2019-03-21

Re-designing a Computer Science Course in Higher Education

Nathan Fox
fox016@gmail.com

Follow this and additional works at: https://scholarsarchive.byu.edu/ipt_projects
Learn more about the BYU department of Instructional Psychology and Technology at http://education.byu.edu/ipt

BYU ScholarsArchive Citation
Fox, N. (2019). Re-designing a Computer Science Course in Higher Education. Unpublished masters project manuscript, Department of Instructional Psychology and Technology, Brigham Young University, Provo, Utah. Retrieved from https://scholarsarchive.byu.edu/ipt_projects/15

This Design/Development Project is brought to you for free and open access by the Instructional Psychology and Technology at BYU ScholarsArchive. It has been accepted for inclusion in Instructional Psychology and Technology Graduate Student Projects by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Re-designing a Computer Science Course in Higher Education

Nathan Fox

Design & Development Project Report
Instructional Psychology & Technology, Brigham Young University
1 Purpose

The computer science department at Brigham Young University feels a need to standardize its courses across instructors. The purpose of this project was to standardize CS 235, the second programming class in the computer science major/minor. There were two existing versions that I evaluated to understand which teaching methods, assessments, and technologies used in each course led to better student outcomes and redesigned a new course based on what appeared to be most effective across courses. The CS department wanted the curriculum to (a) promote deeper learning; (b) help students transition to more proactive, self-paced, student-owned learning; (c) be more scalable to meet increased demand; and (d) encourage students to help each other.

2 Background and Justification

At BYU, computer science instructors are given a lot of freedom in how they design and teach courses, which has led to a wide variety of teaching practices, coding projects, and assessments within certain courses. Although the department has some policies to standardize courses (e.g., all instructors should use the same coding projects when teaching a course), there are cases in which these policies were not being followed. Such was the case with CS 235, a sophomore-level CS class about data structures. The department wanted to know which version of the course was doing a better job of helping students to achieve learning outcomes, and then they wanted me to redesign and teach a section of the course. They did not want me to redesign a third version of CS 235 from scratch: They wanted me to evaluate what was already working well in the two existing versions and then build on top of it.

I began this project by observing and evaluating the different versions of CS 235. My evaluation of the two courses revealed that: (a) the traditional version of the course provides a lot of good resources, but lacks pedagogical strategy; and (b) the flipped version of the course utilizes good pedagogical strategy, but lacks in helpful resources. Both courses had effective coding labs, but they both needed improved written exams. I leveraged the effective resources from the traditional version and the pedagogical strategy of the flipped version to design a course that promotes deeper learning.

3 Product Description

The delivered product is a new CS 235 course in Canvas. It contains new interactive reading assignments to replace the textbook, new exams to better measure higher-level learning outcomes, live coding activities to do in class to model expert performance, a new coding lab, and other resources.

3.1 New Reading Assignments

The new reading assignments replace the textbook and keep students accountable for learning. This accountability for reading makes it easier for the instructor to use the flipped classroom paradigm, allowing the instructor to use lecture time to fill in holes in understanding and dive deeper into important content. The reading assignments contain text blocks separated by
questions, giving students the opportunity to apply what they learn as they read (cf. Fig. 3.1). Students see the results as soon as they submit, and they can attempt reading quizzes multiple times to deepen their understanding and answer all the questions correctly.

**Infix and Postfix**

When we see basic mathematical expressions in the United States, they are generally written in infix notation. Common operations such as + * / are written as:

- **left operand** + operator + **right operand**

such as "7 * 2" or "6 / 3".

With infix notation, the order of operations is very important. For example, 2 + 6 * 3 should be equal to 2 + (6 * 3) because the order of operations gives precedence to parentheses first, then * second, and + third.

**Postfix notation**, also known as reverse Polish notation, expresses mathematical expressions using a different pattern. Basic expressions such as + * / are written as:

- **left operand** + operator + **right operand**

With postfix notation, there are no parentheses and there is no order of operations. Whenever an operator is seen in the expression, the calculation is performed on the last two operands and the result of the calculation fills the position that the two operands and the operator were in. For example:

5 15 6 7 + -

Going from left to right, the first operator we see is +. So we replace "6 7 +" with the result of "6 + 7":

5 15 13 -

Now we see a - operator. So we replace "15 13 -" with the result of "15 - 13":

5 2 *

Now we see a * operator. So we replace "5 2 *" with the result of "5 * 2":

10

And that is our answer. So the postfix expression 5 15 6 7 + - is evaluated to 10. It is equivalent to the infix expression 5 * (15 - (6 + 7)).

https://en.wikipedia.org/wiki/Reverse_Polish_notation#Example shows the evaluation of the infix expression (15 + (7 - (11 + 10) * 3)) / (2 + (1 * 18)) and the equivalent postfix expression 15 7 11 - - + 3 2 1 1 + + -

**Question 4**

Evaluate this postfix expression by hand.

5 6 4 * 7 2 + +

Enter the numerical solution in the box below.

*Figure 3.1 An example of text content followed by an application question from the Stacks and Expressions reading assignment.*

**3.2 New Exams**

To design the new midterm and the new final exam, I used a table of specifications to improve the balance of content and to target more higher-level learning outcomes. I used primarily objectively-scored items, but introduced a couple performance items to measure student ability...
to write short C++ solutions to unfamiliar problems (cf. Fig. 3.2). For the midterm exam, I raised the percentage of application-level questions from 5% to over 50%.

### 3.3 Live Coding Activities

Rubin (2013) performed a study in which students who took a course with live coding performed much better on coding labs than students who took the same course with static code examples. Thus, I introduced new live coding activities into the course that instructors can use to model expert performance during lecture time. Instructors explain the problem to be solved and use principles of cognitive apprenticeship to talk through the problem-solving process. Students follow along on their own computers as the instructor writes and talks through code (cf. Fig. 3.3). Live coding problems are designed to share enough in common with coding labs that they help students approach the labs while being different enough that the labs are still challenging.

---

**Figure 3.2** An example of a performance assessment item to measure student ability to code C++ solutions to unfamiliar problems.

#### Complete the following recursive function so that it works as described in the comments:

```c++
/*
 * Factorial is a recursive function that returns
 * the factorial of a number. Mathematically, 5
 * factorial is written as 5! and it is equal to
 * 5 x 4 x 3 x 2 x 1. Essentially, you multiply all
 * positive integers less than or equal to the input
 * together to calculate a factorial. You can also
 * think of 5! as being equal to 5 x 4 x 3 x 2 x 1. Return 1 if
 * the input is less than or equal to 0. Return 1 if
 * the input is 1.
 * This function cannot use loops and cannot use math
 * libraries. You must figure out a recursive solution.
 * Arguments int n - number to calculate factorial of
 * Return n!
 * Return 0 if n is less than or equal to 0
 */
int factorial(int n)
{
    ...
}
```

This question is worth 4 points. 2 points for base case(s) and 2 points for recursive call(s). Write the C++ code to implement the recursive factorial function.
Figure 3.3 An example of C++ code written during a live coding activity to model how to approach designing and coding recursive backtracking functions.

### 3.4 New Coding Lab

Per the request of the CS department, I made minimal changes to most coding labs. However, I made significant changes to the final coding lab, lab 9. Instead of just having students implement a hashmap, lab 9 tests their hashmap to solve an interesting real-world problem of counting the number of occurrences of each word in a text document. In addition to that, lab 9 has students use heapsort, which is covered in reading assignments but is not typically included in labs. Students download the initial code from GitHub and follow the instructions in the readme file to complete the lab (cf. Fig 3.4).
4 Design Process

This project is an atypical design project for a couple of reasons. Instead of designing a course from the ground up, I was given two versions of a course to evaluate and build off of, so the initial prototype was essentially handed to me along with some ideas of how I could improve upon it. Also, I was able to play the role of designer, subject matter expert, and instructor, allowing me to work efficiently on my own with only a little needed support from other CS 235 instructors and TAs. Furthermore, due to time constraints I had to design the course as I was teaching it, rushing some of the design but also giving me the opportunity to collect feedback I could use to improve my design of later elements of the course.

4.1 Phases

Although this design project is atypical in those regards, the design process still included the typical phases of analysis, initial evaluation, planning, and prototyping, with an added design in flight phase where there was an overlap in design, development, and implementation.

4.1.1 Analysis

In the analysis phase, I worked with Dr. Seppi, Dr. Clement, and Dr. Crandall from the CS department to gather information. I analyzed grade data from previous semesters, observed CS 235 courses, interviewed students, and talked with instructors. Dr. Clement told me that he thought the course’s primary need was more accountability for reading assignments in order to make the flipped classroom approach more effective and to make the Canvas course a viable online or blended course, so I consulted a few computer science textbooks to get an idea of how
to design reading assignments and quizzes. I mapped out the content covered in the course to connect related topics and linearize the content. From my own experience in the CS program and my experiences as a professional software engineer, I made a decision tree to model expert behavior for how to choose the best data structure for any given problem. I also did my literature review, which helped me to identify pedagogical practices to include in my design.

4.1.2 Initial Evaluation

Consulting precedent was a large and necessary part of this project because the CS department asked me to evaluate the two current versions of the course and choose one of them as a base model rather than start the design from scratch. I created a list of evaluation questions and collected data through personal observation, student interviews, historic survey data provided by the CS department, and grade data from Fall 2017 and Winter 2018 provided by the CS department. I looked for how teaching techniques aligned with recommendations in literature, how assessments aligned with course learning outcomes, and how technology was used to enhance the learning experience. In the end, I decided to use the flipped version of the course (as taught by Dr. Clement) as my base model while integrating some learning resources from the lecture-based version of the course (as taught by Dr. Roper).

4.1.3 Planning

From the data collected in the evaluation and analysis phases, I realized that my primary deliverables to enhance the course should be (a) new interactive reading assignments, (b) new exams to better measure higher-level learning outcomes, (c) new live coding activities to do in class to model expert performance, and (d) a web-based think-pair-share application. I created a course schedule document to map out the semester and to figure out what reading assignments and live coding activities I needed to create. In doing so I identified 25 reading assignments that I needed to create. To make sure I was ready to teach to the labs and to the exams, I coded my own solution to each of Dr. Clement’s labs and took his exams.

4.1.4 Prototypes

I started designing prototypes in late July, realizing that I would need at least 4-5 weeks of the course fully designed and developed by early September when class would start. Out of necessity, I started at the beginning and quickly designed and developed the reading assignments and live coding exercises for the first few weeks of class. I developed a web-based think-pair-share application and designed a few think-pair-share questions that I could use in class over the first two weeks. I started designing the midterm so that I could have a better idea of how to design the reading assignments to prepare students for the midterm.

4.1.5 Design in Flight

Once class started in early September, my only option was to design the rest of the course as I taught it. I asked my students for feedback regularly so that I could better design the reading assignments and live coding activities that I had not yet developed. I tried to stay at least three weeks ahead on reading assignments and one week ahead on live coding activities and think-pair-share questions. I met weekly with Dr. Crandall, Dr. Roper, and the CS 235 TAs to get a feel
for what was going on in the other sections of CS 235 and to talk through what I was doing with my section.

5 Design Evolution

Due to the nature of the project, there weren’t any major iterations per se. Primarily there were a lot of small changes as I received and responded to feedback from students, other instructors, and TAs. So in this section, I break the project down into three phases: before, during, and after implementation. I cover how I initially designed the course to be, how I changed it as I taught it, and the modifications I made to the course after teaching it.

5.1 Before Implementation

Early in my project I knew that I wanted to use live coding activities in class. I started my project with my literature review, and that pedagogical concept stood out the most to me as I read through literature reviews and studies about teaching computer science. As I observed Dr. Clement (the CS professor who taught the flipped version of the course in Winter 2018) teach and looked through his course materials in Canvas, I saw that he had designed some live coding activities and I wanted to build upon them and design more of them. In my own experience teaching IP&T 660 in Winter 2018, I learned firsthand how effective live coding was compared to just talking through carefully prepared static code in a lecture.

Another pedagogical tool that really stood out to me from my literature review was think-pair-share. Several research articles and literature reviews that I read through talked about the benefits of using think-pair-share exercises with computer science students. I researched a few existing think-pair-share tools, and after realizing that none were quite right for what I wanted to do, I designed and built my own think-pair-share web application. As I started doing design work, I wrote think-pair-share questions to use during the first several lectures.

When I began doing the design work prior to implementation, I met with Dr. Clement and asked him about what he thought would be most helpful for this course. He said that he wanted more interactive reading assignments to keep students accountable for reading. I looked at a few different CS textbooks—the ZyBooks online textbook used in CS 142, MIT’s textbook for teaching data structures, and my favorite textbook from my CS undergrad—and decided to design reading assignments with blocks of text separated by application questions to promote spaced learning.

Initially I had not planned on changing the exams or any of the coding labs, just the reading assignments and how class time was used. However, I was taking the assessments class as I was doing the front-end analysis and evaluation of the two versions of CS 235, and one of the assignments from the assessments class was to reverse engineer a table of specifications from an assessment. So, I decided to reverse engineer the CS 235 midterm from the flipped version of the course so that I could also use it as part of my front-end evaluation. In doing so, I realized that the exams Dr. Clement was using were unbalanced in terms of content coverage and were primarily measuring low-level learning outcomes such as recall or recognition. After seeing and critiquing the table of specifications, I realized that I should rewrite the midterm and the final exam according to a table of specifications that balanced the content and focused on assessing learning outcomes regarding understanding and application.
5.2 During Implementation

Doing live coding activities for an audience of 60 ended up being different than I had anticipated. It was difficult to know who was keeping up and who was completely lost, and it was more difficult than anticipated to get through everything I wanted to in the 50-minute class time. I spent more time doing live coding in class than I had initially anticipated, which left less time in class for students to work together on their coding labs. From the survey that I assigned to students after the first four weeks of class, I learned that the live coding is something that people liked, but many thought that I as the instructor went through the code too fast. The main suggestions were to slow down, get more discussion and participation, talk more about the labs, and give more time to work on labs. I started designing shorter live coding activities that involved things like designing single functions instead of writing an entire small program, and after that survey results were more positive.

Over the first few weeks of the course, I heard concerns from multiple students about using the Cloud9 environment. I had underestimated how difficult it would be for students to pick up enough Bash to navigate the Linux environment of Cloud9. I assumed that they would learn just from watching me do it in class, but many anxious students reached out to me about how Bash was a nontrivial barrier for them. In response, I developed a Linux/Bash training module in Canvas and devoted a lecture hour to go over it in class. I shared the training module with Dr. Crandall, another CS 235 instructor who was using Cloud9 and facing similar difficulties, and he thought it was very helpful.

From the first survey I assigned to get feedback from students, I learned that many were still struggling with another unexpected obstacle: debugging in the Cloud9 environment. In response I designed a debugging tutorial in Canvas and GitHub to help students with debugging in C++, and I devoted a lecture to go through the tutorial as a class.

I was surprised by how much time the think-pair-share questions took up and how little of an impact they seemed to have had. 50 minutes is not a long time, and taking 5 minutes to go over a think-pair-share question ate up too much of that time. In many cases, the vast majority of the students got the question right the first time, so I just skipped the pair-share and launched into an open discussion about the question. In other cases where I did ask the students to discuss their answers with a neighbor, some students (often the ones who did need help to get the correct answer) did not participate. After the first 6 weeks I stopped doing think-pair-share activities and just did class discussions instead.

My initial plan was to use all of Dr. Clement’s labs, including his lab 9 as the final lab. But unlike his other labs, lab 9 didn’t have an autograder. Instead, he had students grade each other’s in the last week of class. I didn’t want to take up class time to do that, so I designed lab 9 to be more like the other labs with an autograder. I still required students to create a hashmap implementation, but I gave them more of a framework and provided some guidance. As part of the autograder, I wrote a solution to a real-world problem that uses hashmaps and I plugged the student’s hashmap implementation into my solution to test the student’s hashmap.
5.3 After Implementation

During the first few weeks of teaching, I identified two surprise pain points for students: learning Bash/Linux and debugging in Cloud9. I developed resources to help students learn these prerequisite skills, and after I finished teaching I modified the course schedule design to cover these topics earlier in the semester.

Another change I made after teaching is making lab 8 (AVL) extra credit and assigning lab 9 a week earlier to give students more time to get it done. I really like lab 9, but I didn’t like assigning lab 9 out so close to the end of the semester and having it due the last day of class. This change gives students more time to focus on lab 9. I think that the learning outcomes of lab 9—such as understanding the nature and purpose of hash functions—are more important throughout the rest of the CS program than the learning outcomes of the AVL lab, especially considering how similar the AVL lab is to the BST lab. The lecture-based version of the class already uses the AVL lab as an extra credit lab, so there is established precedent for using it as an extra credit lab.

The reading assignments were well-received in general, but there was one notable exception. Several students commented that the AVL Add/Remove reading assignment was more confusing than helpful. I rewrote it completely (compare the original with the new version). Figures 5.1 and 5.2 show examples of before versus after the rewrite.

AVL Tree Rotations

 Lemma 7.2 In a AVLTree that stores a set $S$ of $n$ keys, the following statements hold:

1. For any $x \in S$, the expected length of the search path for $x$ is $H_{\text{AVL}}(x) = O(1)$.
2. For any $x \in S$, the expected length of the search path for $x$ is $H_{\text{AVL}}(x) = H_{\text{BST}}(x)$.

Here, $\text{r}()$ denotes the rank of $x$ in the set $S \cup \{x\}$.

Again, we emphasize that the expectation applies to random choices of the priorities for each node. It does not require any assumptions about the randomness in the keys.

Because of this, an AVLTree can implement the find($x$) operation efficiently. However, the real benefit of an AVLTree is that it can support the add($x$) and delete($x$) operations efficiently. To do this, it needs to perform rotations in order to maintain the heap property. A rotation in a binary search tree is a local modification that takes a parent $u$ of a node $w$ and makes $w$ the parent of $u$, while preserving the binary search tree property. Rotations come in two flavors: left or right depending on whether $w$ is a right or left child of $u$, respectively.

![Figure 5.1. A screenshot of the beginning of the original AVL Add/Remove reading assignment.](image-url)
11

Figure 5.2. A screenshot of part of the new AVL Add/Remove reading assignment.

6 Product Implementation

The pilot of this course started at the beginning of the Fall 2018 semester, before all of the design and development work had been completed. Thus, the experiences that I had in the beginning of the semester as the initial instructor informed my design and development work for the later parts of the course. To pilot the course, I worked with the CS department to teach it as a university course under a section labeled as an experimental section with CS 142 as a listed prerequisite. Students need to be comfortable with C++ programming basics before starting the course.

The primary resources needed for this class are (1) the Canvas course; (2) Cloud9 access for instructors, TAs, and students; (3) laptops for students; and (4) Campuswire accounts for instructors, TAs, and students. The Canvas course is essential as it contains all the reading assignments, the exams, and the autograders for the labs. Cloud9 access is needed for students to write coding solutions and for TAs and instructors to access that code, either for grading or for remote help. Students need to bring a laptop to class to follow along with live coding exercises and to work on labs. Campuswire provides a way for students to get answers to their questions outside of class time. Campuswire gives instructors, TAs, and other students access to answer questions and to upvote others' answers. Due to the Canvas course, Cloud9, and Campuswire...
being online materials, students need internet access to do the assignments, labs, and exams. None of these resources require high bandwidth, so low bandwidth is acceptable but high bandwidth is preferred.

For class time, the required physical setup is pretty basic. The instructor needs a classroom with a projector (or some other way of screen-sharing), a computer with a C++ programming environment, and preferably an internet connection (low bandwidth acceptable).

7 Evaluation

7.1 Criteria

The primary stakeholder for this project is the CS department. Specifically, I report to Dr. Crandall who takes the results to a committee of CS professors who oversee CS 235. The CS department’s goals for this course are to:

1. promote deeper learning;
2. help students transition to more proactive, self-paced, student-owned learning;
3. increase scalability to meet increased demand; and
4. encourage students to help each other (or to better help themselves)

The students who take CS 235 are paying for an education, so they are secondary stakeholders. I collected survey data from them throughout the semester and reported the results of the survey data to the students.

My primary method of collecting student feedback was via surveys that were graded as small assignments. In addition to these surveys, I collected data from student ratings, assessment results, final grades, my experiences as the instructor, and pedagogical concepts from my literature review to evaluate the overall course design.

7.2 Procedures

The data from the monthly surveys I assigned to students were used primarily for formative evaluation, as I was able to quickly analyze feedback and implement changes. Surveys contained quantitative Likert-scale items related to the difficulty of labs, difficulty of reading assignments, frequency of class attendance, frequency of TA lab attendance, usefulness of time in class, and helpfulness of course materials. Surveys also contained qualitative open-ended questions asking what students like about how class time is spent, what suggestions students have to improve how class time is spent, and general suggestions to improve the course. Surveys were due at the end of each month, and students were given a 3-5 day window to take the survey. Once the survey window was closed, I reviewed the quantitative data and I grouped qualitative responses into discrete groups to get a quantitative measure. I then shared the aggregated results with the students and summarized what I planned to do in response.

My experiences as the instructor and pedagogical concepts from literature were also used for formative evaluation. I reviewed the pedagogical concepts from my literature review as I taught the course to make sure that I was consistently using effective teaching techniques in class. My experiences as the instructor—in teaching, grading, communicating with students, and answering
questions—helped me to continuously evaluate myself and make necessary changes throughout the implementation.

The final survey provided data for both formative and summative evaluation, because it was close enough to the end of the semester that many students wrote about their experience in the class as a whole. Student ratings, assessment results, and final grades were used for summative evaluation.

7.3 Evidence

In this section, I summarize evaluation evidence in terms of the four goals of the computer science and student responses to the course. Full evaluation data can be found in the appendix.

7.3.1 Promote Deeper Learning

It is difficult to measure deeper learning, and I recognize that my methods for doing so are not perfect. My best evidence for deeper learning includes a comparison of the table of specifications for the exams, assessment results, and feedback from students.

The appendix includes the exam specifications for the midterms for my new course and the flipped version of the course as taught by Dr. Clement in Winter 2018. The new midterm has a drastic increase in application-level assessment items from 5.1% to 56.7%. By defining deeper learning in terms of Bloom’s taxonomy, it is clear that the new midterm is better designed to measure deeper learning. The distribution of scores for the new midterm—within a mean of 92%, a low of 72%, and a standard deviation of 3.93—show that students performed well on the new midterm designed to measure deeper learning. Thus, the table of specifications for the midterm combined with the assessment results of the midterm provide evidence of deeper learning.

Deeper learning is also evidenced by some of the comments provided by students through surveys and student ratings:

“IT has been a really good class and I actually feel like I am starting to understand Computer Science for the first time ever.”

“There’s not much I would change about the class. It’s been my favorite CS class so far. I think the structure is great. The reading quizzes are super helpful and make for easy learning. The labs are challenging but they really make me learn the material. I give this class an A+!”

“Nathan was the best at teaching difficult concepts simply. The reading quizzes were awesome.”

“Whenever we discussed difficult topics, he would repeatedly touch on aspects that were difficult to do or understand and would often have us tell him what he should explain.”

“Really progressive style of teaching. Felt less like a lecture on a set curriculum and more like actual preparation for future employment.”

7.3.2 Promote Student-owned Learning
Some of the best evidence for promoting student-owned learning is found in the course design. Students never had to attend class because they were provided with a course schedule and all necessary materials online, giving them the freedom own their learning. Several students rarely attended class and still performed well, providing evidence of proactive learning. There were also a few students who completed reading assignments and labs days or even weeks in advance because the entire course (except for the exams, which were restricted by date range) was available to them, providing evidence of self-paced learning.

