience. *ACM SIGCSE Bulletin*, 34(2):79–83, 2002.

[55] Christine Alvarado and Zachary Dodds. Women in cs: an evaluation of three promising practices. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 57–61, 2010.

[56] Michael Haungs, Christopher Clark, John Clements, and David Janzen. Improving first-year success and retention through interest-based cs0 courses. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 589–594, New York, NY, USA, 2012. ACM.

[57] Elizabeth Bonsignore, Kari Kraus, Amanda Visconti, Derek Hansen, Ann Fraistat, and Allison Druin. Game design for promoting counterfactual thinking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2079–2082, 2012.

[58] Robert Bryant, Richard Weiss, Genevieve Orr, and Kathie Yerion. Using the context of algorithmic art to change attitudes in introductory programming. *Journal of Computing Sciences in Colleges*, 27(1):112–119, 2011.

[59] Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 247–252, 2012.

[60] Mark Guzdial. Education teaching computing to everyone. *Communications of the ACM*, 52(5):31–33, 2009.

[61] Mark Guzdial. Does contextualized computing education help? *ACM Inroads*, 1(4):4–6, 2010.

[62] Susanne Hambrusch, Christoph Hoffmann, John T Korb, Mark Haugan, and Antony L Hosking. A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, 41(1):183–187, 2009.

[63] 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. Association for Computing Machinery.

[64] 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, 2016.

[65] 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, 1998.

[66] 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, 2014.

[67] John Dalbey and Marcia C Linn. The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research*, 1(3):253–274, 1985.

[68] Tammy VanDeGrift, Tamara Caruso, Natalie Hill, and Beth Simon. Experience report: getting novice programmers to think about improving their software development process. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 493–498, 2011.

[69] Lauri Malmi, Judy Sheard, Päivi Kinnunen, and Jane Sinclair. Computing education theories: what are they and how are they used? In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 187–197, 2019.

[70] 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. Association for Computing Machinery.

[71] Paul R Pintrich and Elisabeth V De Groot. Motivational and self-regulated learning components of classroom academic performance. *Journal of educational psychology*, 82(1):33, 1990.

[72] K Patricia Cross and Thomas A Angelo. Classroom assessment techniques. a handbook for faculty. 1988.

[73] Richard M Ryan and James P Connell. Perceived locus of causality and internalization: examining reasons for acting in two domains. *Journal of personality and social psychology*, 57(5):749, 1989.

[74] Duane F Shell, Jenefer Husman, Jeannine E Turner, Deborah M Cliffel, Indira Nath, and Noelle Sweany. The impact of computer supported collaborative learning communities on high school students' knowledge building, strategic learning, and perceptions of the classroom. *Journal of Educational Computing Research*, 33(3):327–349, 2005.

[75] Marlene Schommer. Effects of beliefs about the nature of knowledge on comprehension. *Journal of educational psychology*, 82(3):498, 1990.

[76] Andrew J Elliot and Holly A McGregor. A 2× 2 achievement goal framework. *Journal of personality and social psychology*, 80(3):501, 2001.

[77] Paul R Pintrich et al. A manual for the use of the motivated strategies for learning questionnaire (mslq). 1991.

[78] Leo Leppänen, Juho Leinonen, and Arto Hellas. Pauses and spacing in learning to program. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 41–50, 2016.

[79] Laura Toma and Jan Vahrenhold. Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs-a case study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 1–10, 2018.

[80] Lorin W Anderson, Benjamin Samuel Bloom, et al. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman,, 2001.

[81] Benjamin S Bloom. Learning for mastery. instruction and curriculum. regional education laboratory for the carolinas and virginia, topical papers and reprints, number 1. *Evaluation comment*, 1(2):n2, 1968.

[82] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

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

[84] 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.

[85] 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.

[86] Peter C Brown, Henry L Roediger III, and Mark A McDaniel. *Make it stick*. Harvard University Press, 2014.

[87] Michael Pressley, Mark A McDaniel, James E Turnure, Eileen Wood, and Maheen Ahmad. Generation and precision of elaboration: Effects on intentional and incidental learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(2):291, 1987.

[88] Carole A Bagley and C Candace Chou. Collaboration and the importance for novices in learning java computer programming. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 211–215, 2007.

[89] Michael S Kirkpatrick and Samantha Prins. Using the readiness assurance process and metacognition in an operating systems course. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 183–188, 2015.

[90] Lisa Yan, Annie Hu, and Chris Piech. Pensieve: Feedback on coding process for novices. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 253–259, 2019.

[91] 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.

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

[93] 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.

[94] Nate Kornell and Robert A Bjork. The promise and perils of self-regulated study. *Psychonomic Bulletin & Review*, 14(2):219–224, 2007.