Student-owned learning is also evidenced by some of the comments provided by students through surveys and student ratings:

“I loved the organisation of this course. It was really wonderfully done. He made it easy to access everything we needed online and said just to come to class as often as we needed. I loved the freedom! Please do more classes like this!”

“He trusted us and expected us to be honest without putting a ton of restrictions on how we learn.”

“Nathan gives all the help he can give, but still places the responsibility to learn on the students.”

“I personally learned to be better about managing my time and putting the requisite effort from this class.”

7.3.3 Increase Scalability to Meet Increased Demand

I define course scalability as the ability to add students to the course without overburdening the instructors and TAs. In the history of CS 235, most of the TAs’ and instructors’ time outside of class has been devoted to helping students with labs, grading/verifying labs, and grading exams.

Labs are autograded. The autograder checks that the output matches the expected output, but the process still requires TAs or instructors to look over code to verify that the code was written in the expected way. To save the TAs time, I and the other CS 235 instructors decided to have the TAs verify two labs per student (roughly 25%) over the course of the semester. For my course, I verified an additional two labs per student. The verification process does take time, but it is simplified by using Cloud9 to be able to quickly see and run students’ code from the workspace URL that they submit in Canvas. Students also have access to the autograder and test scripts, so they can get immediate, detailed feedback on their labs without having to visit a TA or instructor for lab passoff or troubleshooting. Many of my students reported never going to the TA lab all semester, because there is no requirement to pass off a lab to a TA in person.

I was the only instructor in Fall 2018 to use Canvas to deliver exams. For each exam, all but one item is objectively scored. This provides students with immediate feedback and saves on grading time. There is only one performance item per exam, and it is easy as the instructor to go through the Canvas grader and to grade the single performance item for each student. It took me about one minute per student, or one hour total. By comparison, the other instructors’ exams required several hours of TA time for grading.
Unfortunately, I do not have a lot of direct quantitative evidence for scalability. There are some records of how often students went to the TA lab for help, but the records are incomplete and do not include how much TA time students used per visit.

7.3.4 Encourage Students to Help Each Other

Students were welcome to work on labs and reading assignments in groups, but never required to. Cloud9 allows multiple students to connect to a single workspace and code collaboratively, which several groups of students took advantage of. I was surprised, however, to find that only 20% of students worked with other classmates on the labs.

Throughout the semester I encouraged the use of Campuswire, where students could ask questions or answer others’ questions. Roughly half of the students took advantage of the ability to ask questions, but I was surprised by how rarely students answered each other’s questions. In almost every case, I was the only one to provide answers to questions. At the start of the semester, Campuswire did not have email notification options, so students had no way of knowing that others were asking questions unless they actively checked Campuswire. Near the middle of the semester Campuswire rolled out an email notification feature that I encouraged students to use, but there were still only rare cases where students answered each other’s questions.

7.4 Outcomes

From the evaluation data, I conclude that this version of CS 235 provides better tools for assessing higher-level learning outcomes, helps students to own their own learning, and is more scalable. Furthermore, it is a viable option for creating an online/blended version of CS 235 should the CS department decide to go in that direction.

Although there is evidence that the design encourages students to help each other, there is evidence that only few chose to seek or provide help. There are some students who clearly needed help but never reached out to the instructor, TAs, or other students. One possible solution to improve this going forward is to organize students into study groups. Allow students to self-organize if they choose, then organize the rest into groups of 3-5 students. They should not be required to collaborate on labs, but should be instructed to sit with their groups in class and talk through their questions together when given time to work on labs in class.

I would also advise the CS department to evaluate and rework some of the coding labs. The coding labs were out of the scope of this project due to limited time and the wishes of the CS department. However, it would be worthwhile to take a critical look at the labs, develop some alternatives, do some user testing, and determine which labs best measure what they are designed to measure. Labs should directly relate to learning outcomes with as little overhead as possible, be interesting to the students, and help students to prepare for their careers and future academic work.
8 Reflection and Critique

I learned several important lessons in designing this course that can be applied to design more broadly. Namely, develop empathy for the learners, conduct a literature review, and don’t assume that learners all come with the necessary prerequisite knowledge. I address each of these briefly in this section.

8.1 Develop Learner Empathy

One of the most important lessons that I learned through designing, developing, and teaching this class is the necessity of developing empathy for the learners. I think the biggest reason why my class was rated well by students is that they felt like they had an opportunity to be heard, their feedback was addressed, they understood how the material covered in the course could help them achieve their personal goals, and they felt like they had the support and the resources necessary to succeed. The learner analysis is critical to developing that kind of empathy. It is easy to collect and report quantitative aggregate data regarding a set of learners, but that kind of impersonal analysis does not help the designer to develop empathy. True empathy is developed through interpersonal connection, through story. Knowing learners’ stories—their background, goals, motivations, fears, and setbacks—is critical for designers to create a product that truly connects with the learners and meets their needs. I learned so much more about the learners from the interviews and the qualitative data that I gathered before, during, and after the course than from the quantitative data. The qualitative data I collected is really what helped me to improve the design and improve my teaching to better meet students’ needs.

Encouraging students to help each other was harder than I expected it to be. Because of my own experience as a CS undergrad with social anxiety, I wanted to give students resources to facilitate working together without forcing students to work together. However, some of the students who could have most benefited from working with other students were the students with social anxiety—those who needed help but were too anxious to seek it even when given the resources to do so. This raises the question of how to help encourage communities of learning in a population of adult learners that is prone to social anxiety while respecting their agency. If I were given more opportunities to teach a course like this, I would continue to try new ways of helping students who need help to overcome social anxiety and rely on their peers.

This course has a learner-centric design, which I think is one of its major strengths. It allows students to go at their own pace, work independently or collaboratively, and to voice ideas for change. The reading assignments with spaced learning activities and the live coding exercises in class help students to see the practical application and to connect practice with theory. If I had more time and resources, I would polish the new reading assignments to make them more consistent. I would also like to redesign the coding labs to make them more appealing to learners.

8.2 Review the Literature

Another thing I learned is the importance of the literature review. If someone beginning a project like this asked me for advice, I would tell them to rely heavily on pedagogical tools mentioned in the literature. From my literature review I learned about the value of live coding over static code
review, and it helped me improve my teaching in Winter 2018 and my design for this project in Fall 2018.

The literature highly recommends using live coding as a pedagogical technique, but the literature fails to capture the mental and emotional fortitude that an instructor is required to have to implement live coding in the classroom. Thus, I would add a very important consideration to basing one’s pedagogy on techniques found in the literature. Specifically, designers should try out these practices themselves if possible, to get a better understanding of practical considerations for implementing specific pedagogies. Even though I cognitively understood that it was good for me to make mistakes so that students could observe my debugging process, it was awkward and embarrassing, especially at first. I became easily flustered, which made it harder for me to think clearly while debugging. I worried that students would question my competency and judge me as an unfit instructor. There were times I had to take a few deep breaths and remind myself that making mistakes in front of a live audience is part of the reason why live coding exercises are so effective. In some cases I got to the end of the lecture period without finding or fixing a bug, and I had to sheepishly announce that for the next lecture we would pick up right where we left off. It became easier for me as I got used to it, but I think it is important for instructors who consider using live coding exercises to be aware of the emotional effects that can come with making mistakes while in the spotlight.

8.3 Don’t Assume Learners Come with the Necessary Prerequisite Knowledge and Skills

Part of having a learner-centric design includes understanding what prior knowledge and skills the learners have instead of making assumptions about their aptitude regarding prerequisite skills. If a significant percentage of learners are not comfortable with prerequisite skills, the instructor should adapt the design of the course to review those skills to help provide learners with the necessary scaffolding to succeed. To assess prior knowledge and skills, I asked students to self-report their comfort level with 18 different skills, 12 of which were learning outcomes from the prerequisite course. I presented students with the following scale:

1 - I don’t know what this is
2 - I don’t know much about this
3 - I know a fair amount about this
4 - I could use this today if I brushed up on in for a bit
5 - I could use this effectively right now

I was surprised to find that there were some prerequisite skills that up to 70% of the learners self-reported a comfort level of 3 or below (see section 9.14.2). Using this scale to have students self-report their prerequisite skills helped me as an instructor and designer to modify the course to review prerequisite skills that learners were less comfortable with, thereby helping more learners to succeed.
9 Appendix

9.1 Guide to the Appendix

Actual Product

This section contains links to my deliverables, including the course materials in Canvas.

Product Walkthrough

This section contains a link to a short product walkthrough video that highlights some of the features of the course.

Consulting Precedent

This section contains my front-end evaluation of the two existing versions of CS 235. It includes my evaluation questions, data collection plan, results, and conclusions. The evaluation results are divided into three sections: teaching techniques, assessments, and technology. In the conclusion I share what from the existing classes I intend to use and what needs to be further developed.

Consulting Products

This section explores various text resources used to teach content related to CS 235. My primary contribution to the course is text resources and small assignments tied to those resources, so I explore what makes good computer science text resources.

Learner Analysis

This section contains grade data and information data that help me to better understand and empathize with CS 235 students. I explore possible variables that might have affected grades in the past in order to identify those who are most likely to struggle. I also discuss students’ and instructors’ opinions of various elements of the course.

Environmental Analysis

This section explores the possible tools and technologies I can use to teach. It also summarizes my meetings with the computer science department, listing constraints and possible goals that I can choose to focus on.

Content and Task Analysis

This section contains a concept map that visualizes the relationships between topics covered in CS 235. It also contains a decision tree modelling expert behavior for an important learning outcome.

Annotated Bibliography
This section contains a list of resources that I have used to explore domain knowledge, effective learning theories, and design approaches. Each resource includes a brief description of how I used it.

**Budget and Timeline**

This section contains a Gantt chart to visualize the lifespan of the project.

**Design Specifications**

This section includes a summary of deliverables, a course timeline, and design concepts for classroom activities and reading assignments.

**Design Representations/Prototypes**

This section includes prototypes for classroom activities and reading assignments.

**Intellectual Property Ownership**

This section includes details about the intellectual property ownership of the course materials that I designed and developed.

**Assessment Reports and Instruments**

This section includes descriptions of each coding lab, a sample lab rubric, and aggregate lab score data. It also includes exam content guides and tables of specification along with aggregate exam score data. Cumulative reading assignment scores and final grade distribution are also included.

**Implementation Instruments**

This section includes a list of required resources for instructors, students and TAs. It also summarizes some of the training materials that were designed to help students learn to use required resources. It also includes information about the students who took the pilot section, including demographic data, prior coding experience, and comfort levels with various coding concepts and technologies.

**Evaluation Instruments**

This section includes a summary of the tools I used to gather evaluation data along with some of the data that I gathered. Tools include surveys, assessment data, pedagogical concepts from literature, and instructor experiences.

**9.2 Actual Product**

A copy of the course can be accessed in Canvas: [https://byui.instructure.com/courses/3551](https://byui.instructure.com/courses/3551). The Canvas course contains the syllabus with the updated learning outcomes, the reading assignments I created or modified, the exams I created (the midterm and the final), the coding lab I created (Lab 9), and other learning resources.
I stored code resources in GitHub: https://github.com/BYU-CS235-F18. These code resources include lab requirements, initial code for labs, test drivers, in-class coding activities, and reference code.

The course calendar can be viewed in Google Sheets: https://docs.google.com/spreadsheets/d/1Sep1LfS2Jh9Lq4JfoKrLwZ3YfGqi4_N0WeQd53khKe8. A more detailed version of lesson notes for each day can be found on GitHub: https://raw.githubusercontent.com/BYU-CS235-F18/Week1/master/cs235notes.txt

9.3 Product Walkthrough

My product walkthrough can be viewed here: https://youtu.be/5X5FdnNs49A

9.4 Consulting Precedent

Consulting precedent was a large and necessary part of this project because the computer science (CS) department asked me to evaluate the two current versions of the class and choose one of them as a base model rather than starting from scratch. The CS department has a policy that coding labs should be the same (or at least very similar) across instructors so that all students are graded on the same work. They already had two very different versions of CS 235; they did not need a third. What they wanted was to know which one was doing a better job of helping students to achieve learning outcomes and then have it improved. They wanted me to make few if any changes to the labs, content model, and learning outcomes and to instead focus my attention on making changes to reading material, micro-assessments, exams, and the use of class time.

Due to the limited scope of a Master’s project, I was unable to conduct full-blown formal evaluations of the current versions. However, I wrote evaluation questions and a data collection plan to aid me in my informal evaluation. This section includes my evaluation questions, data collection plan, and evaluation results for the two existing courses.

9.4.1 Evaluation Questions

1. Teaching techniques in class (to promote deeper learning)
   a. What teaching techniques are used in each version of the course?
   b. How well do teaching techniques align with recommendations from modern literature?
   c. Are students encouraged to help each other or work alone?

2. Assessments
   a. Labs, quizzes, and exams
      i. How do the assessments differ in each version of the course?
      ii. How well do the assessments align with CS 235 learning outcomes?
      iii. How well do the assessments align with practical real-life scenarios?
      iv. How well do the assessments prepare students for related problems in industry?
      v. Are students practicing as they learn?
vi. Are students being held accountable for reading and practicing skills as they learn on their own?

vii. Do learning activities reinforce and build upon one another, allowing students to implement successive feedback and demonstrate progression?

b. Grades
i. How do final grades compare in each version of the course?

3. Technology
a. What are the pros and cons of the technologies being used to deliver content to the students for use outside of the classroom?
b. What are the pros and cons of the technologies being used by students to write and debug code?
c. What are the pros and cons of the technologies being used by students to complete and turn in assignments?

9.4.2 Evaluation Data Collection Plan

1. Observations
   a. Observe both classes and take notes
   b. Look through online materials provided to students

2. Interviews
   a. Interview students from both classes

3. Surveys
   a. The CS department has already collected some survey data

4. Records (grades and scores)
   a. The CS department has provided me with grades and scores from Fall 2017 and Winter 2018

9.4.3 Teaching Techniques

In modern literature related to computer science teaching, hot topics include paired programming and communities of learning, unplugged learning, modelling levels of abstraction, game design (use-modify-create, flipped classroom, live coding, code tracing, subgoal modelling (problem decomposition), and developing a deeper understanding of underlying principles of programming. This section takes a critical look at the two existing courses to evaluate which course is using which teaching techniques and how effectively these techniques are employed.

9.4.3.1 Lecture-based Version

The lecture-based version of CS 235 as it was taught in Winter 2018 devoted most of its class time to lecture, as the name implies. These sections had higher enrollment (85 to 135 students) and were held in a larger lecture hall, which makes a better environment for lecture than for discussion or flipped classroom. Lectures were driven by power point slides, which were made available to the students online. The slides include useful code samples, diagrams, graphics, and text. Many of the code samples include helpful labels that describe what the code is doing in plain English (cf. Fig. 9.1). The text highlights and bolds new terms and provides definitions (cf. Fig. 9.2).
**Class Computer**

```cpp
#include <string>

class Computer {
    private:
        std::string manufacturer;
        std::string processor;
        int ramSize;
        int diskSize;
    public:
        Computer(const std::string& man, 
                  const std::string& proc, 
                  int ram, int disk) : 
            manufacturer(man), 
            processor(proc), 
            ramSize(ram), 
            diskSize(disk) {} 
        int getRamSize() const { return ramSize; }
        int getDiskSize() const { return diskSize; }
        std::string toString() const;
    };
}
```

Figure 9.1. An example of a slide with code samples and descriptive labels.

**Is-a Versus Has-a Relationships**

- The **is-a** relationship between classes means that every instance of one class is also an instance of the other class (but not the other way around).
  - A jet airplane is an airplane, but not all airplanes are jet airplanes.
  - The jet airplane class is derived from an airplane class.
  - The **is-a** relationship is represented in object oriented programming by extending a class.

- The **has-a** relationship between classes means that every instance of one class is or may be associated with one or more instances of the other.
  - For example, a jet plane has-a jet engine.
  - The **has-a** relationship is represented by declaring in one class a data field whose type is another class.

Figure 9.2. An example of a slide that defines and explains new terms.
The instructor (Dr. Roper) uses the slides masterfully, timing slide transitions carefully to keep students engaged. Rather than just reading through the slides, he expounds upon what is shown. He gives students time to think about and discuss questions from the slides before revealing the answer.

From the previously mentioned set of popular topics in computer science teaching, this course focuses on code tracing, subgoal modeling, and developing a deeper understanding of the underlying principles of programming. Code tracing is evidenced by the slides containing a lot of sample code that students are expected to read through and understand. Subgoal modeling is done in lecture as the instructor starts by describing a general problem and then walking through the process of decomposing it into manageable chunks. There is also evidence of subgoal modeling in the lab rubrics, which break down the labs into manageable pieces instead of the traditional all-or-nothing method of lab grading. Developing a deeper understanding of the underlying principles of programming is really the foundation of this course. There is some discussion of C++ syntax, but the focus is primarily on transferable language semantics, underlying constructs and structures, and developing usable software artifacts. This is evidenced by the slides, the textbook, and the labs.

Students are told in the course syllabus that they can discuss labs and course material with their classmates and study groups, but they must do their own work. In essence, students can discuss concepts in the abstract with others, but must write their code on their own. In my own experience taking computer science classes at BYU, this meant work alone because the line between collaborating and cheating was so poorly defined. All of the students that I interviewed from this lecture-based course reported doing labs on their own, sometimes with the help of the TAs but never turning to other classmates for help. Although group work is not specifically discouraged, the poorly defined rules of working with others discourage students from helping each other for fear that it might be considered cheating.

9.4.3.2 Flipped Version

The flipped version of CS 235 as it was taught in Winter 2018 devoted most of its class time to discussion, live coding, and students writing code. The section had lower enrollment (60 students) because the department deliberately capped it at 60 students to make the flipped classroom approach more feasible. Lectures often included a brief review of reading assignments and small assignments, a live coding session showing students how to use concepts they need to use for the labs, class discussions around practice problems or practice interview questions involving course content, and students working on labs either individually or with a partner.

The live coding sessions are difficult to design because they need to help students understand the concepts they have to use for the labs without helping too much. They need to relate to the labs enough so that students can understand how to apply the concepts to the labs but avoid being so close to the labs that students can code the labs without really understanding the concepts. For example, the first lab involves students using classes, inheritance, polymorphism, and overloaded functions to make characters for an RPG. The live coding session needs to showcase the use of classes, inheritance, polymorphism, and overloaded functions without being too close to the RPG lab. The instructor (Dr. Clement) used a pinewood derby simulation program (cf. Fig. 9.3) to teach the principles without giving away any code specific to the RPG lab. There is no assignment tied to the pinewood derby simulation program; it is just an in-class activity used
to help teach the concepts of the lab. Students are asked to develop artifacts as part of the in-
class activity, but the artifacts are for their own benefit rather than for a grade.

Pinewood derby simulation program

If you have worked with Cub Scouts, you know they get really excited about the pinewood derby race. If your Cub Scout Pack is low on funds, they probably only have a 2 lane track and so you will have to race each pair of cars against each other to determine the winner. Some cars have rockets on them, some cars have big stuffed animals on top that increase the wind resistance, but all of them need to be kept in the race container. Build an application to simulate a pinewood derby race.

1. First, let's design our application. What are the objects and what are the "isa" and "hasa" relationships?
   • Our main has a pointer to a Race. You will need to create a new Race object that should inherit from the Raceinterface abstract class.
   • You should have a rocket and panda car classes that inherit from the Carinterface abstract class.
   • Practice drawing the UML diagram for your application.

2. Now add the functions that will be needed for each of these objects
3. Once you have the UML document, you can start creating concrete classes that inherit from your abstract classes.
   • Create a concrete class for Race that inherits from Raceinterface, put it in Race.h Put the implementation in Race.cpp
   • Create a concrete class for Car that inherits from Carinterface, put it in Car.h Put the implementation in Car.cpp
   • Create a concrete class for Rocket that inherits from Car.h Put it in Rocket.h Put the implementation in Rocket.cpp
   • Create a concrete class for Panda that inherits from Car.h Put it in Panda.h Put the implementation in Panda.cpp

4. Now you have finished the design, let's go to the implementation

Figure 9.3. An example of a problem developed for a live coding session to showcase important concepts needed for the coding lab.

From the previously mentioned set of popular topics in computer science teaching, this version of the course focuses on pair programming, flipped classroom, live coding, code tracing, subgoal modeling, and developing a deeper understanding of the underlying principles of programming. Students are encouraged (but never required) to work together on labs and are even allowed to turn in the same code as long as they both understand it. They are also encouraged to discuss in-class practice problems with their neighbors. Students are strongly encouraged to bring a laptop to class to code along with the instructor and to work on labs and other assignments. Subgoal modeling is done in lecture as the instructor starts by describing a general problem and then walking through the process of decomposing it into manageable chunks. There is also evidence of subgoal modeling in the lab rubrics, which break down the labs into manageable pieces instead of the traditional all-or-nothing method of lab grading. There is some discussion of C++ syntax in this version, but the focus is primarily on transferable language semantics, underlying
constructs and structures, and developing usable software artifacts, as evidenced by in-class activities and the labs.

9.4.3.3 Conclusions

The flipped version of the course employs more teaching techniques from modern literature than the lecture-based version by including pair programming, flipped classroom, and live coding. It is also better designed to encourage students to rely on each other for support.

However, one limitation of the flipped version of the course is that the reading material is lacking both in terms of quantity and quality. Without the textbook and the slides from the lecture-based version of the class, there is less text for students to turn to and the textual resources that do exist are not as helpful as they could be. One major improvement that could be made to the flipped version of the course is to increase the number of reading assignments and improve the quality of existing reading assignments. Using the content from the slides from the lecture-based version of the course would be an excellent starting point.

9.4.4 Assessments

In CS 235, major assessments include exams and coding labs. Minor assessments typically include reading quizzes, attendance points, and/or small homework assignments. Refer to Table 9.4 for a list of assessments and how they compare for each class.

<table>
<thead>
<tr>
<th>Assessment</th>
<th>Lecture-based Version</th>
<th>Flipped Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 1</td>
<td>Grades</td>
<td>RPG</td>
</tr>
<tr>
<td></td>
<td>Output the average score for all students on each exam followed by a list of student's score and grades.</td>
<td>Write implementations of different types of fighter classes that inherit from a parent fighter class.</td>
</tr>
<tr>
<td></td>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L01-Grades/grades.php">https://students.cs.byu.edu/~cs235ta/labs/L01-Grades/grades.php</a></td>
<td>- Write a complete C++ solution.</td>
</tr>
<tr>
<td></td>
<td>● Use the Visual Studio IDE to write and test your program.</td>
<td>● Use basic I/O manipulators.</td>
</tr>
<tr>
<td></td>
<td>● Write a complete C++ solution.</td>
<td>● Design classes to store and access private data members.</td>
</tr>
<tr>
<td></td>
<td>● Familiarize yourself with breakpoints, single-step, and memory examination.</td>
<td>● Use inheritance.</td>
</tr>
<tr>
<td></td>
<td>● Use command line arguments to select input and output files.</td>
<td>● Use polymorphism.</td>
</tr>
<tr>
<td></td>
<td>● Use C++ dynamic arrays for run-time data storage.</td>
<td>● Overload functions.</td>
</tr>
<tr>
<td></td>
<td>● Use basic I/O manipulators to format output values.</td>
<td>● Use objects to manage resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.6%</td>
</tr>
<tr>
<td>5%</td>
<td>Expressions</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Lab 2</td>
<td>Design classes for a school database and output course and student schedules.</td>
<td></td>
</tr>
<tr>
<td>Lab 3</td>
<td>Implement an array container class with a nested iterator class.</td>
<td></td>
</tr>
<tr>
<td>Lab 4</td>
<td>Develop a Linked List that performs Maze</td>
<td></td>
</tr>
</tbody>
</table>

Expressions
- Perform several operations on infix and postfix expressions.
  - Use an STL stack in parsing expressions.
  - Convert expressions from infix to postfix notation.
  - Convert expressions from infix to prefix notation.
  - Evaluate postfix expressions using an operator and operand stack.