[95] Christopher Watson, Frederick WB Li, and Jamie L Godwin. No tests required: comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 469–474, 2014.

[96] 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.

[97] Richard A Schmidt and Robert A Bjork. New conceptualizations of practice: Common principles in three paradigms suggest new concepts for training. *Psychological science*, 3(4):207–218, 1992.

[98] Nate Kornell. Optimising learning using flashcards: Spacing is more effective than cramming. *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, 23(9):1297–1317, 2009.

[99] 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.

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

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

[102] Regina MF Wong, Michael J Lawson, and John Keeves. The effects of self-explanation training on students' problem solving in high-school mathematics. *Learning and Instruction*, 12(2):233–262, 2002.

[103] 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. Association for Computing Machinery.

[104] 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. Association for Computing Machinery.

[105] 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.

[106] 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.

[107] 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. Association for Computing Machinery.

[108] Viggo Kann and Anna-Karin Högfeldt. Effects of a program integrating course for students of computer science and engineering. In *Proceedings of the 47th ACM technical symposium on computing science education*, pages 510–515, 2016.

[109] Jeffrey A Stone and Elinor M Madigan. Integrating reflective writing in cs/is. *ACM SIGCSE Bulletin*, 39(2):42–45, 2007.

[110] Ben Stephenson, Michelle Craig, Daniel Zingaro, Diane Horton, Danny Heap, and Elaine Huynh. Exam wrappers: Not a silver bullet. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 573–578, 2017.

[111] Y Daniel Liang. *Introduction to Java programming and data structures*. Pearson Education, 2018.

[112] Franklyn Turbak, Constance Royden, Jennifer Stephan, and Jean Herbst. Teaching recursion before loops in CS1. *Journal of Computing in Small Colleges*, 14(May):86–101, 1999.

[113] Seth Poulsen, Carolyn J Anderson, and Matthew West. The relationship between course scheduling and student performance. 2020.

[114] Jerome S Bruner. *The process of education*. Harvard University Press, 2009.

[115] Department of Electrical & Computer Engineering (Unlisted). What is spiral curriculum?, 2021.

[116] Michal Armoni. COMPUTING IN SCHOOLS - Spiral thinking: K-12 computer science education as part of holistic computing education. *ACM Inroads*, 5:31–33, jun 2014.

[117] Joe Feldman. *Grading for equity: What it is, why it matters, and how it can transform schools and classrooms.* Corwin Press, 2018.

[118] Nick Parlante. Codingbat, 2021.

Appendix A

Evaluation Materials

The following appendix shares evaluations for the research methodology that students interacted with.

A.1 Research Disclaimer

Let's start with a survey to get a foundation of where you are all at. We will repeat this survey at the end of the semester to see how your opinions have changed. We are also looking at self-regulation and how that relates to performance.

This semester we are researching differences in teaching pedagogies. Please note, all data will be aggregated (combined) and anonymized - which means at no point are we reporting on individual results. For example, we will say section A had an average grade of X, while section B had an average grade of Y. With that said, we also value your privacy, so you have the option to opt-out.

If you are interested in having your data removed before publication in the comparison of CS 163 pedagogies, please email the following to lionelle@colostate.edu.

"As a possible human subject in the research occurring as part of the CS 163/4 – Fall 2020/Spring 2021, I am opting to not allow my data to be used in the study. I understand that there is no penalty for not contributing my data to the study.

This is only a waiver to opt out any data collected from my participation to be used for publication. This is not a waiver of participation in the course itself, or collection of data for course grade and department use."

Please have in the subject line "OPT-OUT: CS 163/4". I will have the emails go to an email folder, and then confirm after grades are posted of your wish for your data to be removed.

If you have any further questions, feel free to email Albert.Lionelle@colostate.edu.

IRB Approval PROTOCOL NUMBER: 20-10279H

A.2 Exam Question Exemplars

Figure A.1: Sample exam question with color coding. Only certain ones had the color coding, just simply based on who designed the question.