Lab 2
- Design classes to store and access private data members.
- Create classes that include toString method.
- Use inheritance.
- Use polymorphism.
- Overload functions.
- Use objects to manage resources.

Lab 3
- Created a template class container.
- Added a nested iterator class to a container.
- Overloaded class operators.
- Used insertion operator ("<<") to access container data.
- Implemented toString and friend class methods.

Lab 4
- Use STL lists, sets, and maps
- Use file I/O to read in text documents and write text to files
- Solve a basic machine learning problem
<table>
<thead>
<tr>
<th>Lab 5</th>
<th>Expressions (infix, postfix, balance)</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Performs several operations on infix and postfix expressions.</td>
<td>Implement a quicksort algorithm and sort a data set.</td>
</tr>
<tr>
<td></td>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L05-Expressions/expressions.php?TheLab=1">https://students.cs.byu.edu/~cs235ta/labs/L05-Expressions/expressions.php?TheLab=1</a></td>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L06-Railroad/railroad.php">https://students.cs.byu.edu/~cs235ta/labs/L06-Railroad/railroad.php</a></td>
</tr>
<tr>
<td></td>
<td>- Use an STL stack in parsing expressions. \n- Convert expressions from infix to postfix notation. \n- Convert expressions from infix to prefix notation. \n- Evaluate postfix expressions using an operator and operand stack.</td>
<td>- Understand and explain the Quicksort Algorithm \n- Use recursion to quickly sort data. \n- Properly manage memory in arrays.</td>
</tr>
<tr>
<td>5%</td>
<td></td>
<td>5.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lab 6</th>
<th>Railyard (stacks, queues, deques)</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manage the Greater Provo Railroad Station storage facility as directed.</td>
<td>Develop a Linked List that performs several standard operations.</td>
</tr>
<tr>
<td></td>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L06-Railroad/railroad.php">https://students.cs.byu.edu/~cs235ta/labs/L06-Railroad/railroad.php</a></td>
<td>- Develop a Linked List that performs several standard operations.</td>
</tr>
<tr>
<td></td>
<td>- Create and implement a Deque container with constant</td>
<td>- Develop a C++ solution with the use of templates.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lab 7</td>
<td>3D Maze (recursion)</td>
<td>BST</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Two SCUBA divers need to find a path through the storage facility maze.</td>
<td>Add and remove nodes from a binary search tree.</td>
<td></td>
</tr>
<tr>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L07-3dMaze/3dmaze.php">https://students.cs.byu.edu/~cs235ta/labs/L07-3dMaze/3dmaze.php</a></td>
<td>- Develop a Binary Search Tree data structure that performs several standard operations using recursion.</td>
<td></td>
</tr>
<tr>
<td>- Use recursion to solve complex problems.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Take a real world problem and create an algorithm to solve it.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Use recursive backtracking to solve a problem.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Dynamically allocate a multidimensional array.</td>
<td>- Use pointer references as parameters to function calls to modify pointers in the tree.</td>
<td></td>
</tr>
<tr>
<td>5%</td>
<td>- Handle memory management in C++.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lab 8</th>
<th>BST</th>
<th>AVL Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add and remove nodes from a binary search tree.</td>
<td>Use AVL algorithm to self-balance a binary search tree.</td>
<td></td>
</tr>
<tr>
<td><a href="https://students.cs.byu.edu/~cs235ta/labs/L08-BST/bst.php">https://students.cs.byu.edu/~cs235ta/labs/L08-BST/bst.php</a></td>
<td>- Determine if a Binary Search Tree is critically imbalanced and distinguish between the various types of imbalance.</td>
<td></td>
</tr>
<tr>
<td>- Develop a Binary Search Tree data structure that performs several standard operations using recursion.</td>
<td>- Implement functions to rotate nodes and balance a Binary Search Tree.</td>
<td></td>
</tr>
<tr>
<td>Lab 9</td>
<td>Maps and Sets</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add and remove nodes from map and set data structures.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Implement a Set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Implement a Hash Map.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Understand how to iterate through a Set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Use Map creation and lookup to solve a variety of problems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Lab 10</td>
<td>Quicksort</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement a quicksort algorithm and sort a data set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Understand and explain the Quicksort Algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Use recursion to quickly sort data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Properly manage memory in arrays.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Lab 11</td>
<td>AVL Tree (Bonus)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use AVL algorithm to self-balance a binary search tree.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>
- Determine if a Binary Search Tree is critically imbalanced and distinguish between the various types of imbalance.
- Implement functions to rotate nodes and balance a Binary Search Tree.
- Insert and remove nodes from a Binary Search Tree while maintaining tree balance.

5% (extra credit)

<table>
<thead>
<tr>
<th>Small assignments</th>
<th>Daily attendance quizzes that assess material covered in class and the book, help prepare for exam. Required quiz code given in class. Two points for correct answer on the first try, one point for correct answer on the second try.</th>
<th>Small assignments include reading assignments, reading quizzes, and small coding assignments.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>Pre-exam</td>
<td>Required but score does not affect final grade, just get points for completion</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>Midterm 1</td>
<td>Chapters 1-5</td>
<td>Queues</td>
</tr>
<tr>
<td></td>
<td>Software life cycle</td>
<td>Stacks</td>
</tr>
<tr>
<td></td>
<td>Inheritance, hierarchies</td>
<td>Sequential containers</td>
</tr>
<tr>
<td></td>
<td>Correctness and efficiency</td>
<td>Recursion</td>
</tr>
<tr>
<td></td>
<td>Sequential containers</td>
<td>Sorting</td>
</tr>
<tr>
<td></td>
<td>Stacks</td>
<td>Efficiency</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>Binary Search</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.4%</td>
</tr>
<tr>
<td>Midterm 2</td>
<td>Chapters 6-8</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Queues and deques</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recursion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trees</td>
<td></td>
</tr>
</tbody>
</table>
The key differences in assessment between the two versions are as follows:

1. The lecture-based version has two introductory labs whereas the flipped version has a single introductory lab that combines the important outcomes together.
2. Many of the labs are the same but they are in a different order. The labs in the lecture-based version are arranged according to content in the textbook whereas the flipped version arranges labs according to course learning outcomes.
3. The lecture-based version has the railyard lab which is dedicated specifically to helping students understand cases to use has-a relationships instead of is-a relationships.
4. The AVL lab is the hardest lab in the course according to multiple students and instructors. This lab is optional extra credit in the lecture-based version but required in the flipped version.
5. Small assignments in the lecture-based version include only reading quizzes whereas the flipped version has both reading quizzes and small coding assignments for formative and diagnostic purposes.
6. Only students who attend class can complete the small assignments in the lecture-based version, whereas attendance is not mandatory to complete small assignments in the flipped version.
7. The lecture-based version has a pre-exam, two midterms, and a final that together make up 40% of a student’s final grade. The flipped version has a midterm and a final that together make up 30% of a student’s final grade.

For both versions of the course, the labs generally align with learning outcomes. The outcomes for each lab are more explicitly stated in the lecture-based version of the course, but the labs from the flipped version could be modified to state their outcomes as explicitly. The lecture-based labs are ordered by content first then learning outcome (e.g., learn how to use a stack, make your own stack, learn how to use a queue, make your own queue) whereas the flipped labs are ordered by learning outcome first and then by content (e.g., learn how to use a stack, learn how to use a queue, make your own stack, make your own queue). Ordering by learning outcome first brings more attention to the learning outcomes and also creates a course where the content all interrelates instead of making the different content areas seem disjointed. The learning outcomes and content areas are all represented, but trees and recursion are less represented in the lecture-based version due to the fact that the AVL lab is optional instead of required.
In terms of teaching using practical real-life scenarios, both courses are roughly equal due to the close similarities between the labs. However, the flipped version introduces some concepts of machine learning that make the map labs seem more practical and less abstract. The quicksort, BST, and AVL labs are all more abstract, but necessarily so because they are abstract principles that are designed to be applied to a wide range of scenarios.

The assessments of both versions of the course prepare students for related problems in industry, but I think the flipped version of the course does a better job of it. The most applicable content from this course that relates to industry generally is the content about runtime efficiency and hashmaps. The flipped version of the course introduces hashmaps early on and has two labs about hashmaps. In industry, developing programs with fast lookup times is a frequent requirement, and understanding hashmaps is critical for software developers to write code with fast lookup times. Many job interview questions relate to algorithmic efficiency, and many of those interview questions can be answered best by using a hashmap or a related structure.

9.4.4.1 Exam Specifications

To get a better idea of how well the exams align with course learning outcomes, I reverse-engineered a test specification from the midterm of the flipped version of the course. I started by identifying the general instructional objectives (GIOs) and student learning outcomes (SLO) from the midterm:

1. Knows basic terminology, facts, principles, and algorithms
   a. Recalls basic terms and facts
   b. Identifies steps of common algorithms
   c. Identifies important principles
2. Understands important concepts
   a. Identifies benefits and limitations
   b. Identifies examples of best practice
   c. Explains concepts given specific context
      i. Identifies program output given specific code
      ii. Identifies program state given specific code
      iii. Identifies program state given the name of a common algorithm and specific input
      iv. Computes complexity given specific code
      v. Computes complexity given mappings from input sizes to runtimes
3. Applies theory to practical situations
   a. Identifies uses for specific data structures
   b. Selects the best data structure to use in solving various real-world problems

I also created an outline of the general content areas that the midterm covers:

1. Data Structures
   a. Difference between LIFO, FIFO, and random access structures
   b. Strengths and limitations of LIFO, FIFO, and random access structures
   c. Behavior of common operations for stacks, queues, and lists
   d. Common uses for stacks, queues, and lists
2. Recursion
3. Algorithmic Complexity
   a. Big-Oh notation
   b. Big-Oh complexity of iterative code
   c. Best case and worst case Big-Oh complexity of common sorting algorithms

4. Sorting Algorithms
   a. Difference between common sorting algorithms (bubble sort, selection sort, insertion sort, merge sort, and quicksort)
   b. High-level steps of common sorting algorithms
   c. Strengths and limitations of common sorting algorithms

With the learning outcomes and the content areas mapped out, I was able to analyze each of the 79 objectively-scored test items to generate a table of specifications (cf. Table 9.5). In terms of content, the test is fairly balanced but could be improved. Data structures and algorithmic complexity get similar coverage, but the amount of items related to sorting algorithms is disproportionate to the importance of sorting algorithms compared to the importance of other content areas. I would recommend removing some questions about sorting algorithms to make it cover roughly 20% of the exam and add some questions about data structures and algorithmic complexity to make them each cover 30-40% of the exam. The exam emphasizes understanding over recognition and recall, but the exam should include more application level items.

<table>
<thead>
<tr>
<th>Content</th>
<th>Know</th>
<th>Understand</th>
<th>Apply</th>
<th>Analyze/Evaluate</th>
<th>Total #</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data structures</td>
<td>-</td>
<td>18</td>
<td>4</td>
<td>-</td>
<td>22</td>
<td>27.9</td>
</tr>
<tr>
<td>Recursion</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>Algorithmic</td>
<td>11</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>20</td>
<td>25.3</td>
</tr>
<tr>
<td>complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting algorithms</td>
<td>17</td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>35</td>
<td>44.3</td>
</tr>
<tr>
<td>Total #</td>
<td>28</td>
<td>47</td>
<td>4</td>
<td>-</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>Percentage</td>
<td>35.4</td>
<td>59.5</td>
<td>5.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.5: A table of specifications for the midterm of the flipped version of the course.

I did not analyze the exams for the lecture-based version of the course, but the students that I interviewed did not feel that the exams were fair or an accurate measure of their achievement of the learning outcomes. One interviewee from the lecture-based version of the course stated the following about the exams:

The exams were not good. The questions were extremely ambiguous; it was nearly impossible to understand what he was trying to ask for probably more than half of the questions. The answers were also often not understandable. There were many cases where at least two answers could be equally possible and there were some cases where the correct answers directly refuted what was in the textbook. Likewise, [the instructor] also said things in class that directly refuted the textbook, so we were all left wondering what to put on the tests.
Between my own analysis of the midterm for the flipped course and the student opinions of the exams for the lecture-based course, it seems that the assessments that are most in need of improvement are the written exams. They need to be modified so that they accurately measure learning outcomes and measure them in proportion to their overall importance.

9.4.4.2 Final Grades

In the learner analysis I look at the final grades that students earned in the Winter 2018 semester and I look for correlations between student characteristics and final grades. In this section I look for correlations between which version of the course was used and the students’ final grades to get an idea of the level of difficulty and how well students were prepared for their assessments. Figures 7.6 and 7.7 show the grade distributions for students in the lecture-based course and students in the flipped course, respectively.

Figure 9.6. Grade distribution for the 418 students who took the lecture-based version of CS 235 taught in Winter 2018.
These figures show that grades were much more polarized in the flipped version of the course. 83.3% of students got a B or better, and 8.3% either withdrew or got a D- or worse. The lecture-based version shows a full spectrum with fewer people getting As and more getting Bs, Cs, and Ds with 17.5% who either withdrew or got a D- or worse.

The fact that students generally received higher grades in the flipped version of the course can mean a variety of things. First, it might mean that the course is easier than the lecture-based version of the course. The labs are similar in difficulty, but the exams in the lecture-based version count for a higher percentage of the final grade and may be harder than the exams in the flipped version. Student opinions of the exams in the lecture-based version suggest that the exams were harder, perhaps due more to poorly written test items than anything else. Another explanation could be that the students in the flipped course were better prepared for the assessments due to an improved course design. However, I hesitate to make that claim due to the small sample size of 60 students for the flipped course. It is possible that the 60 students started with greater skill levels or better study skills from the outset. Also, the instructor for the flipped version of the course had only one section of 60 students whereas the instructor for the lecture-based version of the course had four sections of 85-135 students each. The instructor for the flipped course had greater ability to give individual time to each student both inside and outside of class, which may account for the students’ better performance. More data are required before making a strong claim, but it is possible that students performed better because of an improved course design.

9.4.4.3 Conclusions

The labs in the flipped version of the course are a little better because they teach to the same outcomes using fewer labs, are grouped together by the course learning outcomes, require more of students without being too difficult or unreasonable, use more practical real-life scenarios, and better prepare students for problems commonly seen in industry. The flipped version does not have the railyard lab dedicated specifically to the difference between has-a and is-a relationships, but smaller reading quizzes and coding practices are used to teach students that difference without as much overhead.

Small assignments in the flipped version end up being worth a bigger part of the final grade, which keeps students more accountable for them. Also, the frequent formative and diagnostic feedback from small assignments allows students and instructors to gauge understanding without needing to introduce another midterm. The wide variety of small assignments, along with the coding activities that students actively work on in class, give students more opportunity to actively apply what they are learning as they learn it.

Exams should be designed with a table of specifications to ensure that each content area is balanced in proportion to its importance and that higher level learning outcomes are being measured.

Student performance in the flipped version of the course was shockingly higher, but there are multiple possible explanations. More data are required before making a strong claim, but it is possible that students performed better because of an improved course design.
9.4.5 Technology

In CS 235, technology is generally used for three different reasons: (a) for instructors to deliver content to the students for use outside of the classroom; (b) for students to write and debug code, and (c) for students to submit labs and other assignments. This section compares and contrasts the technologies used for these purposes between the two versions of the course.

To deliver content to the students for use outside of the classroom, the lecture-based version uses a home-baked online content management system (CMS) along with Learning Suite. The CMS is used to share PowerPoint files, share code files that students need to start labs, provide descriptions and rubrics for assignments, share the course syllabus and calendar, provide a help queue for students to request help from TAs, provide a way for students to submit code solutions electronically, and provide a way for students to manage the code reviews they are required to do for each lab. Learning Suite is used to communicate scores and grades and host quizzes. Exams are taken in the Testing Center.

The flipped version uses Canvas and GitHub. Canvas is used to share reading content, provide descriptions and rubrics for assignments, share the course syllabus and calendar, provide a help queue for students to request help from TAs, provide a way for students to submit code solutions electronically, communicate scores and grades, host quizzes, and host exams. GitHub is used to share code files that students need to start labs.

The main pros of using Canvas over a home-baked CMS are (a) Canvas has a clean, unified design; (b) Canvas is more likely to be secure; (c) Canvas can be used as both a CMS and an LMS; (d) Canvas is easier to update; (e) Canvas maintains the servers and databases for you; (f) Canvas is mobile-friendly; and (g) Canvas makes it easier to store multiple versions or sections of a course. The main benefits of using a home-baked CMS over Canvas are that (a) the instructor has total control over the system and (b) it is easier to make custom solutions such as the code review management system. Security and usability are important to me, so I used Canvas rather than a home-baked CMS.

The lecture-based version uses the CMS to provide code files for students while the flipped version uses GitHub. With the CMS, students can download the individual files and then organize them into the development environment one at a time. With GitHub, students can copy all of the files at the same time while preserving the file structure. GitHub also provides students with a way to store their changes and change history online, but that requires a deeper knowledge of Git that CS 235 students likely do not have initially. Git and GitHub are highly used in industry, so students need to learn them as professional programmers anyway. There is no downside that I see to using GitHub, so I used GitHub rather than a CMS or an LMS to provide code files.

In the lecture-based version of the course, students use Visual Studio to write and debug code and they submit a zip file of their code through the home-baked CMS for passoff. Students taking the flipped version use Cloud9, an online development environment, to write and debug code and they submit a Cloud9 URL through Canvas for passoff. None of the students that I interviewed had strong opinions about using Visual Studio or Cloud9. Visual Studio is more robust and the students are already familiar with it, but it only runs on Windows computers. Cloud9 does not have the robust developer tools that Visual Studio has, but it is cloud-based and
Cloud9 is thus accessible on any computer with a browser and an internet connection. Cloud9 also has a built-in Linux terminal that makes it easy to download code from GitHub into Cloud9. However, Cloud9 requires more time up front for students to create accounts and become acquainted with how to use it. There are definitely pros and cons to each development environment, but I used Cloud9 to take advantage of its accessibility.

9.4.5.1 Conclusions

I used Canvas as an LMS and GitHub as a means of providing code files to students. It is harder to make a choice between using Visual Studio and using Cloud9 because the pros and cons are fairly balanced, but I chose Cloud9 because of the benefits that come from it being a cloud-based system.

9.4.6 Evaluation Conclusions

After evaluating both courses, I chose to use the flipped version of the course as a basis for designing my version of the course. It employs more teaching techniques from modern literature and is better designed to encourage student collaboration. The labs are better organized and build off of each other better. A wide variety of small assignments helps keep students accountable and provides formative and diagnostic feedback to both students and instructors. The technology that is used by instructors and students has proven to be effective.

However, there are limitations of the flipped version of the course that need to be addressed. The most significant limitation is the lack of robust textual resources. Also, the exams need to be critically examined and reworked to better balance out the content and measure higher level learning outcomes.

9.5 Consulting Products

The results of my initial evaluation of the existing versions of CS 235 helped me to realize that the most significant limitation that I can address is the lack of robust textual resources, thus I need to get a better understanding of what makes a good textbook for introductory data structures courses. In this section I explore textbooks that are used in similar courses of other higher education institutions to get a better idea of how I can design effective reading material and reading quizzes for my implementation of the course.

In CS 142, students use an online interactive textbook published by ZyBooks (cf. Fig 7.8). There is a ZyBooks textbook for data structures as well, and it is similar. Small blocks of text are separated by participation activities (generally objectively-scored items), animations, and code samples. The participation activities are not coding problems, and thus are better suited for preparing learners for exams rather than for coding labs. Animations are used to visualize algorithm logic or connections between various content. The code samples are written in pseudocode by default, but users have the option to change the code to be language-specific. Because there are no coding problems, the textbook provides sample code for all of the algorithms it discusses, which is one reason why the computer science department does not want to use the ZyBooks textbook for CS 235. The participation activities and the animations are helpful, but the code samples provide too much detail that CS 235 students should be able to figure out on their own. There is
already a problem with students relying so much on code samples that they find in textbooks or online that they are able to complete the labs without really achieving the learning outcomes.

Commonly, a data member has a pair of associated functions: a mutator for setting its value, and an accessor for getting its value, as above. Those functions are also known as a setter and getter functions, respectively, and typically have names that start with set or get.

Additional mutators and accessors may exist that aren’t directly associated with one data member; the above has two additional accessors for getting a player’s high score. These additional mutators and accessors often have names that start with words other than set or get, like compute, find, print, etc.

Accessor functions usually are defined as const, to enforce that they do not change data members. The keyword const after a member functions declaration and definition causes the compiler to report an error if the function modifies a data member. If a const member function calls another member function, that function must also be const.

Figure 9.8. Example content from ZyBooks.

One of the primary goals of the computer science department is to get the computer science program ranked in the top five nation-wide. I looked at the top two computer science programs—MIT’s and Stanford’s—to see what textbooks they use for related courses. Their data structures courses—EECS 6.006 at MIT and CS 166 at Stanford—both use the same textbook: Introduction to Algorithms, 3rd edition by Cormen, Leiserson, Rivest, and Stein (2009). The book has a lot of text with some diagrams and pseudocode. There is a lot of mathematical notation (cf. Fig. 9.9). There are exercises at the end of each chapter and many are mathematical in nature, such as proofs. The book is not language-specific but is rather quite abstract as to apply broadly.

Chapters 6-14 of the book cover similar content to what is currently covered in CS 235; the rest of the content is covered in more advanced algorithms courses such as CS 312.
One of my favorite computer science textbooks from when I was a CS student is Bioinformatics Algorithms: An Active Learning Approach by Compeau and Pevzner (2014). Though the content is different from CS 235 content, the book has strengths that can be applied to computer science textbooks in general. The book has an almost informal tone, which makes it read more like a friend sitting down and explaining concepts than a professor giving a lecture. Instead of having practice questions and coding problems at the end of each chapter, they are spaced throughout the text so that learners can practice applying principles and check their own understanding as they go (cf. Fig 7.10). Code samples are written in pseudocode rather than being language-specific.
9.5.1 Main Implications

If textbooks are not language-specific, they can still be used to teach core principles even if instructors choose to use a different programming language or environment. The downside is that beginning students may not be able to translate pseudocode into code if they are unfamiliar with the programming language and how to implement common constructs. One possible solution would be to provide text that is not language-specific and to also provide a separate guide for the programming language. Due to time constraints, however, I had to design some language-specific content for this project. The computer science department’s ultimate goal is for these materials to be language agnostic, so I strove for that as much as possible.
Coding problems and practice questions should be spaced throughout the text so that learners can apply principles and check understanding as they go. Spaced learning is one of the most well-understood and robust teaching methods (Kang, 2016), so it should be used whenever possible. Students can be encouraged to have a development environment open as they read so that they can solve small coding problems as they come across them in the reading.

Even in the few books that I consult here, there is a wide range of formality in tone. Introduction to Algorithms is very mathematical and formal. Bioinformatics Algorithms, on the other hand, is very informal and conversational. Given that many learners in my audience are relatively new to coding, I think that the informal, conversational tone would be more approachable.

9.6 Learner Analysis

Students coming into CS 235 have a wide range of skill and prior programming experience. Some of them passed CS 142 without difficulty while others had to put a lot of effort into CS 142 just to scrape by with a passing grade. To learn more about the population of learners, I started by analyzing the grade data of all 478 students who took the course in the Winter 2018 semester. I looked into three variables that I thought might affect student performance in CS 235: gender, declared major, and grade earned in CS 142 (the prerequisite programming course). In addition to a quantitative analysis of grade data, I met with five students and three instructors to get a better idea of what students liked, disliked, and struggled with in the course.

9.6.1 Grade Analysis

I decided to measure the impact of gender on CS 235 performance because it has often been said that females are marginalized in computer science. At BYU it is true that fewer females take computer science courses: In Winter 2018, 14.9% of students who registered for CS 235 were female. I wanted to look at how females performed in the course compared to males to see if their performance was significantly better or worse. The data show that female students performed very similarly to their male counterparts (cf. Fig. 9.11). The only notable exception is that only 8.4% of female students failed or withdrew while 16.7% of male students failed or withdrew. Measuring the correlation between the number of females and the number of males who received each possible grade yielded a correlation of 0.95, again signifying that gender had no significant impact on overall performance. This suggests that as I design the course, I do not need to put any special effort into making sure that the content and learning activities are designed to be of specific interest to female students.

However, there are some historic data to refute that. In speaking with instructors who have taught over a long period of time, I learned that CS 235 has historically scared off the vast majority of female students. The computer science department has worked on making changes to CS 235 to make it more appealing to female students, and these data show that they have been successful. This suggests that I need to continue along the foundation that they have laid for me in regards to reaching out to the female students, but I do not need to do anything extra on top of that.
The next variable I explored was the declared major of each student. My initial hypothesis was that declared computer science majors would outperform students from other majors. CS 235 covers some deeper topics, such as algorithm efficiency and recursion, that I thought would appeal more to computer science majors than engineers or IT majors. However, the grade distribution by major (cf. Fig. 9.12) illustrates that there is no significant difference between how computer science majors performed in comparison to other majors. Measuring the correlation between the number of computer science majors and the number of other majors who received each possible grade yielded a correlation of 0.96. There are some anomalies (e.g., for statistics majors it was more common to withdraw than to get an A), but overall there is no significant difference between the performance of computer science majors and other students, proving my initial hypothesis to be incorrect. This suggests that as I design the course, I do not need to put any special effort into making sure that the content and learning activities are designed to be of specific interest to students from other majors.

**Figure 9.11.** CS 235 grade distribution by gender for the Winter 2018 semester.
The last correlation I explored was that between each student’s CS 142 grade and CS 235 grade. To measure the correlation, I converted each letter grade to a numerical grade on a 4.0 scale (A is 4.0, A- is 3.7, B+ is 3.3, etc.) and used Pearson’s $r$ to get a correlation of 0.43, which is statistically significant at $p < 0.01$. Out of 379 students who did well (B or above) in CS 142, 59 of them (15.6%) did poorly (C- or below) in CS 235. Also, out of 22 students who did poorly in CS 142, only 4 of them (18%) did well in CS 235. There is a moderate positive correlation between performance in CS 142 and performance in CS 235 with evidence that those who are most at risk of doing poorly in CS 235 are those who performed poorly in CS 142, but performing well in CS 142 does not necessarily guarantee success in CS 235.

For those who did poorly in CS 142 and did well in CS 235, there are many possible explanations: improved study skills, improved motivation, fewer things detracting from study time, extra effort to improve technical skills and understanding, increased maturity, etc. I hope to be able to make some design decisions that can help those who did poorly in CS 142 to catch up to speed without compromising the rigor of the course, but the best advice to give to these students might be to have them retake CS 142 until their skill level is adequate in those important prerequisite skills.

### 9.6.2 Interview Analysis

During the Winter 2018 semester, I talked with five students and three instructors. Three of the students were in the lecture-based version and two were in the flipped version. Of the instructors, one taught the flipped version in Winter 2018 and the other two had taught in previous semesters. These interviews helped me to get a feel for what students like, dislike, and struggle with. I asked questions about the development environment, the use of class time, the textbook, communities of learning, and content that students find challenging. Even though I only spoke with a small sample of the population, I saw some noteworthy themes amongst the responses.
Two different development environments were used in Winter 2018 depending on the instructor. One instructor encouraged students to use Visual Studio, which is made available on the CS lab computers. The other instructor used Cloud9, an online open source development environment that has built-in support for C++. The main benefit of Visual Studio is that it is very robust: It is easy to use and it has a lot of features with good documentation and support. However, Visual Studio is only available on Windows and it is not cloud-based, constraining some students to only be able to work on their coding labs from the CS lab computers. Some of these students did not mind having to use the CS lab computers, but there were some students who live and work off-campus and thus found it difficult to schedule time to spend in the CS lab. Cloud9 has the benefit of being accessible from any device with an internet connection, but it lacks the robust developer tools that are available in Visual Studio. However, students who used Cloud9 reported that they found the available developer tools to be good enough for what they needed to do, and they appreciated the mobility.

Another difference between the two versions of the course was that one did traditional lectures with slides containing helpful diagrams and code samples and the other used a flipped classroom paradigm where students brought a laptop to class and worked on coding problems related to the labs. In the flipped classroom version of the course, the instructor started class by teaching a principle and then did some live coding to show students some practical application of the principle. Sometimes the instructor had issues with unexpected bugs in his code, but he used these opportunities to teach students how to debug effectively. He then posed some coding problems to the class and allowed students to work individually or in small groups to come up with answers. When I asked the students what they would prefer, none of them had any strong opinions one way or the other. They all were fine with how their instructor had decided to do things.

Both versions of the course required the same textbook: Objects, Abstraction, Data Structures and Design Using C++ by Koffman and Wolfgang (2005). Regarding the textbook, I got some mixed responses. The instructors said that it is a decent book, but they do not think that it is necessary. One instructor chose to not use the textbook and instead linked students out to Wikipedia and other online resources to help introduce them to topics before class. Another instructor said that he thinks the students might actually be better off without the textbook. Of the students who took the class where the textbook was required, most said that they did not do the reading because they felt like they learned the material better from the lecture or online resources. However, one student I interviewed relied heavily on the book and attended class less frequently, and he did very well in the class. Of the students I interviewed who took the class that relied on open resources, all reported that they at least glanced at most reading assignments. When I asked what kind of resources they prefer to use to learn coding skills on their own, most mentioned online resources as their first preference. Based on the feedback that I got from students and instructors, it makes sense for me to list the textbook as optional rather than required. For each reading assignment, I list sections of the book as well as open resources so that those who learn best by reading are able to do so.

I also asked the students and the instructors about their thoughts on communities of learning. One instructor reported that he encourages students to talk together about projects, but that each student has to write and turn in code separately. Another instructor reported that he did away with the rule that students have to write their own code and instead implemented a rule that if students submit code that they could not explain or reproduce on their own, then they are
in the wrong. In talking with the students about communities of learning, I got some mixed responses. Some of them have very strong preferences for working alone. In CS 142 students did a lab where pair programming was required, and some of them had a hard time with it due to social inhibitions, communication issues, scheduling conflicts, unfair division of tasks, etc. Others reported that they enjoy working in groups of two to three so that they can bounce ideas off of each other and help each other get unstuck. Given the responses I received, it makes sense to encourage but not require working in small groups for the coding labs.

Because CS 235 is a sophomore-level course it is not uncommon for students to take it just after returning home from an LDS mission, so they might need some time to recall the fundamentals before jumping into new material. One of my interviewees said, “I went on a mission between CS 142 and CS 235, so jumping right into objects and classes and data structures was hard. I think just a week or two to refresh the stuff from 142 that we need to know would be really helpful.” One of the instructors I interviewed said that roughly one-third of CS 235 students report at the beginning of the semester that they cannot write a simple C++ program despite the CS 142 prerequisite.

9.6.2.1 Course-level Content Issues

When I asked students about what content they struggled with the most, the most common answer was recursion. One interviewee’s response sums it up:

Recursion was the most difficult part of the course for me. I understood the data structures, Big-Oh stuff, and sorting algorithms pretty well. When the instructor talked about recursion I felt like I understood but then when I was asked to do recursion for the labs, it was hard to write and hard to think through. I could see a simple recursive function and tell you what it was doing, but the kind of recursion required in the labs was just so much more complex than the stuff we talked about in class. We talked about how recursion works but not really how to write recursive code. I didn’t feel like we were really prepared for it. I had to get lots of help from the TAs for those labs. I couldn’t confidently explain to you exactly how my code for those labs works.

This student’s experience aligns with the observations of the instructors. One instructor noted that many of the students who were able to complete the recursion labs performed poorly on the final exam items that required students to write simple recursive functions. Of all the course learning outcomes, gaining the ability to write good recursive code seems to be the hardest for students to achieve.

Another struggle that was mentioned by students was understanding the difference between inheritance and composition (also referred to as “is-a” and “has-a” relationships, respectively). One interviewee reported his confusion:

I feel like CS 142 taught us about inheritance and how to use inheritance to reuse code. In 235 we talked about all this has-a versus is-a stuff and I just don’t understand how it was supposed to apply to the labs. [The instructor] really wanted us to use the has-a relationship and talked about how has-a is often better than is-a, but in CS 142 the emphasis was on that is-a relationship. I’m just left feeling confused about when to use
which and why he wanted us to use has-a for the train lab instead of is-a. I feel like it could be done either way.

The main limitation of this interview analysis is that it is limited to experiences of the five students interviewed. The preferences and struggles represented in this analysis are not necessarily representative of the population. However, the feedback I did get provides me with a solid starting point that I can use to empathize with students as I get started on course design. Throughout the implementation I reached out to students to get a feel for their struggles and I iteratively implemented changes as appropriate.

9.6.3 Survey Analysis

In addition to the grade data and the interview data that I obtained, the computer science department provided me some CS 235 student survey data from Fall 2016 to Fall 2017. The data I was given are student-centric and include student grades, individual assignment grades, and student responses to 56 survey questions. It is possible that analyzing these data may reveal other variables of interest that help predict student success in CS 235. I have not yet analyzed these data, but I plan to work on data analysis during the implementation of my project. As I found any other variables of interest, I iteratively made changes to my course design and reported my findings to the computer science department.

9.6.4 Main Implications

From doing the grade analysis I learned that neither gender nor declared major play a significant role in student performance. This means that as I design the course, I do not need to be especially concerned with modifying content to be more accessible or attractive to any specific gender or any specific major. This goes against my initial instinct to put extra effort into accommodating the needs of female students and non-CS majors because I had believed them to be at a disadvantage. I also verified that there is a significant correlation between student performance in CS 142 and CS 235, which makes sense because the skills learned in CS 142 are unquestionably prerequisite skills needed to succeed in CS 235. As I make design choices, I should consider ways to help those who did poorly in CS 142 to catch up to speed without compromising the rigor of the course.

From the interview analysis I learned the following:

1. Some students like the textbook, but many did not use it and it is not a necessary component for student success;
2. There are pros and cons to using Visual Studio or Cloud9, but the accessibility of Cloud9 outweighs its limitations;
3. Students had no strong opinions about doing the flipped classroom with live coding approach versus the lecture-based approach;
4. Students don’t want to be forced to do group work or participate in communities of learning but want it as an option;
5. Recently returned missionaries (among others) may need some extra review before jumping into new material; and
6. Students struggle most with writing recursive code and with understanding the pros and cons of inheritance versus composition.
9.7 Environmental Analysis

One of the greatest limitations I faced is time. I was given about six weeks at the end of Winter 2018 to observe and evaluate the two versions of CS 235 and to gather data for my front-end analysis. I had roughly eight weeks to write up my analysis and do the design and development work necessary before the start of Fall 2018, which started September 4 and ended December 20. During that time I had to refine the design and develop materials because I was not able to do all of the design and development work in the eight weeks that I had before the start of term. However, I scheduled all of the Winter 2019 semester for evaluation work which included further analysis, design, and development that were informed by my experiences with the implementation.

My classroom resources include a podium, whiteboards, a computer connected to a projector, and a high-speed internet connection. The projector made it easy to do activities such as live coding and getting student responses in real time. The room itself has 60 seats that are immobile with a stadium seating configuration that makes any sort of rearrangement impossible. When students were asked to split into groups during class, they had to split up into groups of two or three at the most. At the start of term all 60 slots were filled, but I was told by an experienced instructor that enrollment would be down to 50 by the add/drop deadline and I would likely see an average of 30-40 students in each class session. The room is not a computer lab, but the computer science department has permitted me to tell students that they must bring their own devices to class if I choose to use a flipped classroom design.

Outside of the classroom the students have access to the computer science department computer labs where they have resources such as Visual Studio to help organize their code. There are also TAs to help answer students’ questions and to help with grading. Reading materials are online as is Cloud9 (development environment), Canvas (LMS), and GitHub (code repository system for me to share source code with students). All of the necessary materials are online, making it possible to easily convert the course into an entirely online course.

I have experienced some difficulty in trying to get the online Cloud9 development environment set up. It used to be independent but now has been acquired by AWS, and I do not know if it still works the way that it used it. If it does not and I cannot set it up, then I may have to use Visual Studio instead of Cloud9. I have done some research into other online C++ development environments (C++ Shell, Online GDB, Codechef, Tutorials Point, Replit, Geeks for Geeks, Ideone, and JDoodle) but I cannot find one that handles multiple files well, which is a requirement. If I cannot get Cloud9 up and running, I may have to use an offline IDE such as Visual Studio, which only runs on Windows. Using Visual Studio would greatly limit the extent to which I could follow a flipped classroom paradigm because I could not require each student to bring a laptop with Visual Studio installed. As an alternative that would allow me to follow a flipped classroom paradigm, I could teach in an IDE-free environment where students are expected to use Bash and Vim to write, compile, and execute code.

I have met with computer science faculty several times before the start of term. They have communicated various ideas for my project which are outlined as follows:

1. Evaluate the two current versions of the class and choose one of them as a base model rather than starting from scratch.
This constraint is partly due to the fact that I did not have the time to develop an entire course from scratch, but the reasons go deeper than that. The computer science department has a policy that projects and assessments should be the same (or at least very similar) across instructors so that all students are graded on the same work. They already have two very different versions of CS 235; they do not need a third. What they want is to know which one is currently doing a better job of helping students to achieve learning outcomes and then have it improved. They want me to make few if any changes to the labs, exams, content model, and learning outcomes and to instead focus my attention on making changes to reading material, micro-assessments, and the use of class time.

2. Develop smaller quizzes and coding problems to keep students accountable for their learning and to help instructors know where students are struggling.

For CS 142, the prerequisite course for CS 235, instructors use an interactive online textbook hosted by ZyBooks. The textbook includes small interactive coding activities that help students to learn the material by applying what they’re reading as they read it. These activities help to keep students accountable and help instructors to see where students are struggling so that they can fill in the gaps. Unfortunately the department does not like how ZyBooks handles the content related to CS 235, so using ZyBooks for CS 235 is out of the question. However, the department does want to see some more micro-assessments (such as small quizzes or coding problems) in CS 235 that fill formative and diagnostic purposes. The exams and the labs as they exist provide a good basis for summative assessment, but formative and diagnostic assessment are lacking.

3. Improve scalability and accessibility while maintaining rigor so that the course can be converted into an online course.

Enrollment in computer science courses is rising steadily and the department is working on scaling up to meet the demand. One of the existing versions of CS 235 uses all online resources, so I can use those established resources to meet this criterion. As I develop additional learning resources, I need to make them available online and keep scalability in mind. Developing ways to automate or simplify grading could allow them to have larger class sizes without putting too much extra burden on instructors or TAs.

4. Help students transition to more proactive, self-paced, student-owned learning.

One of the instructors I met with mentioned that there are a significant number of students who take these early computer science courses have the technical ability but lack the life skills necessary to succeed. As students progress in the program, they are expected to set their own pace and be more proactive. In higher-level courses there usually are not small assignments or quizzes to keep students accountable throughout the semester; there are just big coding labs that are due periodically. Students are expected to get started early in order to finish assignments by their deadlines. In the early classes it’s important to do a little bit more hand-holding because students are not as capable of estimating how much time projects require, but students should be expected to start developing those life skills during this course if they have not already.

5. Help students to help each other.
Ideally, students would form communities of learning so that they could help each other rather than always turning to instructors or TAs for help. TAs and instructors need to be available to help, but it would improve scalability to have students first turn to each other. Computer science students need to learn how to bounce ideas off of each other and learn from each other in the same way that many industry professionals do when they work with their development teams.

9.7.1 Main Implications

The classroom allows me to do live coding exercises and have students work in pairs as they work on problems in class. Online resources give students flexibility as they work on labs and do reading assignments.

The computer science department has given me some specific ideas to consider as I work on the design. First they expect me to evaluate the two current versions of CS 235 and choose one as a base model. Then I can choose to focus on any combination of the following: developing smaller quizzes and coding problems, improving scalability and accessibility, helping students transition to more proactive learning, and/or helping students to help each other.

9.8 Content and Task Analysis

This section includes the content analysis and the task analysis. The content analysis includes a summary of the course content and shows the relationships between content. It is necessary to understand these relationships to determine the order in which the content should be presented. The task analysis includes some models of expert performance.

9.8.1 Content Analysis

The content model for CS 235 is part of the course that the computer science department wants few if any changes to because of how it might affect other courses in the program. Thus, the content model matches the content model of the other two implementations.

The content can be divided into the major content areas of data structures, recursion, runtime complexity, object-oriented programming, C++ basics, and professional skills. However, much of the content applies to more than one main category. Trees, for example, are a data structure, require recursion for depth-first traversal, can have different runtime complexity based on certain attributes, and are implemented using object-oriented programming. Thus the concept map (cf. Fig. 9.13) is less like a tree structure and more like a dense graph.
Figure 9.13. A concept map showing topics covered in CS 235 and how they interrelate.

**9.8.2 Task Analysis**

One of the most important and practical learning outcomes for this course is the first learning outcome: Use the fundamental data types (lists, stacks, queues, priority queues, sets, maps, trees, etc.) to solve computational problems. Most of the students who take this course will never have to implement their own AVL tree, and many will only seldom if ever use recursion. Most (if
not all) students will later be in situations where they are presented with a problem to solve and must know how to choose the best data structure for the given problem. Expert coders can quickly choose the best data structure for a problem without having to think much about it, because the process becomes tacit with practice. I developed a simple decision tree to model expert performance for choosing a data structure (cf. Fig. 9.14).
Figure 9.14. A decision tree modelling expert behavior for choosing a data structure.

Another important learning outcome is the ability to choose the best sorting algorithm for a given problem. I created a table (cf. Table 9.15) that includes the best-, average-, and worst-case Big O efficiency ratings for common sorting algorithms. For algorithms that behave differently depending on the initial shape of the data being sorted, I included a brief explanation of how the data must be shaped to achieve the best-, average-, or worst-case. Experts use this knowledge to determine which sorting algorithm to use based on the data being sorted.

<table>
<thead>
<tr>
<th>Sorting Algorithm</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>O(n)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td></td>
<td>When elements are mostly sorted</td>
<td>For the average case, bubble sort is the least efficient of these sorting algorithms in practice</td>
<td>When elements are in reverse order</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>O(n)</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td></td>
<td>When elements are mostly sorted</td>
<td></td>
<td>When elements are in reverse order</td>
</tr>
<tr>
<td>Mergesort</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Heapsort</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td></td>
<td>When elements are mostly sorted, heap is built in O(n) but still takes O(n log n) to remove elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quicksort</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td></td>
<td>For the average case, quicksort is the most efficient of these sorting algorithms in practice</td>
<td></td>
<td>When pivots are highest or lowest values in selected portion, behaves like selection sort, very rare</td>
</tr>
</tbody>
</table>

Table 9.15. Common sorting algorithms with their best-, average-, and worst-case efficiency ratings and explanations of conditions.

9.8.3 Main Implications

The interrelatedness of the various topics makes it difficult to prescribe order to the content areas. It makes sense to start the course with C++ basics, professional skills, and object-oriented
programming, but the other content can be taught in any order. Runtime complexity, sorting, searching, recursion, and non-linear data structures are all so interrelated and interdependent that they almost have to be taught in parallel rather than in sequence. However, learning materials have to be assigned some sequential order. Given that students and instructors report that recursion is one of the more difficult concepts to grasp for CS 235 students, introducing recursion earlier in the course may be beneficial. Starting there, one possible ordering of content could be recursion, searching, runtime complexity, sorting, then non-linear data structures.

9.9 Annotated Bibliography

9.9.1 Domain Knowledge

This is a reference book for UML, a system of diagramming relationships in object-oriented design. UML can be used to visualize classes, public and private methods, inheritance, composition, aggregation, and general association. It is often used as a starting point in the process of translating a problem from natural language to a programmatic solution. I used UML in my live coding sessions to show students how to decompose problems and organize solutions, and I used this text as a reference.

This is the textbook that was used by both versions of CS 235 and was optional for the section I taught. It includes all of the major content from the course including C++ basics, algorithmic complexity, object-oriented design, elementary data structures, recursion, trees, heaps, hash tables, and sorting. This was my primary source of domain knowledge.

This is an excellent resource for learning C++ basics and general object-oriented design principles. CS 235 requires students to use C++ as a programming language, otherwise this resource would not be appropriate for this course because it does not cover much of the content. I used it primarily to look up C++ syntax and best practices as I designed live coding exercises.

Even though the code samples in this textbook are in Java rather than C++, this book is an excellent resource for the content covered in CS 235. It covers object-oriented design principles, algorithm analysis, recursion, sorting algorithms, elementary data structures, trees, hash tables, heaps, and more advanced structures. I used it as a resource to gain deeper knowledge of these important coding paradigms so that I can better design reading assignments and live coding exercises.

There is a lot of overlap in the content that these resources cover, but there are no major contradictions or differences that I need to resolve. All of these texts combined cover all of the content for the course with the exception of specific 3rd-party technologies such as Git, Cloud9,
and Valgrind. Previous instructors have designed text-based resources for these technologies that I used to help students get started.

9.9.2 Learning Theories and Instructional Strategies


This literature review covers what the field knows about pedagogy and assessment in computing, providing a comprehensive review on effective computing pedagogy. It discusses a wide range of problems and solutions such as apprentice-based learning, functional programming first, problem-based learning, the use of online tutorials, object-oriented programming, environments to introduce programming, pair programming, model-driven programming (as opposed to the prevailing language-driven approach), teaching software engineering, testing, extreme programming, frameworks, feedback and assessment, active learning, technology-based individual feedback, and mini project programming exams. This resource provides several ideas on learning theories and instructional strategies that I can use as I design the course.


In a study looking at the understanding of abstractions when programming, the authors created an Abstraction Transition (AT) taxonomy. They described the taxonomy as having three main levels: code, CS speak, and English. Using it explicitly can help students transition between reading English problem descriptions and developing coding solutions. I used this technique in class as I did live coding to give students a better idea of how to break down problems and design solutions.


Hundreds of educational psychology studies have demonstrated that spacing out repeated encounters with content over time produces superior long-term learning compared with repetitions that are massed together. I dispersed homework problems throughout reading assignments and repeated material over time to promote better long-term learning.


Use subgoal modelling, where meaningful labels are added to worked examples to visually group steps into subgoals to highlight the structure of the code. Students given subgoals perform significantly better than those who have no subgoals or make their own. Subgoals and labels reduce cognitive load and provide a way for novices to organize the problem in memory. I can break down coding labs into subgoals and organize my code during live coding sessions according to subgoals.