Question 5 4 pts

Giving the following code, match the input with the outcome / returned String. For example, if you have 10,9,8 - you are assuming branchingCheck(10, 9, 8); is the method called and you should match it to the correct return value.

```
public String branchingCheck(int valOne, int valTwo, int valThree) {

    double pi = 3.14;
    String hasPie = "I like pie";
    String morePie = "I need more pie";
    String actualPie = "n";

    if (valOne > valTwo) {
        if (valOne > pi) {
            return hasPie;
        }
    } else if (valTwo >= valThree) {
        return morePie;
    } else {
        if (valThree < 3.14) {
            return actualPie;
        }
        return "The value of pi is: " + pi;
    }
    return "no pie";
}
```

10, 9, 8 [Choose]

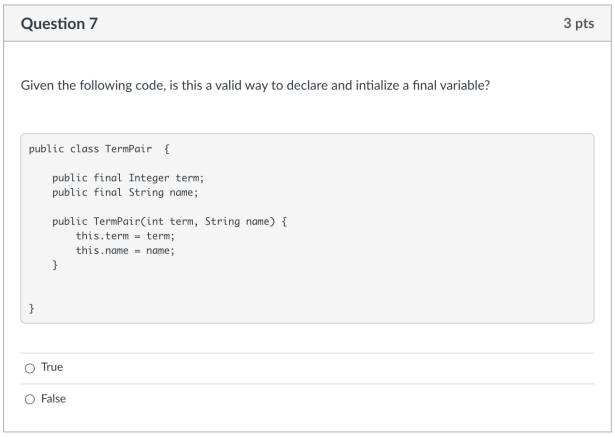
8, 10, 20 [Choose]

0, 0, 0 [Choose]

3, 0, 1 [Choose]

0,1,2 [Choose]

Figure A.2: Sample drop down choices exam question. Often 'noise' answers were added.



Question 7 3 pts

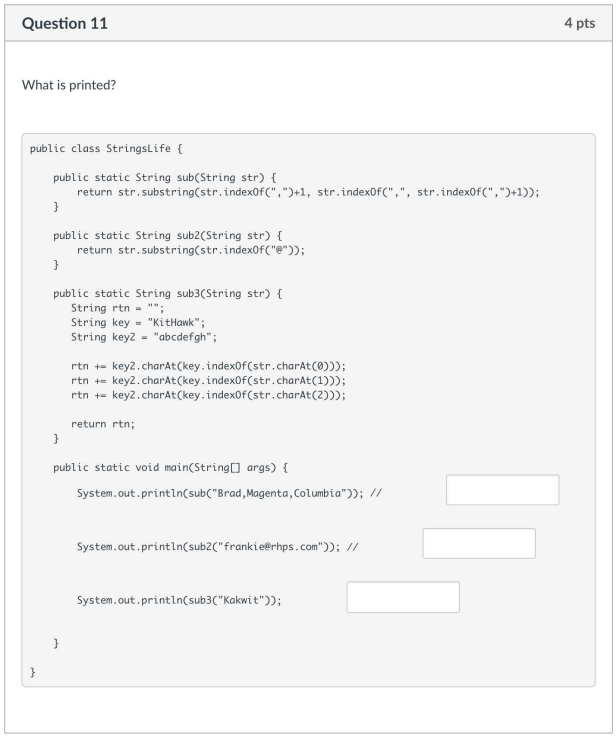
Given the following code, is this a valid way to declare and intialize a final variable?

```
public class TermPair  {

    public final Integer term;
    public final String name;

    public TermPair(int term, String name) {
        this.term = term;
        this.name = name;
    }


}
```

○ True

○ False

Figure A.3: Sample true/false exam question.

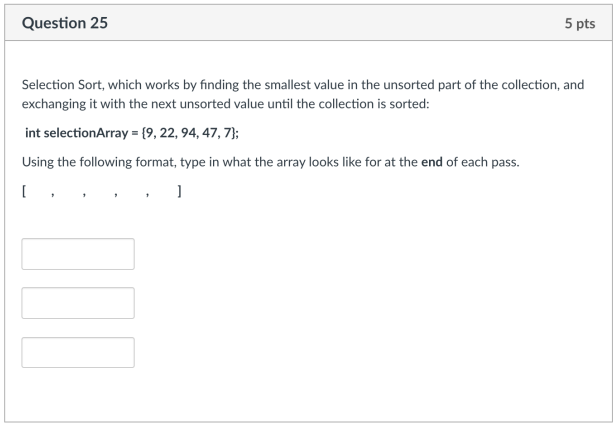


Question 11 4 pts

What is printed?

```
public class StringsLife {

    public static String sub(String str) {
        return str.substring(str.indexOf(",")+1, str.indexOf(",", str.indexOf(",")+1));
    }

    public static String sub2(String str) {
        return str.substring(str.indexOf("@"));
    }

    public static String sub3(String str) {
       String rtn = "";
       String key = "KitHawk";
       String key2 = "abcdefgh";

       rtn += key2.charAt(key.indexOf(str.charAt(0)));
       rtn += key2.charAt(key.indexOf(str.charAt(1)));
       rtn += key2.charAt(key.indexOf(str.charAt(2)));

       return rtn;
    }

    public static void main(String[] args) {

        System.out.println(sub("Brad,Magenta,Columbia")); //


        System.out.println(sub2("frankie@rhps.com")); //


        System.out.println(sub3("Kakwit"));


    }
}
```

Figure A.4: Sample fill in the blank exam question.