A comparison was made between university students reviewing static code versus their instructor modelling live coding. For the end of term projects, students’ grades were significantly higher if they were part of the live coding version of the course. In midterm assessments, the live coding cohort performed as well as the static code group. I used live coding to help walk students through the problem solving process.


Peer instruction has been used to increase engagement in lectures. Having students answer multiple-choice questions on a device, talking amongst themselves, then answering the question again increases engagement and student understanding. Students reported it as valuable to their learning and instructors reported that it helped students share their own misconceptions. Average correctness after discussion was 63-68% with a normalized gain of 35-41%. I used this teaching strategy in lectures to help students help each other with the help of free online think-pair-share products such as Kahoot or Socrative.


This literature review summarizes what is known about pedagogies for teaching computing in schools. The overall intention is to identify what recent clear evidence is emerging from research. The author includes ideas such as pair programming, gamemaking pedagogy, unplugged pedagogy, physical computing pedagogy, the use-modify-create strategy, code reading, code tracing, code annotation, live coding, think aloud techniques, learning templates, etc. Instruction should use a constructionism-based authentic problem-solving approach with information processing, scaffolding, and reflection activities. Before coding, teach high-level problem solving skills such as abstracting away details, clearly stating problems, reformulating problems, and avoiding implementation constraints. This resource provides several ideas on learning theories and instructional strategies that I can use as I design the course.

There is general consensus in the literature that live coding, code tracing, think aloud techniques (cognitive apprenticeship), subgoal modelling (problem decomposition), social learning, and teaching problem solving strategies are some of the best instructional strategies to use for teaching computer science skills in early higher education. Live coding was my primary technique for this course, but I also used code tracing, think aloud techniques, subgoal modelling, and think-pair-share activities.

There are many differences in opinion on what type of programming language to use: imperative, object-oriented, functional, or graphical (model-driven). Among the types of programming languages, there are even more differences in opinion on which specific language is best to use
For this course I am constrained to use C++ (an imperative and object-oriented language), but without that constraint it would be difficult to select a programming language due to the differences of opinion found in the literature.

### 9.9.3 Design Approaches


This book breaks down parts of a design into modular layers, where each layer poses unique design questions. Instructional design can be divided into the content layer, strategy layer, message layer, control layer, representation layer, data management layer, and media-logic layer. I aimed to keep my design strategy-centric to give more flexibility to the specific messages and media that can be replaced over time.


This book chapter highlights the importance of observing users as they act within a specific context. Observing people can generate inspiration for design as you notice challenges, difficulties, or frustrations that people have in that context. Asking people what they want can only be useful if people already know what they want without having to think about it or see it, but people often do not realize they want something until they see it. Thus observing can be more powerful than surveying. I observed student behavior as I taught and occasionally in the TA labs to get inspiration for how these students could be better served.


This book chapter emphasizes asking the right questions during design. Instead of asking questions about what to teach, for example, ask questions about what the students’ goals are and what role the course can play in helping them to attain those goals. What obstacles exist that cause students to fail? How can students better learn from the mistakes of others? How can we better understand the students’ goals? I used this resource as a repository of good questions to ask as I did design work.

Sims, R. (2014). What is the design alchemy pedagogy? In *Design alchemy: Transforming the way we think about learning and teaching* (pp. 121-160). Cham, Switzerland: Springer International Publishing. doi: 10.1007/978-3-319-02423-3

The Design Alchemy pedagogy is a learner-centered paradigm where the course is considered from the perspective of the student. The elements of the paradigm include being inclusive, being active, solving problems, providing context, working collaboratively, being creative, and enabling emergence. One of the beliefs of the pedagogy is that instructors should not explicitly teach critical thinking, but should rather design activities that require students to do critical thinking. Students are capable and do not need the instructor and course materials to provide everything for them; they just need to be presented with well-designed activities that help them develop problem-solving skills on their own.

This article summarizes what activities designers commonly perform for each step of the ADDIE model. It also explores certain design paradigms, including the pragmatic paradigm. The pragmatic paradigm suggests that a design is definitively good if and only if it has proved practical and effective for users in a specific context. Pragmatic designs are generally iterative, created through a process of quickly building, testing, and revising prototypes. For this project I am constrained because I can only teach the course once, but over the course of the semester I can try, test, and revise prototypes to get a feel for what students prefer and what is most effective.

The general consensus among these resources is that the learner’s perspective should be strongly considered for every design decision made. It is important to observe students, understand their goals, and empower them to construct knowledge for themselves. Asking the right kinds of design questions can help designers to get the answers they really need to better understand and empathize with learners.

### 9.10 Budget and Timeline

I am not being paid for this project, but if I were being paid I would expect roughly $2,500 for course design and evaluation and another $2,500 for teaching a section, summing up to roughly $5,000 total.

The timeline is best communicated via Gantt chart (cf. Fig. 9.16). The project is relatively analysis-heavy due to the computer science department’s request for the front-end evaluation of the two courses. Because of that and other time constraints, most of the design and development were completed during and after the implementation of the course. The first four weeks of the course were designed and developed beforehand, and then design work continued concurrently with implementation, ongoing analysis, and evaluation. There were design and development changes that I would have liked to implement but did not have time to do so, so I scheduled time after the implementation to make those changes retroactively. Although this timeline does not reflect my initial intentions, the benefit of it is that some of my design decisions were informed by my implementation and evaluation, which I believe resulted in a better final product for the client.
Figure 9.16. Gantt chart of project timeline. Click here to view full size.
9.11 Design Specifications

For this section of CS 235, I focused my attention on designing classroom activities, reading assignments, and exams. Classroom activities and reading assignment design are outlined here; exams are outlined in the Assessment Plan section of this document.

9.11.1 Primary Deliverables

Designing and teaching an entire sophomore-level computer science course requires a lot. Here is a high-level overview of my primary deliverables:

1. new interactive reading assignments to replace textbook,
2. new exams to better measure higher-level learning outcomes,
3. live coding activities to do in class to model expert performance,
4. a simple web-based think-pair-share implementation,
5. a new lab in which students use both hash tables and heapsort to calculate word frequencies in a text file and print out a list of each word with its frequency in sorted order
6. reworded learning outcomes that are more precise and clear, and
7. the integration of existing web technologies such as Canvas, Cloud9 IDE, Campuswire, Google Docs, GitHub, etc.

9.11.2 Course Overview

The course overview outlines the discussion topics for each week as well as the due dates of labs, small assignments, and exams (cf. Fig. 9.17). It also provides a suggested time span for working on each lab. Where the time span for one lab ends, the time span for the next lab begins, showing the students that they should always be working on a lab throughout the course.

Doing 9 labs over the course of 14 weeks allows for a rough average of 1.5 weeks per lab. Lab 2 is easier because students are provided with the pseudocode and merely need to translate it into working code, so I only allotted one week to lab 2. According to past scores, labs 7 and 8 are generally the most difficult, so I allotted a full two weeks to lab 7 and nearly three weeks to lab 8 (recognizing that lab 8 overlaps with Thanksgiving).

Generally two to three reading assignments/quizzes (marked with "(R)") are assigned each week. These are meant to be small assignments (10-20 minutes) to keep students accountable for their learning outside of class. Reading assignments generally relate to either the current lab or the following lab and should thus help the students to better understand the labs.

There is nothing to prevent students from going faster should they choose to do so. The exams have specific date ranges, but assignments and labs can be done as early as students like, allowing more experienced students to go at a faster pace.

For a live version of the course overview, see the Course Schedule document.
<table>
<thead>
<tr>
<th>Week Number</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>9/5</td>
<td>9/7</td>
</tr>
<tr>
<td></td>
<td>9/10</td>
<td>9/12</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Refresh</td>
<td>Lab 1</td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>Introduction to Algorithms</td>
<td>C++ Refresh Activity</td>
</tr>
<tr>
<td></td>
<td>Syntax</td>
<td>C++ Review Activity</td>
</tr>
<tr>
<td></td>
<td>Resources</td>
<td>C++ Objects</td>
</tr>
<tr>
<td></td>
<td>C++ Review Activity</td>
<td>UML</td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>Start of Semester Survey</td>
<td>Pinewood Derby</td>
</tr>
<tr>
<td></td>
<td>Refresh Activity</td>
<td>Pinewood Derby</td>
</tr>
<tr>
<td></td>
<td>(R) C++ Objects</td>
<td>(R) Inheritance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week Number</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>9/17</td>
<td>9/19</td>
</tr>
<tr>
<td></td>
<td>9/21</td>
<td>9/24</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Lab 2</td>
<td>Lab 3</td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>Expressions</td>
<td>Container Overview</td>
</tr>
<tr>
<td></td>
<td>Stack Algorithms</td>
<td>Stacks Data Queues</td>
</tr>
<tr>
<td></td>
<td>Container Performance</td>
<td>Lists Queue</td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>(R) Stacks</td>
<td>Lab 1</td>
</tr>
<tr>
<td></td>
<td>Lab 2</td>
<td>(R) Stacks, Lists, and Queues</td>
</tr>
<tr>
<td></td>
<td>Lab 2</td>
<td>(R) Maps</td>
</tr>
<tr>
<td></td>
<td>Lab 2</td>
<td>(R) Basic Data Structures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week Number</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>10/1</td>
<td>10/3</td>
</tr>
<tr>
<td></td>
<td>10/5</td>
<td>10/8</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Lab 4</td>
<td>Lab 5</td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>Recursion</td>
<td>Recursive Search</td>
</tr>
<tr>
<td></td>
<td>Recursion</td>
<td>Recursive Search</td>
</tr>
<tr>
<td></td>
<td>Big-O</td>
<td>Logarithms</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>Big-O</td>
</tr>
<tr>
<td></td>
<td>Searching</td>
<td></td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>(R) Recursion Survey #2</td>
<td>Lab 3</td>
</tr>
<tr>
<td></td>
<td>(R) Recursive Search</td>
<td>(R) Sorting</td>
</tr>
<tr>
<td></td>
<td>(R) Big-O</td>
<td>(R) Selection Sort</td>
</tr>
<tr>
<td></td>
<td>(R) Merge Sort</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week Number</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>10/15</td>
<td>10/17</td>
</tr>
<tr>
<td></td>
<td>10/19</td>
<td>10/22</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Lab 5</td>
<td></td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>Quicksort</td>
<td>Midterm Review</td>
</tr>
<tr>
<td></td>
<td>Quicksort</td>
<td>Quicksort</td>
</tr>
<tr>
<td></td>
<td>Valgrind</td>
<td>Valgrind</td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>Lab 4</td>
<td>(R) Quicksort</td>
</tr>
<tr>
<td></td>
<td>Midterm</td>
<td>(R) Exceptions</td>
</tr>
<tr>
<td></td>
<td>Lab 5</td>
<td>(R) Valgrind</td>
</tr>
<tr>
<td></td>
<td>(R) Linked List</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week Number</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>10/29</td>
<td>10/31</td>
</tr>
<tr>
<td></td>
<td>11/2</td>
<td>11/5</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Lab 6</td>
<td>Lab 7</td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>Linked List Interview Questions</td>
<td>Binary Tree</td>
</tr>
<tr>
<td></td>
<td>BT Traversal</td>
<td>BT Traversal</td>
</tr>
<tr>
<td></td>
<td>Binary Search</td>
<td>BST Basics</td>
</tr>
<tr>
<td></td>
<td>Modifying BST</td>
<td>AVL</td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>(R) Templates</td>
<td>Lab 6</td>
</tr>
<tr>
<td></td>
<td>(R) Binary Tree</td>
<td>(R) BT Traversal</td>
</tr>
<tr>
<td></td>
<td>Survey #3</td>
<td>(R) BT Traversal</td>
</tr>
<tr>
<td></td>
<td>(R) BST Basics</td>
<td>(R) Modifying BTs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week Number</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>11/12</td>
<td>11/14</td>
</tr>
<tr>
<td></td>
<td>11/16</td>
<td>11/19</td>
</tr>
<tr>
<td><strong>Lab</strong></td>
<td>Lab 8</td>
<td></td>
</tr>
<tr>
<td><strong>Discussion Topics</strong></td>
<td>AVL Rotations</td>
<td>AVL</td>
</tr>
<tr>
<td></td>
<td>AVL</td>
<td>Lab 8</td>
</tr>
<tr>
<td><strong>Due (before midnight)</strong></td>
<td>(R) AVL Rotation</td>
<td>(R) AVL Homework</td>
</tr>
</tbody>
</table>

| **Lab**     | Lab 7 |
9.11.3 Classroom Activities

A big part of what the computer science department wants me to do is to design new (and make modifications to existing) classroom activities to promote deeper learning. Following the flipped classroom format, classroom activities were more focused on hands-on learning and working on coding labs than on lecturing. Classroom activities should be designed to promote deeper learning and should be able to transfer seamlessly into an online course in order to meet the goals of the computer science department. The bulk of class time consisted of live coding activities (including code tracing), think-pair-share activities to generate student discussion, and giving students time to work together on homework and labs.

9.11.3.1 Live Coding Activities

Much of class time was dedicated to live coding activities where I demonstrated how to use concepts that students needed to use for the labs. These live coding activities need to relate to the labs enough so that students can understand how to apply the concepts to the labs but avoid being so close to the labs that students can code the labs without really understanding the concepts. There are already some live coding activities that have been used in the flipped version of the course that I modified and used. I also designed additional live coding activities.

The benefits of live coding are well defended in modern literature. Live coding can be used to demonstrate subgoal modelling (Margulieux & Catrambone, 2016), to model expert performance (Cutts et al. 2012), and to give tacit mental processes more transparency (Rubin, 2013). Rubin found that students who are taught with live coding perform significantly better on coding labs than students who are only given static code to review. Sometimes instructors make errors in their code when doing live coding, but this provides an opportunity for instructors to walk students through the mental processes of finding and fixing errors in code.
9.11.3.2 Think-Pair-Share Activities

I had been hoping to use a free online tool like Socrative or Kahoot to ask questions to students, collect feedback, have them discuss with a neighbor, and resubmit an answer. Simon, Kohanfars, Lee, Tamayo, and Cutts (2010) found that having computer science students answer multiple-choice questions on a device, talk amongst themselves, then answer the question again can increase engagement and student understanding. However, I learned that the free version of Socrative has a limit of 50 students. Kahoot could work, but it does not provide a convenient way of asking the same question twice and storing both result sets. So I developed my own online solution that is specifically designed for these think-pair-share activities. The student view consists of a keypad with answers A-F, much like an iClicker. The administrator view is only accessible by me, and I can set the current round, see the response count, toggle the view of the response graph, and reset the system. Questions and multiple choice options were communicated on a slide or on the whiteboard. (See the design representations/prototypes section for more details.)

9.11.3.3 Working with Peers

Remaining classroom time was given to students to work together on homework and labs. This allowed students to do work in an environment where they can get help from the instructor and form communities of learning with other classmates.

9.11.4 Reading Assignments

My conclusions from the evaluation of the flipped version of the course and from my literature review lead me to believe that adding small reading assignments is a good way to keep students accountable for their learning and to help them pace themselves.

Coding problems and practice questions should be spaced throughout the text so that learners can apply principles and check understanding as they go. Reading assignments were accessible throughout the semester and could be taken multiple times, which allowed students to keep trying until they understand everything. Practice questions were objectively scored by the LMS so that students got immediate feedback (Hattie, J. & Timperley, H., 2007) and could try again as soon as they so desired.

9.12 Design Representations/Prototypes

9.12.1 Live Coding Activities

Recursion is one of the more advanced and difficult topics within the course. The first recursion lab has the students use recursive backtracking to find a way through any solvable 3D maze. In the past, some instructors have provided code for finding a way through any 2D maze and then ask the students to figure out how to add the logic for the third dimension. Instead of doing something that is so close to the actual lab, I did a live coding activity to solve another recursive backtracking problem, such as a Boggle solver. Finding every word on a Boggle board using recursive backtracking is actually very similar to finding a way through a 2D maze, but the
connection isn’t as obvious for beginners. Students had to have slightly better understanding of recursion to see the connection, which I pushed them in a necessary way to help them learn recursion. A Boggle solver also needs to use binary search algorithms, which we covered in the same unit. I developed a Boggle solver outline in C++ and made it available through GitHub. Here are the contents of the header file that we wrote implementations for as a class:

```cpp
#pragma once
#include <iostream>
#include <sstream>
#include <fstream>
#include <algorithm>
#include <unordered_set>
#include <set>
#include <vector>
define SIZE 5
using namespace std;

class Boggle
{
private:
    string board[SIZE][SIZE]; // A 2-dimensional array of strings representing the boggle board
    vector<string> dictionary; // A collection of dictionary words in alphabetical order
    int minWordLength; // The minimum word length of words to find in boggle

    /*
    * The recursive function to find words on the boggle board
    * @param root - the current word that you’re building as you visit positions on the board
    * @param path - a collection of positions that you have visited while building your current word
    * @param wordsFound - a collection of all words that have been found so far
    * @param x - the x-coordinate in the position to visit
    * @param y - the y-coordinate in the position to visit
    */
    void findWords(string root, unordered_set<string> path, set<string>& wordsFound, int x, int y);

    /*
    * Clear the dictionary
    * Reset minWordLength to 3
    * Set all positions on the board to "a"
    */
    void clear();

    /*
    * Copy the values passed in to the board array
    */
    void setBoard(string boardValues[SIZE][SIZE]);

    /*
    * Is the root a valid prefix, a valid word, or neither?
    * @param root - the string to test if it is a valid word/prefix in the dictionary
    * @param start - the lower-bound index of the dictionary to check
    * @param end - the higher-bound index of the dictionary to check
    * @param fullWordOnly - if this is true, then only return true if root is a word in the dictionary
    * @return - return true if root is found to be a valid word or prefix in the dictionary.
    */
```
In the case that fullWordOnly is true, then return false if it is not a valid word in the dictionary

```cpp
bool findPrefix(string root, int start, int end, bool fullWordOnly);
```

```cpp
public:
    /*
    * Initialize everything (use the clear function)
    */
    Boggle();

    // Destructor
    ~Boggle();

    /*
    * Get a string representation of the board
    * Every cell should be separated by a space
    * No space at the end of each line
    * No extra line at the end of the string
    */
    string getBoardString();

    /*
    * Generate a random board and store it in the board array
    */
    void createRandomBoard();

    /*
    * Read in a board from a file and store it in the board array
    * If for some reason the import fails, then don't overwrite current board
    * If the import is successful, use the setBoard function to store to board array
    */
    bool importBoard(string file_name);

    /*
    * Read in the dictionary
    * @param file_name - dictionary file
    * This file should have one word per row and should already be sorted in alphabetical order
    */
    bool importDictionary(string file_name);

    /*
    * Set minWordLength
    * This value should not impact the values in the dictionary, just the values returned by solveBoard
    */
    void setMinWordLength(int length);

    /*
    * Return true if word is found in the dictionary
    * Use findPrefix and set fullWordOnly to true
    */
    bool isWord(string word);

    /*
    * Return true if prefix is found at the start of any word in the dictionary
    * Use findPrefix and set fullWordOnly to false
    */
    bool isPrefix(string prefix);

    /*
    * Return a collection of words found on the boggle board
    */
```
9.12.2 Think-Pair-Share Implementation and Examples

My think-pair-share implementation is simple and effective, allowing students to answer from any device without having to log in (cf. Fig. 9.18). The admin view can show a graph of the answers in real time or can hide the graph until the question is closed (cf. Fig. 9.19).

![Figure 9.18. Screenshot of the student view of my think-pair-share implementation.](image)
Figure 9.19. Screenshot of the administrative view of my think-pair-share implementation.

Here is an example of a think-pair-share question I used:

```c
// Stack S1:
S1.push(4);
S1.push(2);
S1.push(4);
S1.push(8);
pop(S3);

// Stack S2:
Stack S3;
while (!S1.empty()) {
    S2.push(S1.top());
    S1.pop();
    S3.push(S1.top());
    S1.pop();
}
pop(S2);
pop(S3);

while (!S2.empty()) {
    S3.push(S2.top());
    S2.pop();
}
pop(S3);
```

What does this code print out? Assume that the top of the stack is printed out first and is printed when the stack is empty.

- A. 4248 42 48
- B. 4248 44 28
- C. 8424 44 28
- D. 8424 28 44
- E. 4248 28 44
- F. 8424 42 48

9.12.3 Reading Assignments

See Inheritance Reading and Quiz for an example reading assignment with quiz questions.
9.13 Intellectual Property Ownership

In working with the CS department to establish intellectual property ownership of the course materials that I designed and developed, it was determined that I as the student developer hold the intellectual property rights per section IV.E. of the Brigham Young University Intellectual Property Policy. I was not paid by the university for this project and I only nominally used university resources, so the university relinquishes ownership of creative works and rights to me as the developer.

9.14 Assessment Reports and Instruments

For my CS 235 course, assessments fall into three categories: coding labs, exams, and small assignments. Coding labs measure a student’s ability to apply knowledge and skills to create code solutions to common computing problems, exams measure a student’s understanding and ability to apply the underlying theory, and small assignments help keep students accountable as they prepare for labs and exams.

9.14.1 Labs

The labs fulfill both formative and summative purposes: Students learn by doing and produce an artifact that communicates their level of competency to the instructor. Students achieve many of the apply- and create-level learning outcomes through doing the labs.

Per the request of the computer science department, I made minimal changes to the labs. I used eight of the nine labs that Dr. Clement used in the flipped version of the course taught Winter 2018 (see the Consulting Precedent section for a list of the labs used in Winter). In some cases I modified the rubrics and/or instructional materials provided to help students code the lab. I made significant changes to the ninth lab because the previous version did not have an autograder and instead required students to grade each other’s which violates FERPA, takes up class time, and reduces student autonomy. Refer to Table 9.20 for a list of the labs I used and the learning outcomes they are designed to assess.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Learning Outcomes</th>
</tr>
</thead>
</table>
| 1 | RPG | Lab Outcomes:  
- Write a complete C++ solution.  
- Use basic I/O manipulators.  
- Design classes to store and access private data members.  
- Use inheritance.  
- Use polymorphism.  
- Overload functions.  
- Use objects to manage resources.  

Course Outcomes:  
- Use language-specific best practices to write and |
<table>
<thead>
<tr>
<th>Course</th>
<th>Lab Outcomes</th>
<th>Course Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Expressions</td>
<td>Use data structures from language-provided libraries to solve problems.</td>
</tr>
<tr>
<td></td>
<td>Lab Outcomes:</td>
<td>Understand the benefits and limitations of various data types.</td>
</tr>
<tr>
<td></td>
<td>● Use an STL stack in parsing expressions.</td>
<td>Use language-specific best practices to write and organize code.</td>
</tr>
<tr>
<td></td>
<td>● Convert expressions from infix to postfix notation.</td>
<td>Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.</td>
</tr>
<tr>
<td></td>
<td>● Convert expressions from infix to prefix notation</td>
<td>My Changes:</td>
</tr>
<tr>
<td></td>
<td>● Evaluate postfix expressions using an operator and operand stack.</td>
<td>● I modified the pseudocode provided to students to make the algorithms more specific and clear.</td>
</tr>
<tr>
<td>3</td>
<td>Maps</td>
<td>Use STl lists, sets, and maps.</td>
</tr>
<tr>
<td></td>
<td>Lab Outcomes:</td>
<td>Use file I/O to read in text documents and write text to files.</td>
</tr>
<tr>
<td></td>
<td>● Implement a simple machine learning algorithm that reads in homogeneous text and generates related text.</td>
<td>Solve a basic machine learning problem.</td>
</tr>
<tr>
<td></td>
<td>Lab Outcomes:</td>
<td>Use data structures from language-provided libraries to solve problems.</td>
</tr>
<tr>
<td></td>
<td>● Use STL lists, sets, and maps</td>
<td>Understand the benefits and limitations of various data types.</td>
</tr>
<tr>
<td></td>
<td>● Use file I/O to read in text documents and write text to files</td>
<td>Use language-specific best practices to write and organize code.</td>
</tr>
<tr>
<td></td>
<td>● Solve a basic machine learning problem</td>
<td>Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.</td>
</tr>
<tr>
<td>4</td>
<td>Maze</td>
<td>Lab Outcomes:</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>Two SCUBA divers need to find a path through the storage facility maze.</td>
<td>- Use recursion to solve complex problems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Take a real world problem and create an algorithm to solve it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use recursive backtracking to solve a problem.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Dynamically allocate a multidimensional array.</td>
<td></td>
</tr>
<tr>
<td>Course Outcomes:</td>
<td>- Understand how recursion works and the best practices of designing recursive algorithms.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Understand the benefits and limitations of recursion.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use recursive backtracking to create solutions to computational problems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use language-specific best practices to write and organize code.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.</td>
<td></td>
</tr>
<tr>
<td>My Changes:</td>
<td>- In addition to providing the PathfinderInterface header file, I provided a Pathfinder header file to define a Pathfinder class to inherit from PathfinderInterface. By providing some of these additional function signatures, I gave students a better idea of how they should go about solving problems using recursive backtracking.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- I modified the provided 2D maze solution to match a more typical pattern of using recursive backtracking.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Quicksort</td>
<td>Lab Outcomes:</td>
</tr>
<tr>
<td>Implement a quicksort algorithm and sort a data set.</td>
<td>- Understand and explain the Quicksort Algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Use recursion to quickly sort data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Properly manage memory in arrays.</td>
<td></td>
</tr>
<tr>
<td>Course Outcomes:</td>
<td>- Understand how recursion works and the best practices of designing recursive algorithms.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Understand the benefits and limitations of recursion.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Apply knowledge of algorithmic complexity to make informed decisions about which sorting and searching algorithms to use to solve a given problem.</td>
<td></td>
</tr>
</tbody>
</table>
|  | - Use language-specific best practices to write and
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
|   | organize code.  
|   | - Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.  
| My Changes: | None |

| 6 | Linked List  
|   | Develop a Linked List that performs several standard operations.  
| Lab Outcomes: |  
|   | - Develop a Linked List that performs several standard operations.  
|   | - Develop a C++ solution with the use of templates.  
|   | - Handle memory management in C++.  
| Course Outcomes: |  
|   | - Understand the implementation details of all the fundamental data types.  
|   | - Create implementations of some fundamental data types.  
|   | - Use language-specific best practices to write and organize code.  
|   | - Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.  
| My Changes: |  
|   | I allowed students to choose whether or not to use a tail pointer in their doubly-linked list implementations (instead of mandating that they not use a tail pointer). |

| 7 | BST  
|   | Add and remove nodes from a binary search tree.  
| Lab Outcomes: |  
|   | - Develop a Binary Search Tree data structure that performs several standard operations using recursion.  
|   | - Use pointer references as parameters to function calls to modify pointers in the tree.  
|   | - Handle memory management in C++.  
|   | - Prepare for upcoming AVL lab.  
| Course Outcomes: |  
|   | - Understand the implementation details of all the fundamental data types.  
|   | - Create implementations of some fundamental data types.  
<p>|   | - Understand how recursion works and the best practices of designing recursive algorithms. |</p>
<table>
<thead>
<tr>
<th>8</th>
<th>AVL Trees</th>
<th>Use AVL algorithm to self-balance a binary search tree.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Hash Maps</td>
<td>Create a hashmap mapping strings to ints to count word</td>
</tr>
</tbody>
</table>

- Understand the benefits and limitations of recursion.
- Use language-specific best practices to write and organize code.
- Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.

My Changes:
- None

Lab Outcomes:
- Determine if a Binary Search Tree is critically imbalanced and distinguish between the various types of imbalance.
- Implement functions to rotate nodes and balance a Binary Search Tree.
- Insert and remove nodes from a Binary Search Tree while maintaining tree balance.

Course Outcomes:
- Understand the implementation details of all the fundamental data types.
- Create implementations of some fundamental data types.
- Understand how recursion works and the best practices of designing recursive algorithms.
- Understand the benefits and limitations of recursion.
- Use language-specific best practices to write and organize code.
- Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.

My Changes:
- In class, I provided code for the add function (using a Node* & solution) and rotate functions as well as high-level pseudocode for the rebalance function and the remove function. The code for the add function helps students understand the recursive pattern for the Node* & solution. The rotate functions are just a few lines of code to help them get the pointer changes right.

Lab Outcomes:
- Implement a hashmap.
- Write a hash function that distributes values uniformly.
frequency in a text file.

- Use operator overloading for the [] operator to allow getting value by key, updating value by key, and inserting a new key/value pair.
- Use heapsort and a custom sorting function to sort words in a text file.

Course Outcomes:
- Understand the implementation details of all the fundamental data types.
- Create implementations of some fundamental data types.
- Apply knowledge of implementation details to make informed decisions about which data type(s) to use to solve a given problem.
- Use language-specific best practices to write and organize code.
- Use professional tools (IDE, debugger, text editor, compiler, version control, file management, etc.) to develop coding solutions.

My Changes:
- The bulk of this lab was written by me. I rewrote the rubric, the description/instructions, the test scripts, the provided code, etc. The only element I kept was the requirement that students implement a hashmap.

Table 9.20: The labs for this course, including descriptions and associated learning outcomes.

The rubrics break down each project into smaller chunks and provide a point amount for each chunk. The autograder runs a separate set of tests to test each chunk so that the autograder can give credit for each section of the rubric. An example rubric is included (cf. Table 9.21).

Lab 9 Rubric

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Feature Name</th>
<th>Description</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Insert and Get</td>
<td>Implement the constructor, unsigned int hash(string), void insert(string key, int value), int get(string key), int size(), and string toString(). The test driver inserts key/value pairs and get values by key. After each test file is read in and processed, the main.cpp test driver prints your toString result to show you the contents of your map.</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Insert</td>
<td>Implement bool contains(key). Return true if the</td>
<td>10</td>
</tr>
</tbody>
</table>
Contains key is in the map, false otherwise. Remember to start by using your hash function to figure out which bucket the key would be in if it was in the map.

Includes additional tests on insert to make sure that your code can handle multiple inserts into a single bucket.

| 3 | Overloading the [] operator | Overload the [] operator for inserting a new key/value pair, getting value by key, or updating value by key. The operator [] function returns an int& to allow all of these operations. Use 0 as the default value for new keys.

Includes additional tests on insert and get to make sure that insert can be used to overwrite values and that get throws an invalid_argument exception when the key isn’t in the map. Also tests to make sure that multiple keys can be chained together in a single bucket |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Remove and Clear</td>
<td>Implement bool remove(string key) and void clear(). Make sure to update all pointers and to deallocate removed nodes to avoid memory leaks.</td>
</tr>
<tr>
<td>5</td>
<td>Everything Combined</td>
<td>Intensive tests of insert, get, contains, [] operator, remove, and clear.</td>
</tr>
<tr>
<td>6</td>
<td>WordCounter and ToSortedString</td>
<td>The WordCounter class is provided for you in its entirety. Look at the code and make sure you understand it. Implement the string toSortedString() method in your hashmap to get it to work. The toSortedString() method should use heapsort (using C++ STL priority_queue) to sort nodes using the custom NodeCompare comparison function.</td>
</tr>
<tr>
<td>7</td>
<td>Valgrind</td>
<td>No memory definitely lost, directly lost, indirectly lost, or possibly lost. Run &quot;valgrind --leak-check=full /lab9&quot; to check with valgrind.</td>
</tr>
</tbody>
</table>

| Total | 100 |

*Table 9.21: The rubric for the new version of lab 9.*
Labs were turned in online and autograded. Autograding labs provides students quick feedback and allows the course to be scalable. However, there are limits to autograders. In a class like this, getting the correct output is not the only important thing; students must get the correct output in the correct way. To account for this, students also turned in their raw source code so that the TAs or instructors could verify that the code was written in the expected way. Cloud9 simplified this by allowing students to simply submit the URL for their cloud-based workspace. The graders put a verification score in Canvas for each lab.

Lab scores (both from the autograder and the TA verification scores) were saved in Canvas where I could analyze them to get an idea of how well students were doing on individual labs and across labs.

### 9.14.1.1 Lab Scores

This section contains histograms of score distributions for each lab (cf. Figs. 7.22 to 7.31). The most common score on any given lab was 100%. There were at least three 0% scores for each lab (there were three students who never turned in a lab but remained enrolled for credit). Each lab had about 5-10 scores greater than 0% and less than 100%.

**Figure 9.22.** Distribution of Lab 1 scores.

**Figure 9.23.** Distribution of Lab 2 scores.
Figure 9.24. Distribution of Lab 3 scores.

Figure 9.25. Distribution of Lab 4 scores.

Figure 9.26. Distribution of Lab 5 scores.
Figure 9.27. Distribution of Lab 6 scores.

Figure 9.28. Distribution of Lab 7 scores.

Figure 9.29. Distribution of Lab 8 scores.
9.14.2 Exams

The labs assess most of the learning outcomes, but not all of them. Most notably missing are the outcomes related to algorithm analysis. The labs also do not require students to understand the implementation details of all of the fundamental data types and sorting functions. Exams were the primary tool used to assess these learning outcomes.

I used one midterm and a final, as had been done previously in the flipped version of the course. I contemplated introducing a second midterm but decided against it for two reasons. First, I introduced more small assignments for formative and diagnostic purposes, so a second midterm was not necessary to fulfill those purposes. Second, the bulk of the learning outcomes are assessed through the labs rather than the exams, and introducing another midterm may put too much emphasis on the few learning outcomes that the exams are designed to assess.

The exams are primarily designed for summative purposes, although the midterm may also serve some diagnostic purpose. The test items were generally objectively scored, with only one performance assessment per exam where students had to write code. I used a table of specifications (included below) to design the exams to ensure that I balanced content and measured both high- and low-level learning outcomes. I also used Canvas to perform a post-assessment item analysis using the top and bottom thirds (the default in Canvas) to make sure
that items were keyed correctly and were not ambiguous or unfair. After looking at the post-
asessment data, I decided to keep exam scores unchanged. Exam questions and statistics can
be viewed in Google Drive: (View final stats). (View midterm stats).

9.14.2.1 Exam Specifications

I created a content guide and a table of specifications for both the midterm (cf. Table 9.32) and
the final (cf. Table 9.33) to balance content and learning outcomes. My primary goal was to
deemphasize sorting algorithms and shift the emphasis to application-level questions about data
structures.

9.14.2.1.1 Midterm Specifications

1. Object-Oriented Programming
   1. Classes
      1. UML
      2. "Has-a" versus "is-a" relationships
   2. Inheritance
      1. Overriding parent methods
      2. Extending parent methods

2. Data Structures
   1. Linear Structures
      1. Vector
      2. List (Linked List)
      3. Stack
      4. Queue
      5. Priority Queue
      6. Deque
   2. Other Structures
      1. Set
      2. Map
   3. What to know
      1. Benefits and limitations of each data structure
      2. Look at code and determine output
      3. Look at code and determine the values inside a data structure
      4. Given a problem to solve, choose the most appropriate data structure
      5. Using stacks to evaluate postfix expressions

3. Recursion
   1. Read and understand recursive code
   2. Look at recursive code and determine output
   3. Look at recursive code and determine the order that recursive calls are made in
      and the order of values returned
   4. Identify parts of a recursive function (base case, recursive call)
   5. Given a problem to solve, write a short recursive function to solve it
   6. Binary search

Commented [2]: +pjrich@gmail.com I downloaded the
questions/stats pages for the exams PDFs and put
them in my Google Drive project folder. You should be
able to see the questions and statistics for both exams
now.
4. Sorting
   1. Common algorithms
      1. Bubble sort
      2. Selection sort
      3. Insertion sort
      4. Merge sort
      5. Quicksort
   2. What to know
      1. Steps of each sorting algorithm
      2. Given a list of numbers and a sorting algorithm, determine the order of values in the list after each iteration of the sorting algorithm
      3. Best and worst case Big O runtime values for sorting algorithms

5. Big O
   1. Given some code, determine the Big O runtime value
   2. Given a table of input sizes and runtimes, determine the Big O runtime value

<table>
<thead>
<tr>
<th>Content</th>
<th>Know</th>
<th>Understand</th>
<th>Apply</th>
<th>Analyze/Evaluate/Create</th>
<th>Total #</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>10%</td>
</tr>
<tr>
<td>Data Structures</td>
<td>1</td>
<td>-</td>
<td>20</td>
<td>-</td>
<td>21</td>
<td>35%</td>
</tr>
<tr>
<td>Recursion</td>
<td>-</td>
<td>11</td>
<td>4</td>
<td>-</td>
<td>15</td>
<td>25%</td>
</tr>
<tr>
<td>Sorting</td>
<td>3</td>
<td>-</td>
<td>10</td>
<td>-</td>
<td>13</td>
<td>21.7%</td>
</tr>
<tr>
<td>Big O</td>
<td>-</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>8.3%</td>
</tr>
<tr>
<td>Total #</td>
<td>4</td>
<td>22</td>
<td>34</td>
<td>-</td>
<td>60</td>
<td>-</td>
</tr>
<tr>
<td>Percentage</td>
<td>6.6%</td>
<td>36.7%</td>
<td>56.7%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 9.32: Table of specifications for the new midterm.*

9.14.2.1.2 Final Specifications

1. Object-Oriented Programming
   1. Classes
      1. UML
      2. "Has-a" versus "is-a" relationships
   2. Inheritance
      1. Overriding parent methods
      2. Extending parent methods

2. Data Structures
   1. Linear Structures
      1. Vector
      2. List (Linked List)
3. Stack
4. Queue
5. Priority Queue
6. Deque

2. Other Structures
   1. Set
   2. Map
   3. Tree
      a. Binary Tree (BT)
         i. Binary Search Tree (BST)
            A. AVL Tree
         b. Binary Heap
            i. Min Heap
            ii. Max Heap
   4. Hash Table

3. What to know
   1. Benefits and limitations of each data structure
   2. Look at code and determine output
   3. Look at code and determine the values inside a data structure
   4. Given a problem to solve, choose the most appropriate data structure
   5. Underlying implementations
      a. Know how to add to, remove from, and find value in:
         i. Linked List
         b. BST
            i. Know preorder, inorder, postorder, breadth-first
               traversals
         c. AVL
         d. Binary Heap (Max or Min)
         e. Hash Table
            i. Understand requirements for good hash functions
   2. Know what data structures can be used to implement other data
      structures
      a. Set/map can be implemented by BST or Hash Table
         i. How do these different implementations of Set
            affect functions like adding, finding, and iterating?
      b. Heap can be implemented by Binary Tree or Array
         i. Understand how to go back and forth between
            binary tree and array
      c. Etc.

3. Recursion
   1. Read and understand recursive code
   2. Look at recursive code and determine output
   3. Look at recursive code and determine the order that recursive calls are made in
      and the order of values returned
   4. Identify parts of a recursive function (base case, recursive call)
   5. Given a problem to solve, write a short recursive function to solve it
6. Binary search
7. BSTs and AVL Trees

4. Sorting
   1. Common algorithms
      1. Bubble sort
      2. Selection sort
      3. Insertion sort
      4. Merge sort
      5. Quicksort
      6. Heapsort
   2. What to know
      1. Steps of each sorting algorithm
      2. Given a list of numbers and a sorting algorithm, determine the order of values in the list after each iteration of the sorting algorithm
      3. Best and worst case Big O runtime values for sorting algorithms

5. Big O
   1. Given some code, determine the Big O runtime value
   2. Given a table of input sizes and runtimes, determine the Big O runtime value
   3. Given a data structure and a function (insert, remove, find), determine the Big O runtime value

<table>
<thead>
<tr>
<th>Content</th>
<th>Know</th>
<th>Understand</th>
<th>Apply</th>
<th>Analyze/Evaluate/Create</th>
<th>Total #</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP</td>
<td>-</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>6</td>
<td>5%</td>
</tr>
<tr>
<td>Data Structures</td>
<td>6</td>
<td>24</td>
<td>30</td>
<td>-</td>
<td>60</td>
<td>50%</td>
</tr>
<tr>
<td>Recursion</td>
<td>2</td>
<td>12</td>
<td>10</td>
<td>-</td>
<td>24</td>
<td>20%</td>
</tr>
<tr>
<td>Sorting</td>
<td>11</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>10%</td>
</tr>
<tr>
<td>Big O</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>18</td>
<td>15%</td>
</tr>
<tr>
<td>Total #</td>
<td>19</td>
<td>60</td>
<td>41</td>
<td>-</td>
<td>120</td>
<td>-</td>
</tr>
<tr>
<td>Percentage</td>
<td>15.8%</td>
<td>50%</td>
<td>34.2%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 9.33 Table of specifications for the new final exam.

9.14.2.2 Exam Scores

This section contains histograms of exam scores. One student did not take the midterm and two did not take the final, resulting in some low outliers. The distributions are close to what I had expected, and they seem to suggest that the exams were neither too easy nor too difficult.
Figure 9.34. Distribution of scores for the new midterm (max points 60).

Figure 9.35. Distribution of scores for the new final exam (max points 120).
9.14.3 Small Assignments

For my version of the course, small assignments consist of small learning activities that accompany reading assignments done outside of class. Because class time largely consists of live coding and working on projects, students have to be diligent with reading assignments in order to become well acquainted with the underlying theory. Small assignments keep students accountable for their reading and fulfill a primarily formative purpose. These small assignments take the form of reading quizzes or small coding problems, and questions are spaced throughout the text so that students can practice applying as they learn. Reading quizzes are autograded by Canvas. For small coding problems, students submit their code and it is graded on completion. 85% of the students got an average score of 88% or higher on assignments (cf. Fig. 9.37).
To get a better idea of how assignment performance impacted performance on labs and exams, I created charts showing these relationships and calculated the correlation. The correlation between assignment scores and lab scores is 0.468 (cf. Fig. 9.38), and the correlation between assignment scores and exam scores is 0.684 (cf. Fig. 7.39). There is a strong positive correlation between doing well on small assignments and doing well on exams and a moderately strong correlation between doing well on small assignments and labs, suggesting that the small assignments may have helped students prepare for larger assessment items.

### Assignment Scores VS Lab Scores

![Assignment Scores VS Lab Scores](image)

*Figure 9.38. Correlation between assignment scores and lab scores.*

### Assignment Scores VS Exam Scores

![Assignment Scores VS Exam Scores](image)

*Figure 9.39. Correlation between assignment scores and exam scores.*

#### 9.14.4 Final Grades

In the lecture-based version of the course, the percentage breakdown is 50% labs, 40% exams, and 10% small assignments. In the flipped version, the breakdown is 50% labs, 30% exams, and 20% small assignments.

At least 40% of the final grade should be from the coding labs because they are the most effective measure of the higher-level learning outcomes. However, the exams are important
because they assess some learning outcomes that are not assessed by the labs. Also, the exams provide an assessment of the isolated individual, whereas labs can be completed with help from others and a wide variety of resources. At least 30% of the final grade should be from exams so that students have to do reasonably well on the exams in order to get an A in the class. The small assignments are important because they keep students accountable and serve formative and diagnostic purposes, but students should be able to skip a few small assignments without it having a major impact on their grade, so about 15% of the final grade should come from small assignments. Taking all of this into consideration, I believe the best measurement of learning outcomes can come from using a breakdown of 50% labs, 35% exams, and 15% small assignments. This breakdown resulted in the following grade distribution for my Fall 2018 section:

![Grade Distribution for My Section (n=56) - Fall 2018](image)

*Figure 9.40. Grade distribution for the 56 students who took my version of CS 235 taught in Fall 2018.*

### 9.15 Implementation Instruments

The pilot of this course started at the beginning of the Fall 2018 semester, before all of the design and development work had been completed. Thus, the experiences that I had in the beginning of the semester as the initial instructor informed my design and development work for the later parts of the course. To pilot the course, I worked with the CS department to teach it as a university course under a section labeled as an experimental section with CS 142 as a listed prerequisite.

#### 9.15.1 Required Resources

The primary resources needed for this class are (1) the Canvas course; (2) Cloud9 access for instructors, TAs, and students; (3) laptops for students; and (4) Campuswire accounts for instructors, TAs, and students. The Canvas course is essential as it contains all the reading assignments, the exams, and the autograders for the labs. Cloud9 access is needed for students to write coding solutions and for TAs and instructors to access that code, either for grading or for remote help. Students need to bring a laptop to class to follow along with live coding exercises and to work on labs. Campuswire provides a way for students to get answers to their questions.
outside of class time. Campuswire gives instructors, TAs, and other students access to answer questions and to upvote others’ answers.

Necessary resources for instructors:
- Computer with C++ programming environment (online or offline)
- Internet connection (low-bandwidth okay, high-bandwidth preferred)
- Classroom with projector (or some way of sharing instructor screen)
- Resources in Canvas (training materials, reading assignments, exams, lab specifications, etc.)
- Access to BYU-CS235-F18 GitHub account (to view/modify code samples, lab code/instructions, etc.)
  - See https://github.com/BYU-CS235-F18/
- Cloud9 instructor account (to set up C9 groups so students can set up workspaces, to help students with their code remotely, etc.)
- Campuswire instructor account (to facilitate communication with students outside of class hours)
  - See https://campuswire.com/c/GD95222BB

Necessary resources for TAs:
- Computer with C++ programming environment (online or offline)
- Internet connection (low-bandwidth okay, high-bandwidth preferred)
- Resources in Canvas (training materials, lab specifications, grading system, etc.)
- Cloud9 TA account (to access student workspaces for lab verifications)
- Campuswire TA account (to help instructor answer student questions that are asked through Campuswire)

Necessary resources for students:
- Laptop with C++ programming environment (online or offline)
- Internet connection (low-bandwidth okay, high-bandwidth preferred)
- Resources in Canvas (training materials, reading assignments, exams, lab specifications, etc.)
- Cloud9 student account
  - See https://byu.instructure.com/courses/3551/pages/programming-environment?module_item_id=225366 for related training materials
- Campuswire account
  - See https://byu.instructure.com/courses/3551/pages/campuswire?module_item_id=225381 for related training materials
  - Basic understanding of Bash shell and command-line git

One of the greatest obstacles to overcome near the beginning of the semester was helping students to learn enough Bash to work in Cloud9. I had underestimated how difficult it would be for students to pick up basic Bash, and many anxious students reached out to me about how Bash was a nontrivial barrier for them. In response to this, I developed a Bash training module in Canvas (see the module here, cf. Fig. 9.41) and devoted a lecture hour to go over it in class. In the

Commented [3]: This is a handy resource that I think I’ll use myself (not to the extent you do, of course).

Your students’ obstacle with Bash reminds me of the need for pre-training, which is discussed in e-Learning and the science of Instruction.
In this case, “mkdir” is the command and “test” is a command line argument. You can think of a command line argument as a parameter being passed to a function. You’re calling the “mkdir” function and passing “test” as a parameter. That will make a new directory with a name of “test”. You could do “mkdir base” to make a new directory with a name of base. So “mkdir” is the command, whatever follows will be the name of the new directory.

After running “mkdir test” you can run “ls -l” to see everything in your workspace directory again. And look! Now there is a new directory called “test” that exists in your workspace directory.