Question 25 5 pts

Selection Sort, which works by finding the smallest value in the unsorted part of the collection, and exchanging it with the next unsorted value until the collection is sorted:

int selectionArray = {9, 22, 94, 47, 7};

Using the following format, type in what the array looks like for at the **end** of each pass.

[, , , ,]

Figure A.5: Sample theory question.

A.3 Attitude Survey and MSLQs

The basic survey questions are as follows:

- Based on a scale where 1 is the lowest and 10 is the highest, how would you rate the following statement? **I know that I like Computer Science**
- Based on a scale where 1 is the lowest and 10 is the highest, how would you rate the following statement? **I think Computer Science is interesting**
- Based on a scale where 1 is the lowest and 10 is the highest, how would you rate the following statement? **I feel confident in my ability to learn Computer Science**
- Based on a scale where 1 is the lowest and 10 is the highest, how would you rate the following statement? **I feel confident in my problem-solving ability**
- Do you plan to take more CS courses after this one? (CS 165, etc) true/false answer.
- Have you tried learning a programming language before? You can include HTML and MATLAB as languages for the purposes of this question. If you have tried learning a language before, please list them below separated by a comma. (only answered on pre-course survey)

A.3.1 MSLQ

The questionnaire was developed by Pintrich and Groot, and has been used as a means to measure student belief in their motivational beliefs and self-regulated learning strategies. It is currently used in our field, along with other measures. We listed the questions as they are listed in Pintrich and Groot [71], skipping the questions they do not use for evaluation. Students did not see the skipped questions, but kept them listed for preservation of the questionnaire numbering.

The scale for each question is a 7 point Likert scale(1 = not at all true for me, to 7 = very true of me).

1. I prefer class work that is challenging so I can learn new things.
2. Compared with other students in this class I expect to do well.
3. I am so nervous during a test that I cannot remember facts I have learned.
4. (skipped)
5. It is important for me to learn what is being taught in this class.
6. I like what I am learning in this class.
7. I'm certain I can understand the ideas taught in this course.
8. (skipped)
9. I think I will be able to use what I learn in this class in other classes.
10. I expect to do very well in this class.
11. Compared with others in this class, I think I'm a good student.
12. I often choose paper topics I will learn something from even if they require more work.
13. I am sure I can do an excellent job on the problems and tasks assigned for this class.
14. I have an uneasy, upset feeling when I take a test.
15. I think I will receive a good grade in this class.
16. (skipped)
17. Even when I do poorly on a test I try to learn from my mistakes.
18. I think that what I am learning in this class is useful for me to know.
19. (skipped)
20. My study skills are excellent compared with others in this class.

21. I think that what we are learning in this class is interesting.

22. Compared with other students in this class I think I know a great deal about the subject.

23. I know that I will be able to learn the material for this class.

24. I worry a great deal about tests.

25. Understanding this subject is important to me.

26. (skipped)

27. When I take a test I think about how poorly I am doing.

28. (skipped)

29. (skipped)

30. When I study for a test, I try to put together the information from class and from the book.

31. When I do homework, I try to remember what the teacher said in class so I can answer the questions correctly.

32. I ask myself questions to make sure I know the material I have been studying.

33. It is hard for me to decide what the main ideas are in what I read.

34. When work is hard I either give up or study only the easy parts.

35. When I study I put important ideas into my own words.

36. I always try to understand what the teacher is saying even if it doesn't make sense.

37. (skipped)

38. When I study for a test I try to remember as many facts as I can.

39. When studying, I copy my notes over to help me remember material.

40. I work on practice exercises and answer end of chapter questions even when I don't have to.

41. Even when study materials are dull and uninteresting, I keep working until I finish.

42. When I study for a test I practice saying the important facts over and over to myself.

43. Before I begin studying I think about the things I will need to do to learn.

44. I use what I have learned from old homework assignments and the textbook to do new assignments.

45. I often find that I have been reading for class but don't know what it is all about.

46. I find that when the teacher is talking I think of other things and don't really listen to what is being said

47. When I am studying a topic, I try to make everything fit together.

48. (skipped)

49. (skipped)

50. (skipped)

51. (skipped)

52. When I'm reading I stop once in a while and go over what I have read.

53. When I read material for this class, I say the words over and over to myself to help me remember.

54. I outline the chapters in my book to help me study.

55. I work hard to get a good grade even when I don't like a class.

56. When reading I try to connect the things I am reading about with what I already know.