**Figure 9.41** Screenshot of part of the Linux/Bash training module.

### 9.15.2 My Students

At the beginning of the semester, I had my students fill out a survey (see Canvas survey for details) to give me some information such as age, gender, major, minor, and prior coding experience. The data collected provide some insight into the learners who participated in the first implementation of the course.

**Figure 9.42** Age distribution of learners.
Figure 9.43. Gender distribution of learners (not uncommon for CS courses).

Figure 9.44. Distribution of majors.
What coding experience did you have prior to taking CS 142? Select all that apply, or none if none apply.

I also asked students to use the scale below to rate their comfort level with certain programming technologies or concepts:

1 - I don’t know what this is
2 - I don’t know much about this
3 - I know a fair amount about this
4 - I could use this today if I brushed up on in for a bit
5 - I could use this effectively right now

The figures below (cf. Figs. 7.46 to 7.63) represent the aggregate data that I gathered through this initial survey.

**C++ Syntax**

<table>
<thead>
<tr>
<th>Rating</th>
<th>Number of Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25 respondents</td>
<td>64%</td>
</tr>
<tr>
<td>2</td>
<td>1 respondents</td>
<td>3%</td>
</tr>
<tr>
<td>3</td>
<td>34 respondents</td>
<td>66%</td>
</tr>
<tr>
<td>4</td>
<td>26 respondents</td>
<td>53%</td>
</tr>
<tr>
<td>5</td>
<td>17 respondents</td>
<td>44%</td>
</tr>
</tbody>
</table>
Visual Studio (or other C++ development environment)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0 %</td>
</tr>
<tr>
<td>2</td>
<td>5 respondents</td>
<td>9 %</td>
</tr>
<tr>
<td>3</td>
<td>10 respondents</td>
<td>18 %</td>
</tr>
<tr>
<td>4</td>
<td>23 respondents</td>
<td>41 %</td>
</tr>
<tr>
<td>5</td>
<td>18 respondents</td>
<td>32 %</td>
</tr>
</tbody>
</table>

Figure 9.47: Initial familiarity with C++ development environments.

Debugging tools

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>7 %</td>
</tr>
<tr>
<td>2</td>
<td>21 respondents</td>
<td>38 %</td>
</tr>
<tr>
<td>3</td>
<td>14 respondents</td>
<td>25 %</td>
</tr>
<tr>
<td>4</td>
<td>12 respondents</td>
<td>21 %</td>
</tr>
<tr>
<td>5</td>
<td>5 respondents</td>
<td>9 %</td>
</tr>
</tbody>
</table>

Figure 9.48: Initial familiarity with debugging tools.

Algebra (variables, equations, graphs, etc.)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0 %</td>
</tr>
<tr>
<td>2</td>
<td>1 respondents</td>
<td>2 %</td>
</tr>
<tr>
<td>3</td>
<td>6 respondents</td>
<td>11 %</td>
</tr>
<tr>
<td>4</td>
<td>18 respondents</td>
<td>32 %</td>
</tr>
<tr>
<td>5</td>
<td>31 respondents</td>
<td>55 %</td>
</tr>
</tbody>
</table>

Figure 9.49: Initial familiarity with algebraic principles.
### Strings (data type used to store and manipulate text)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0%</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0%</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>4 respondents</td>
<td>7%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12 respondents</td>
<td>21%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24 respondents</td>
<td>43%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16 respondents</td>
<td>29%</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.50**: Initial familiarity with string data types.

### Functions (input parameters, process, return type)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0%</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0%</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>3 respondents</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>19 respondents</td>
<td>34%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20 respondents</td>
<td>36%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14 respondents</td>
<td>25%</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.51**: Initial familiarity with functions.

### File I/O (writing code to read data from files and write data to files)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 respondents</td>
<td>5%</td>
</tr>
<tr>
<td>2</td>
<td>19 respondents</td>
<td>34%</td>
</tr>
<tr>
<td>3</td>
<td>15 respondents</td>
<td>27%</td>
</tr>
<tr>
<td>4</td>
<td>17 respondents</td>
<td>30%</td>
</tr>
<tr>
<td>5</td>
<td>2 respondents</td>
<td>4%</td>
</tr>
</tbody>
</table>

**Figure 9.52**: Initial familiarity with file input and output.
### Conditionals (if/then/else branching)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0% ✓</td>
</tr>
<tr>
<td>2</td>
<td>1 respondents</td>
<td>2%</td>
</tr>
<tr>
<td>3</td>
<td>4 respondents</td>
<td>7%</td>
</tr>
<tr>
<td>4</td>
<td>18 respondents</td>
<td>32%</td>
</tr>
<tr>
<td>5</td>
<td>33 respondents</td>
<td>59%</td>
</tr>
</tbody>
</table>

*Figure 9.53. Initial familiarity with conditionals.*

### Arrays and vectors

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2% ✓</td>
</tr>
<tr>
<td>2</td>
<td>3 respondents</td>
<td>5%</td>
</tr>
<tr>
<td>3</td>
<td>18 respondents</td>
<td>32%</td>
</tr>
<tr>
<td>4</td>
<td>23 respondents</td>
<td>41%</td>
</tr>
<tr>
<td>5</td>
<td>11 respondents</td>
<td>20%</td>
</tr>
</tbody>
</table>

*Figure 9.54. Initial familiarity with arrays and vectors.*

### Loops (for, while)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0% ✓</td>
</tr>
<tr>
<td>2</td>
<td>1 respondents</td>
<td>2%</td>
</tr>
<tr>
<td>3</td>
<td>4 respondents</td>
<td>7%</td>
</tr>
<tr>
<td>4</td>
<td>26 respondents</td>
<td>46%</td>
</tr>
<tr>
<td>5</td>
<td>25 respondents</td>
<td>45%</td>
</tr>
</tbody>
</table>

*Figure 9.55. Initial familiarity with loops.*
### Object-oriented programming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 respondents</td>
<td>7% ✔</td>
</tr>
<tr>
<td>2</td>
<td>4 respondents</td>
<td>7% ✔</td>
</tr>
<tr>
<td>3</td>
<td>23 respondents</td>
<td>41% ✔</td>
</tr>
<tr>
<td>4</td>
<td>16 respondents</td>
<td>29% ✔</td>
</tr>
<tr>
<td>5</td>
<td>9 respondents</td>
<td>16% ✔</td>
</tr>
</tbody>
</table>

**Figure 9.56.** Initial familiarity with object-oriented programming.

### Class inheritance (polymorphism)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 respondents</td>
<td>7% ✔</td>
</tr>
<tr>
<td>2</td>
<td>15 respondents</td>
<td>27% ✔</td>
</tr>
<tr>
<td>3</td>
<td>18 respondents</td>
<td>32% ✔</td>
</tr>
<tr>
<td>4</td>
<td>12 respondents</td>
<td>21% ✔</td>
</tr>
<tr>
<td>5</td>
<td>7 respondents</td>
<td>13% ✔</td>
</tr>
</tbody>
</table>

**Figure 9.57.** Initial familiarity with polymorphism.

### Recursion

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14 respondents</td>
<td>25% ✔</td>
</tr>
<tr>
<td>2</td>
<td>16 respondents</td>
<td>29% ✔</td>
</tr>
<tr>
<td>3</td>
<td>16 respondents</td>
<td>29% ✔</td>
</tr>
<tr>
<td>4</td>
<td>6 respondents</td>
<td>11% ✔</td>
</tr>
<tr>
<td>5</td>
<td>4 respondents</td>
<td>7% ✔</td>
</tr>
</tbody>
</table>

**Figure 9.58.** Initial familiarity with recursion.
Maps (data structures)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33 respondents</td>
<td>59%</td>
</tr>
<tr>
<td>2</td>
<td>17 respondents</td>
<td>30%</td>
</tr>
<tr>
<td>3</td>
<td>4 respondents</td>
<td>7%</td>
</tr>
<tr>
<td>4</td>
<td>1 respondent</td>
<td>2%</td>
</tr>
<tr>
<td>5</td>
<td>1 respondent</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 9.59. Initial familiarity with the map data structure.

Trees (data structures)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38 respondents</td>
<td>68%</td>
</tr>
<tr>
<td>2</td>
<td>12 respondents</td>
<td>21%</td>
</tr>
<tr>
<td>3</td>
<td>5 respondents</td>
<td>9%</td>
</tr>
<tr>
<td>4</td>
<td>1 respondent</td>
<td>2%</td>
</tr>
<tr>
<td>5</td>
<td>1 respondent</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 9.60. Initial familiarity with the tree data structure.

Search algorithms (bubble, selection, insert, merge, etc.)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32 respondents</td>
<td>57%</td>
</tr>
<tr>
<td>2</td>
<td>16 respondents</td>
<td>29%</td>
</tr>
<tr>
<td>3</td>
<td>6 respondents</td>
<td>11%</td>
</tr>
<tr>
<td>4</td>
<td>2 respondents</td>
<td>4%</td>
</tr>
<tr>
<td>5</td>
<td>0 respondents</td>
<td>0%</td>
</tr>
</tbody>
</table>

Figure 9.61. Initial familiarity with common search algorithms.
These results show that most students were not familiar with course content such as git, Big O, search algorithms, trees, maps, and recursion before taking the class. This is unsurprising since 44% of the class had no coding experience prior to CS 142, and CS 142 doesn’t cover these topics. I was surprised, however, by the following stats:

- 34% reported a 3 or below in their comfort level with C++ syntax (cf. Fig. 9.46).
- 70% reported a 3 or below in their comfort level with debugging tools, with 7% reporting that they didn’t know what debugging tools were (cf. Fig. 9.48).
- 28% reported a 3 or below in their comfort level with the string data type, a very basic programming concept used throughout CS 142 (cf. Fig. 9.50).
- 39% reported a 3 or below in their comfort level with functions, a basic programming concept used throughout CS 142 (cf. Fig. 9.51).
- 39% reported a 3 or below in their comfort level with arrays and vectors, a basic programming concept used throughout CS 142 (cf. Fig. 9.54).
- 55% reported a 3 or below in their comfort level with object-oriented programming, an important programming concept used in the later labs of CS 142, with 7% reporting that they didn’t know what object-oriented-programming was (cf. Fig. 9.56).
- 66% reported a 3 or below in their comfort level with polymorphism, a central concept of object-oriented programming covered in CS 142, with 7% reporting that they didn’t know what polymorphism was (cf. Fig. 9.57).
These results made me realize that even though CS 142 is a prerequisite class, there are many who attempt CS 235 without the prerequisite skills. Reviewing these prerequisite skills before covering new material is essential for these students to succeed. I modified my initial design so that I could use the first two-and-a-half weeks of class to review these prerequisites.

### 9.16 Evaluation Instruments

The CS department's goals for this course were to:

1. promote deeper learning;
2. help students transition to more proactive, self-paced, student-owned learning;
3. increase scalability to meet increased demand; and
4. encourage students to help each other (or to better help themselves)

This section reviews the instruments that I used to collect data regarding how this course measured up to these goals. I rely on student feedback, assessment results, final grade data, my experiences as the instructor, and concepts from literature to evaluate the overall course design.

#### 9.16.1 Data Collection Summary

My primary method of collecting student feedback was via surveys that were graded as small assignments. In addition to these surveys, I collected data from assessment results, final grades, my experiences as the instructor, and concepts from literature to evaluate the overall course design. Below is a summary of the type of data that I collected.

1. Surveys
   a. Start of semester
      i. Coding experience prior to CS 142 (select all that apply)
         1. Personal tinkering
         2. K8 class
         3. High school class
         4. University course
         5. None
      ii. Age
      iii. Gender
      iv. Major/minor
   v. Rate comfort levels with various concepts and technologies
      1. C++
      2. Visual Studio (or other C++ dev environment)
      3. Debugging tools
      4. Algebra
      5. Strings
      6. Functions
      7. Conditionals (if/then/else branching)
      8. Arrays
      9. Loops
      10. Vectors
      11. Object-oriented programming
      12. Inheritance
13. Recursion
14. Maps
15. Trees
16. Search algorithms
17. Big-O analysis
18. Git/GitHub

b. Throughout semester (every 4 weeks)
   i. Rate difficulty of coding labs
   ii. Rate difficulty of quizzes and small assignments
   iii. Frequency of personal class attendance
   iv. Frequency of TA lab attendance (or seeing TA/instructor for help)
   v. Rate usefulness of time in class
   vi. Rate usefulness of out of class assignments
   vii. What do you like about how class time is used?
   viii. What would you like to change about how class time is used?
   ix. What do you like about the class in general?
   x. What would you like to change about the class in general?

c. End of semester
   i. University student ratings

2. Assessments
   a. Lab scores
   b. Assignment scores
   c. Test scores
   d. Final grades

3. Popular Pedagogical Concepts from Literature Review
   a. Paired programming and communities of learning
   b. Unplugged learning
   c. Modelling levels of abstraction
   d. Game design
   e. Use-modify-create
   f. Flipped classroom
   g. Live coding
   h. Code tracing
   i. Subgoal (problem decomposition) modelling

4. Instructor Experiences

9.16.2 Surveys

The first survey was designed to give me information about my students to see where they were as programmers coming into the class. The next 3 surveys (collected at the end of September, the end of October, and the end of November) were designed to give students an opportunity to tell me about their experience in the course and the changes they would like to see implemented. Surveys contained quantitative Likert-scale items related to the difficulty of labs, difficulty of reading assignments, frequency of class attendance, frequency of TA lab attendance, usefulness of time in class, and helpfulness of course materials. Surveys also contained qualitative open-ended questions asking what students like about how class time is spent, what suggestions students have to improve how class time is spent, and general suggestions to improve the course.
9.16.2.1 September Survey Results

In summary, the live coding is something that people liked, but many thought that I as the instructor went through the code too fast. Main suggestions were to slow down, get more discussion and participation, talk more about the labs, and give more time to work on labs. A few students also reported that they felt that their main pain point was debugging, and I think there are probably many more who also felt that way.

In response to this survey, I slowed down and talked through my code more, asked more discussion questions, gave more time to talk about and work on labs, designed a debugging tutorial to help students with debugging in C++ (see https://github.com/BYU-CS235-F18/segfault), and devoted a lecture to the debugging tutorial.

Rate the difficulty of Coding Lab 1 - RPG Lab

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Easy</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Moderate</td>
<td>19</td>
<td>37</td>
</tr>
<tr>
<td>Hard</td>
<td>23</td>
<td>44</td>
</tr>
<tr>
<td>Very hard</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

*Figure 9.64. Rated difficulty of Lab 1.*

Rate the difficulty of Coding Lab 2 - Expressions Lab (with stacks)

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Easy</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Moderate</td>
<td>18</td>
<td>35</td>
</tr>
<tr>
<td>Hard</td>
<td>19</td>
<td>37</td>
</tr>
<tr>
<td>Very hard</td>
<td>6</td>
<td>15</td>
</tr>
</tbody>
</table>

*Figure 9.65. Rated difficulty of Lab 2.*
Rate the **average difficulty** of the reading assignments up to this point:

<table>
<thead>
<tr>
<th></th>
<th>Number of Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>24</td>
<td>46%</td>
</tr>
<tr>
<td>Moderate</td>
<td>26</td>
<td>50%</td>
</tr>
<tr>
<td>Hard</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

*Figure 9.66: Rated difficulty of reading assignments for first month.*

How frequently do you **attend class**?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Number of Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rarely or never</td>
<td>2</td>
<td>4%</td>
</tr>
<tr>
<td>Once a week on average</td>
<td>3</td>
<td>6%</td>
</tr>
<tr>
<td>Twice a week on average</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>Always or almost always</td>
<td>42</td>
<td>81%</td>
</tr>
</tbody>
</table>

*Figure 9.67: Class attendance frequency for first month.*

How many times have you **gone to TAs or instructors for help**?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Number of Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 times</td>
<td>12</td>
<td>23%</td>
</tr>
<tr>
<td>1-2 times</td>
<td>24</td>
<td>46%</td>
</tr>
<tr>
<td>3-4 times</td>
<td>9</td>
<td>17%</td>
</tr>
<tr>
<td>5+ times</td>
<td>7</td>
<td>13%</td>
</tr>
</tbody>
</table>

*Figure 9.68: TA help frequency for first month.*
Figure 9.69. Rated usefulness of time spent in class for first month.

Rate how helpful course materials (reading assignments, materials in Canvas) have been in helping you succeed in this course.

Figure 9.70. Rated usefulness of course materials for first month.

Table 9.71. Aggregate summary of qualitative class time likes for first month.
### Table 9.72: Aggregate summary of qualitative class time suggestions for first month.

<table>
<thead>
<tr>
<th>Suggestion</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow down</td>
<td>22</td>
</tr>
<tr>
<td>Questions/discussion</td>
<td>17</td>
</tr>
<tr>
<td>More time to work on labs</td>
<td>8</td>
</tr>
<tr>
<td>No changes</td>
<td>4</td>
</tr>
<tr>
<td>Debugging lecture</td>
<td>3</td>
</tr>
<tr>
<td>Less live coding</td>
<td>3</td>
</tr>
<tr>
<td>Too basic/not helpful</td>
<td>2</td>
</tr>
<tr>
<td>More talking about labs</td>
<td>2</td>
</tr>
<tr>
<td>Pictures/animations</td>
<td>1</td>
</tr>
<tr>
<td>Too slow, get through more</td>
<td>1</td>
</tr>
<tr>
<td>Assigned groups, force collaboration</td>
<td>1</td>
</tr>
<tr>
<td>Debrief previous labs</td>
<td>1</td>
</tr>
<tr>
<td>More reminders about due dates</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 9.73 Aggregate summary of qualitative suggestions for improvement for first month.

9.16.2.2 October Survey Results

The survey results from the end of October were much more positive. Many students recommended not changing anything.

The majority of students reported liking the course how it was, so I tried to continue doing the same things. The fact that students reported so positively was promising, and it shows that the course was enjoyable and engaging for them.
Rate the difficulty of Coding Lab 3 - Sets/Maps

<table>
<thead>
<tr>
<th>Level</th>
<th>Response</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>4</td>
<td>8%</td>
</tr>
<tr>
<td>Easy</td>
<td>17</td>
<td>35%</td>
</tr>
<tr>
<td>Moderate</td>
<td>20</td>
<td>41%</td>
</tr>
<tr>
<td>Hard</td>
<td>7</td>
<td>14%</td>
</tr>
<tr>
<td>Very hard</td>
<td>1</td>
<td>2%</td>
</tr>
</tbody>
</table>

Figure 9.74. Rated difficulty of Lab 3.

Rate the difficulty of Coding Lab 4 - 3D Maze

<table>
<thead>
<tr>
<th>Level</th>
<th>Response</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>3</td>
<td>6%</td>
</tr>
<tr>
<td>Easy</td>
<td>8</td>
<td>16%</td>
</tr>
<tr>
<td>Moderate</td>
<td>28</td>
<td>57%</td>
</tr>
<tr>
<td>Hard</td>
<td>7</td>
<td>14%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>3</td>
<td>6%</td>
</tr>
</tbody>
</table>

Figure 9.75. Rated difficulty of Lab 4.

Rate the difficulty of Coding Lab 5 - Quicksort

<table>
<thead>
<tr>
<th>Level</th>
<th>Response</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Easy</td>
<td>9</td>
<td>18%</td>
</tr>
<tr>
<td>Moderate</td>
<td>23</td>
<td>47%</td>
</tr>
<tr>
<td>Hard</td>
<td>14</td>
<td>29%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>3</td>
<td>6%</td>
</tr>
</tbody>
</table>

Figure 9.76. Rated difficulty of Lab 5.
Rate the average difficulty of the reading assignments from the last month

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>16</td>
<td>33%</td>
</tr>
<tr>
<td>Moderate</td>
<td>23</td>
<td>47%</td>
</tr>
<tr>
<td>Hard</td>
<td>9</td>
<td>18%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

*Figure 9.77: Rated difficulty of reading assignments from second month.*

How frequently did you attend class last month?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rarely or never</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>Once a week on average</td>
<td>4</td>
<td>8%</td>
</tr>
<tr>
<td>Twice a week on average</td>
<td>9</td>
<td>18%</td>
</tr>
<tr>
<td>Always or almost always</td>
<td>31</td>
<td>63%</td>
</tr>
</tbody>
</table>

*Figure 9.78: Attendance frequency for second month.*

How many times did you go to TAs or instructors for help last month?

<table>
<thead>
<tr>
<th>Times</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 times</td>
<td>18</td>
<td>37%</td>
</tr>
<tr>
<td>1-2 times</td>
<td>18</td>
<td>37%</td>
</tr>
<tr>
<td>3-4 times</td>
<td>9</td>
<td>19%</td>
</tr>
<tr>
<td>5+ times</td>
<td>4</td>
<td>8%</td>
</tr>
</tbody>
</table>

*Figure 9.79: TA help frequency for second month.*
Rate how helpful the time in class has been in helping you succeed in this course.

<table>
<thead>
<tr>
<th>Not helpful (too hard, I can’t keep up)</th>
<th>respondents</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not helpful (too basic, I don’t need it)</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>Somewhat helpful</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>Moderately helpful</td>
<td>17</td>
<td>35%</td>
</tr>
<tr>
<td>Very helpful</td>
<td>15</td>
<td>31%</td>
</tr>
</tbody>
</table>

*Figure 9.80. Rated usefulness of time spent in class for second month.*

Rate how helpful course materials (reading assignments, materials in Canvas) have been in helping you succeed in this course.

<table>
<thead>
<tr>
<th>Not helpful (too difficult to understand)</th>
<th>respondents</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not helpful (too basic)</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Somewhat helpful</td>
<td>7</td>
<td>14%</td>
</tr>
<tr>
<td>Moderately helpful</td>
<td>16</td>
<td>33%</td>
</tr>
<tr>
<td>Very helpful</td>
<td>24</td>
<td>49%</td>
</tr>
</tbody>
</table>

*Figure 9.81. Rated usefulness of course materials for second month.*

Q9 - Class time likes  

<table>
<thead>
<tr>
<th>Activity</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk about labs</td>
<td>21</td>
</tr>
<tr>
<td>Practical application/live coding</td>
<td>20</td>
</tr>
<tr>
<td>Write/draw on board</td>
<td>13</td>
</tr>
<tr>
<td>Work on labs/help queue</td>
<td>12</td>
</tr>
<tr>
<td>Reviewing reading assignments</td>
<td>5</td>
</tr>
<tr>
<td>Talk about theory</td>
<td>5</td>
</tr>
<tr>
<td>Student questions</td>
<td>4</td>
</tr>
<tr>
<td>Students write code</td>
<td>4</td>
</tr>
<tr>
<td>Attendance not mandatory</td>
<td>2</td>
</tr>
<tr>
<td>Interview questions</td>
<td>1</td>
</tr>
<tr>
<td>I can go at my own pace</td>
<td>1</td>
</tr>
<tr>
<td>Midterm review</td>
<td>1</td>
</tr>
<tr>
<td>Examples are online</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 9.82. Aggregate summary of qualitative class time likes for second month.*
<table>
<thead>
<tr>
<th>Q10 - Class time suggestions</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>No changes</td>
<td>26</td>
</tr>
<tr>
<td>Slow down</td>
<td>4</td>
</tr>
<tr>
<td>More time to work on labs</td>
<td>3</td>
</tr>
<tr>
<td>Debrief previous labs</td>
<td>3</td>
</tr>
<tr>
<td>Questions/discussion</td>
<td>2</td>
</tr>
<tr>
<td>Spend more time coding up labs near lab due date</td>
<td>2</td>
</tr>
<tr>
<td>Debugging lecture</td>
<td>1</td>
</tr>
<tr>
<td>Less live coding</td>
<td>1</td>
</tr>
<tr>
<td>Too basio not helpful</td>
<td>1</td>
</tr>
<tr>
<td>Erase whiteboard less</td>
<td>1</td>
</tr>
<tr>
<td>Clarify when other students’ questions are relevant or not</td>
<td>1</td>
</tr>
<tr>
<td>Less theory, more application</td>
<td>1</td>
</tr>
<tr>
<td>Live coding - time wasted on “filler” code</td>
<td>1</td>
</tr>
<tr>
<td>Smaller-scale class activities (can miss a day)</td>
<td>1</td>
</tr>
<tr>
<td>Spend less time coding up labs near lab due date</td>
<td>1</td>
</tr>
<tr>
<td>More discussion of practical, real-world application</td>
<td>1</td>
</tr>
<tr>
<td>Spend more time explaining new syntax</td>
<td>1</td>
</tr>
<tr>
<td>More theory, fewer programming activities</td>
<td>1</td>
</tr>
<tr>
<td>More time for questions/help on labs</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 9.83. Aggregate summary of qualitative class time suggestions for second month.
Table 9.84: Aggregate summary of qualitative suggestions for second month.

9.16.2.3 November Survey Results

This survey produced more promising results. 82% of the class reported the average difficulty of reading assignments between very easy and moderate. 92% reported that time spent in class was helping them succeed, 98% reported that the reading assignments and course materials were helping them succeed, with 50% reporting that course materials were very helpful. Many students wrote about their positive experiences in the course, and many did not recommend changes.
### Rate the difficulty of Coding Lab 6 - LinkedList

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>Moderate</td>
<td>26</td>
<td>52%</td>
</tr>
<tr>
<td>Hard</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>Very hard</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>N/A</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

*Figure 9.85. Rated difficulty of Lab 6.*

### Rate the difficulty of Coding Lab 7 - BST

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>3</td>
<td>6%</td>
</tr>
<tr>
<td>Moderate</td>
<td>16</td>
<td>32%</td>
</tr>
<tr>
<td>Hard</td>
<td>24</td>
<td>48%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>5</td>
<td>10%</td>
</tr>
<tr>
<td>N/A</td>
<td>1</td>
<td>2%</td>
</tr>
</tbody>
</table>

*Figure 9.86. Rated difficulty of Lab 7.*

### Rate the difficulty of Coding Lab 8 - AVL

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Moderate</td>
<td>10</td>
<td>20%</td>
</tr>
<tr>
<td>Hard</td>
<td>15</td>
<td>30%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>18</td>
<td>36%</td>
</tr>
<tr>
<td>N/A</td>
<td>6</td>
<td>12%</td>
</tr>
</tbody>
</table>

*Figure 9.87. Rated difficulty of Lab 8.*
Rate the **average difficulty** of the reading assignments from the last month.

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Responses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>1</td>
<td>2%</td>
</tr>
<tr>
<td>Easy</td>
<td>36</td>
<td>32%</td>
</tr>
<tr>
<td>Moderate</td>
<td>24</td>
<td>48%</td>
</tr>
<tr>
<td>Hard</td>
<td>9</td>
<td>18%</td>
</tr>
<tr>
<td>Very Hard</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

(*Figure 9.88: Rated difficulty of reading assignments for third month.*

How frequently did you **attend class** last month?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Responses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rarely or never</td>
<td>11</td>
<td>22%</td>
</tr>
<tr>
<td>Once a week on average</td>
<td>6</td>
<td>12%</td>
</tr>
<tr>
<td>Twice a week on average</td>
<td>4</td>
<td>8%</td>
</tr>
<tr>
<td>Always or almost always</td>
<td>29</td>
<td>58%</td>
</tr>
</tbody>
</table>

(*Figure 9.89: Class attendance frequency for third month.*

How many times did you **go to TAs or instructors for help** last month?

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Responses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 times</td>
<td>19</td>
<td>38%</td>
</tr>
<tr>
<td>1-2 times</td>
<td>17</td>
<td>34%</td>
</tr>
<tr>
<td>3-4 times</td>
<td>10</td>
<td>20%</td>
</tr>
<tr>
<td>5+ times</td>
<td>4</td>
<td>8%</td>
</tr>
</tbody>
</table>

(*Figure 9.90: TA help frequency for third month.*

Rate how helpful the **time in class** has been in helping you succeed in this course.

<table>
<thead>
<tr>
<th>Helpfulness</th>
<th>Responses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not helpful (too hard, I can't keep up)</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Not helpful (too basic, I don't need it)</td>
<td>4</td>
<td>8%</td>
</tr>
<tr>
<td>Somewhat helpful</td>
<td>34</td>
<td>28%</td>
</tr>
<tr>
<td>Moderately helpful</td>
<td>17</td>
<td>34%</td>
</tr>
<tr>
<td>Very helpful</td>
<td>15</td>
<td>30%</td>
</tr>
</tbody>
</table>
Figure 9.91: Rated usefulness of time spent in class for third month.

Rate how helpful course materials (reading assignments, materials in Canvas) have been in helping you succeed in this course.

| Not helpful (too difficult to understand) | 1 respondents | 2 % |
| Not helpful (too basic) | 0 % |
| Somewhat helpful | 11 respondents | 22 % |
| Moderately helpful | 13 respondents | 26 % |
| Very helpful | 25 respondents | 50 % |

Figure 9.92: Rated usefulness of course materials for third month.

<table>
<thead>
<tr>
<th>Q9 - Class time likes</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussing/working on labs</td>
<td>18</td>
</tr>
<tr>
<td>Live coding</td>
<td>11</td>
</tr>
<tr>
<td>Review reading</td>
<td>11</td>
</tr>
<tr>
<td>Diagrams and pseudocode on whiteboard</td>
<td>10</td>
</tr>
<tr>
<td>Tailored to class, ask what we want to cover</td>
<td>6</td>
</tr>
<tr>
<td>I don't have to attend</td>
<td>4</td>
</tr>
<tr>
<td>Answers to students' questions</td>
<td>3</td>
</tr>
<tr>
<td>Practice interview questions</td>
<td>3</td>
</tr>
<tr>
<td>Teaching to labs and exams</td>
<td>2</td>
</tr>
<tr>
<td>Wish I had gone more</td>
<td>1</td>
</tr>
<tr>
<td>Instructor's programmer humor</td>
<td>1</td>
</tr>
<tr>
<td>Instructor maximizes each 50 minute class</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 9.93: Aggregate summary of qualitative class time likes for third month.
### Table 9.94
Aggregate summary of qualitative class time suggestions for third month.

<table>
<thead>
<tr>
<th>Q10 - Class time suggestions</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>28</td>
</tr>
<tr>
<td>More time working on labs</td>
<td>5</td>
</tr>
<tr>
<td>Better alignment with lecture topics and labs</td>
<td>3</td>
</tr>
<tr>
<td>Less review of reading assignments, more practical application</td>
<td>3</td>
</tr>
<tr>
<td>More discussion/Q&amp;A</td>
<td>2</td>
</tr>
<tr>
<td>Lectures on BST/AVL got repetitive</td>
<td>2</td>
</tr>
<tr>
<td>Focus more on theory and less on code</td>
<td>2</td>
</tr>
<tr>
<td>Strike balance between too much help and not enough help</td>
<td>1</td>
</tr>
<tr>
<td>Slower coding</td>
<td>1</td>
</tr>
<tr>
<td>More practice interview questions</td>
<td>1</td>
</tr>
<tr>
<td>Greater emphasis on algorithm implementation</td>
<td>1</td>
</tr>
<tr>
<td>More review of reading assignments to solidify concepts</td>
<td>1</td>
</tr>
<tr>
<td>More “test yourself” practice problems</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q11 - General suggestions</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>No change</td>
<td>20</td>
</tr>
<tr>
<td>Improve some reading assignments (AVL, Hashing, etc.)</td>
<td>3</td>
</tr>
<tr>
<td>TAs not helpful</td>
<td>2</td>
</tr>
<tr>
<td>More lectures on debugging techniques</td>
<td>2</td>
</tr>
<tr>
<td>More labs, but make them smaller and easier</td>
<td>2</td>
</tr>
<tr>
<td>More readings, but make them optional</td>
<td>2</td>
</tr>
<tr>
<td>More practice interview questions</td>
<td>2</td>
</tr>
<tr>
<td>More optional practice problems</td>
<td>2</td>
</tr>
<tr>
<td>Break down the larger reading quizzes</td>
<td>1</td>
</tr>
<tr>
<td>Show common coding mistakes for labs and what resulting output is, then how to fix it</td>
<td>1</td>
</tr>
<tr>
<td>Have Friday readings due on Saturday</td>
<td>1</td>
</tr>
<tr>
<td>Too much packed into course, too fast-paced</td>
<td>1</td>
</tr>
<tr>
<td>Meet twice a week for 75 minutes instead of 3x for 50 minutes</td>
<td>1</td>
</tr>
<tr>
<td>More review of CS 142 topics at beginning of semester</td>
<td>1</td>
</tr>
<tr>
<td>Less writing code/pseudocode on the board (use computer with projector instead)</td>
<td>1</td>
</tr>
<tr>
<td>Better instructions for labs</td>
<td>1</td>
</tr>
<tr>
<td>Labs 7 and 8 too difficult, easy to give up</td>
<td>1</td>
</tr>
<tr>
<td>Show lab solution after lab is due</td>
<td>1</td>
</tr>
<tr>
<td>Better alignment with lecture topics and labs</td>
<td>1</td>
</tr>
<tr>
<td>Coding exams instead of objectively-scored exams</td>
<td>1</td>
</tr>
<tr>
<td>More small coding assignments</td>
<td>1</td>
</tr>
<tr>
<td>More help and class time for Lab 8</td>
<td>1</td>
</tr>
<tr>
<td>Project-based labs, use data structures instead of implement them</td>
<td>1</td>
</tr>
<tr>
<td>More focus on theory and practicality than implementation</td>
<td>1</td>
</tr>
<tr>
<td>Extra credit for early labs</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 9.95. Aggregate summary of qualitative suggestions for third month.

Because the end-of-November survey was the last survey assignment of the course, many students wrote about their overall experience in the course. Below are some of the qualitative responses to the November survey from students who enjoyed the course.

“I feel that Nate has done a great job adjusting the time in class to be super helpful. I’m amazed at how much we get out of a 50 min class.”

“I love not having to go to class! This is the best thing in the world! Please make all the classes like this!”

“I think you do a great job of using class time for subjects we are needing help with. Around the time of lab due dates it is nice having class time to work on them with your help.”

“I really like how Sir Nathan walks through all the different implementations for the Labs. This helps us understand the guidance needed to code up the labs.”

“I really liked how we went over hypothetical code for labs like BST and AVL, and we went over the same parts so many times. It helped me understand not just what it was doing but how to code that.”

“Everything we do in class relates very directly to the labs and the assignments.”

“I like the reading quizzes a lot! And I like how the labs are focused on building the data structures we are learning about.”

“I've been very happy with the way things have gone this semester. I appreciate the pacing and the progression of topics.”

“Overall I really enjoyed this class and I don’t know what I would change. I remember coming into this class people saying it was really difficult, but this class was actually really cool while still having a good level of challenge.”

“I have enjoyed this class and in general wouldn’t change anything about it. I really liked the set up of the class and expectations have always been very clear.”

“It has been a really good class and I actually feel like I am starting to understand Computer Science for the first time ever.”

“There's not much I would change about the class. It’s been my favorite CS class so far. I think the structure is great. The reading quizzes are super helpful and make for easy learning. The labs are challenging but they really make me learn the material. I give this class an A+!”

“Honestly this has been a really great class. Everything is very fair, I like that the reading quizzes are unlimited. I felt like I learned a lot from them. We had enough time for each lab and I never felt overly rushed unless it was my own fault. I felt like I had enough resources for help that I never felt like I was drowning... so overall I wouldn't change anything.”
“This has been my fav class in college so far.”

9.16.2.4 Student Ratings

In teaching a university-level course, the students are just as much clients as the department. The students are adults who are paying for an education, and instructors should be just as accountable to the students as they are to the department. Thus, student ratings are important to take into account when evaluating a course.

That being said, student ratings have their limitations. Not all students fill out student ratings—only 35 out of 55 responded in my case—so the data are incomplete. Student ratings also share the same limitations of any Likert-scale assessment. Biases and current mood can affect the outcomes, and not everyone interprets the scale in the same way.

In terms of instructor effectiveness, I was consistently ranked above the department average in every category (cf. Fig. 9.96). Most notably, 88.6% of students ranked Opportunities to get help as Very Effective, with the remaining students ranking it as Effective. One of the project goals was to encourage students to help each other or better help themselves, and these numbers provide evidence that that goal was attained. Also notable is that 74.3% of students ranked Explained concepts effectively at Very Effective compared to the department average of 50%, providing evidence of deeper learning. No student responded Not at All Effective or Not Very Effective in any categories.

![Figure 9.96. Overview of student ratings regarding instructor effectiveness. The green represents percentages for my section, blue the average percentages for the department.](image-url)
In terms of achieving the aims of a BYU education, I was ranked relatively low in the *Spiritually strengthening* category (cf. Fig. 9.97). In my project proposal I had mentioned the possibility of adding some learning outcomes to include that aim, but in my design I forgot to account for it. Consequently I had no lectures or assignments that attempted to connect subject matter to gospel subjects, which explains the low rating. I was ranked relatively high in the *Intellectually enlarging* category, and ranked close to the department averages for *Character building* and *Leading to lifelong learning and service*.

*Figure 9.97 Overview of student ratings regarding the aims of a BYU education. The green represents percentages for my section, blue the average percentages for the department.*

9.16.2.4.1 Qualitative Responses

Included below are all the comments submitted through the BYU Student Ratings system.

1. **Instructor Effectiveness Comments**
   a. Explained concepts effectively
      i. In describing certain data structures, he’d go through step by step, but didn’t focus so much on illustrations that might help you understand the big picture.
      ii. The in class discussion was relevant to the new concepts that came up in the labs.
      iii. Visual learning was the best way for me to learn, and his teaching style changed to match the learning styles of many by the early middle of the semester.
      iv. Nathan was the best at teaching difficult concepts simply. The reading quizzes were awesome.
      v. Explained most of the difficult code well and effective
      vi. Everything made sense, so he did a good job.
      vii. Many of the concepts of this class are very challenging and he was able to explain them with diagrams and did a really good job.

Commented [5]: Tell students that it’s appropriate to start your coding projects with a prayer, since they’ll probably end that way ;)

[^1]: New footnote reference.
viii. It was very obvious that he understood the concepts and the reading/quizzes were great to help me learn.
ix. Whenever we discussed difficult topics, he would repeatedly touch on aspects that were difficult to do or understand and would often have us tell him what he should explain.
x. Explained by showing the concept and then giving an example of actual code.
xii. Very effective. I left the class fully informed of what was expected of me, and he always focused on industry specific content.
ixii. Having discussion based learning made it easy to ask questions about difficult concepts.
ixiii. When it came to learning challenging material, most of the time instead of trying to teach the material, we were just given most of the work that we needed and then were left to figure out the last part.
ixiv. The material was taught very well in class and in the readings.
b. Well organized
i. He put all sorts of helpful resources on Canvas, and gives us many different links to tutorials or other helpful things.
ii. In class discussion was always helpful and relevant to what we needed to learn to complete the labs. In-class discussion and the lab schedule didn't seem to line up very well towards the end of the course though.
iii. Though the organization changed over the course of the semester (pilot class probably had a lot to do with that), I found the coursework fairly effective in enhancing my understanding.
iv. This class was organized really well. I wish more classes were like this.
v. Content explained very well
vi. I really liked the reading assignments he had us do. They were to the point and the questions were helpful in making sure I understood what I read.
vii. It would have been better for me to be tested on a material (complete a lab) before going on to tons of new stuff.
viii. I feel like the topics didn’t really build off one another, but I don’t think it was always possible. However, the order was logical and helped make the progression towards more difficult topics possible.
ix. I loved the organisation of this course. It was really wonderfully done. He made it easy to access everything we needed online and said just to come to class as often as we needed. I loved the freedom! Please do more classes like this!
x. very, very organized. no surprices [sic]
xi. Students knew what to expect as a schedule was provided and followed. Readings were assigned before they were discussed in class.
xii. Nate did everything in his power to make sure that we had every resource available to complete labs, excel on exams and learn all required concepts.
xiii. I liked the readings and the quizzes that came with it, it made it much easier to study for exams too.
xiv. Really well organized curriculum and information. He's basically provided as much information as possible and created his own quizzes, which were probably the best for information retention.
xv. I loved the way this class was organized. I have some friends in other CS235 sections, and their class work seemed a bit more of an unnecessary pain.

c. Opportunities to get help
   i. He would directly help you with labs. He always respected questions and never condescended.
   ii. Sometimes class time was taken to give help for upcoming labs. He was also always available to answer questions through Campuswire.
   iii. Always held multiple help sessions at the end of classes for students, and indicated a strong desire to help everyone understand fully.
   iv. Always willing to answer my questions and explain concepts.
   v. I’m not sure about this, but I assume he was good about this.
   vi. He had a group chat on campus wire to encourage people to post there [sic] questions. He was always trying to help us understand the concepts.
   vii. Great use of class time in helping students as well as very quick response online when students asked questions.
   viii. He was very open to questions in class and often set aside time in class specifically to help those who needed it.
   ix. Always willing to help us directly, set aside time during class for students to come for him to help, also very accessible outside of class through campuswire.
   x. Took time to answer students questions when time permitted. Helped in TA lab at the end of the semester.
   xi. Very very helpful. He always made himself available, and did everything in his power to give the students an edge on the lab and the exams.
   xii. He spends some of the time in class helping us, he’s also been very good about answering questions online and spending time outside of class to help.
   xiii. He created an online group where students could ask questions and he always did a great job answering them. His in class help sessions were also very useful.
   xiv. Really progressive style of teaching. Felt less like a lecture on a set curriculum and more like actual preparation for future employment.
   xv. There were lots of ways to ask for help, and the TAs were available. Nathan also did a good job of addressing needs in class and offering time for students to come up individually and get help.

d. Opportunities for student involvement
   i. There wasn't enough classroom activities or practice problems to help perfect your knowledge.
   ii. Questions in class were always welcome and answered well. There was also group classwork and polls.
   iii. Surveys at the beginning of some classes, group activities, and group lab work encouraged participation.
   iv. Some lectures were just that, lectures. Very little response and thought.
   v. He was good about this.
   vi. He encouraged us to use are laptop and to program with him. He also would show really good examples.
   vii. The class structure was very great for learning.
Most of the in-class work was done on the whiteboard so we could all see while he discussed it, and he was open to questions throughout. However, it was mostly lecture (which I like).

Opportunities were given to follow along with examples in lecture, though sometimes it was a little fast to do so.

He was very open to questions, and when someone (mostly me) would give a dumb answer in class, he would never say the student was wrong, but that they were close and then go on to explain the difference between the students answer and the concept that was being taught.

If you don't participate in the learning process you'll basically fail, that's really just the nature of the subject material, you can't learn to program unless you do it. Class time was less like lectures and more like discussions, which I find much more helpful.

I felt the learning process could have been much more participatory. He didn't ask many questions during lecture.

Coding in class, quizzes, in class quizzes. Lots of labs that are actually helpful to future coding interviews.

I feel like class time was well spent.

Responded to students respectfully

He always appreciated questions and concerns.

Questions were always answered in a helpful manner.

Clearly this professor understood our concerns and made sure we were answered even if he didn't fully understand the questions at first. Excellent support.

Always very willing to answer questions. Went out of his way to answer them.

Never felt demeaned

He was good about this.

Like mentioned before, he was very good about answering all students' questions, whether in class or outside of it.

All around respectful.

Very very respectful. See answer for number 2.

I never felt like anyone was shut down when asking a question or making a comment.

Being a student himself, Nathan really helped me feel like he understood how I felt. Especially with regards to employment and practical project advice.

I don't think anyone ever felt like they had a dumb question, which is sometimes the case in CS classes with a wide spectrum of experience in the class.

2. BYU Aims Comments

a. Spiritually Strengthening

i. Good environment.

Getting my code to compile has taught me a lot about prayer and therefore brought me closer to God.

ii. Spiritually Strengthening

This was a CS class, there wasn't really any chances to bring up the gospel

v. I can't recall any sort of religious discussion.
vi. ...computer science isn't really a spiritual topic...I also may or may not have sworn at my code a few times over the semester, but that had nothing to do with class instruction.

vii. Gave me faith in CS again.

viii. It's CS, so there isn't too much directly related to spiritually strengthening from a gospel perspective. If we are defining "spiritually strengthening" as being excited to learn and motivated to carry on in the field, then I totally agree. This semester has been the biggest boost to my motivation!

ix. Not really an emphasis on spirituality, but an uplifting environment none the less [sic].

x. There wasn't much religious involvement in class.

xi. I mean, there's only so much you can do in CS235, but I enjoyed class and the work.

b. Intellectually enlarging

i. Nate was at his best when he explained the usefulness of various concepts, how it relates to industry. And he even gives advice for job interviews.

ii. I became a much better programmer throughout this course.

iii. Definitely a challenge, definitely grew a lot from this level of problem solving.

iv. Very helpful for understanding problem solving in certain

v. Definitely feel smarter, especially since finally understanding recursion.

vi. I feel a lot more confident in my ability to code now.

vii. For all the reasons above.

viii. I have a broader understanding of the course material.

ix. I learned a lot from him and I really enjoyed it.

x. I learned so much in this class!

b. Character building

i. The labs taught you patience, but weren't unreasonable.

ii. I now know how to not break down when compiler errors get into the hundreds.

iii. Honesty and integrity were fostered

iv. Helpful for problem solving

v. Trusted students which not all professors do.

vi. He trusted us and expected us to be honest without putting a ton of restrictions on how we learn. As long as you understand what your code does and how it does it, you could use many resources. He also provided a lot of resources. Also, doing hard things build character.

vii. Rewarded attendance and excelling heavily.

viii. Nathan gives all the help he can give, but still places the responsibility to learn on the students. I was able to get my first campus technical position thanks to his advice.

ix. I learned to put in time and effort in challenging projects with deadlines. My patience and problem solving was tested.

x. I personally learned to be better about managing my time and putting the requisite effort from this class.

xi. I feel better prepared to have conversations about these topics and work with other computer scientists with these tools.
d. Leading to lifelong learning and service
   i. Nate expressed a love of the subject, which rubbed off on the students.
   ii. The things in this class are relevant to the fields I am planning on going into.
   iii. Showed us basic structures that we can build upon for all future classes
   iv. Helpful for teaching us how to use given resources to solve our own problems
   v. I just keep getting more excited about this field of study!
   vi. Very good at teaching stuff at an industry level.
   vii. I really liked the fact that he was very concerned about teaching us what would be useful in that career versus just teaching us about random subjects.
   viii. I am very excited to continue learning more within computer science, especially after this class.
   ix. I think the principles and concepts in this class will apply to my career for years.

3. Workload Comments
   a. The labs always take longer than the average expected work-time.
   b. Varied a lot per week depending on which labs were due and how difficult they were.
   c. I felt this was a reasonable amount of time spent in this type of class working on lab work, quizzes, and readings.
   d. Depends on the week. Some weeks I only spent time on the reading quizzes and other weeks I spend hours doing the labs. Overall very doable.
   e. Most time spent on lab
   f. The labs were much easier than the labs in CS 224.
   g. We had many labs that were due almost weekly and they required us to use a lot of time.
   h. Nathan did a great job of not overloading us with busy work, which I really appreciated.
   i. It would fluctuate depending on what lab we were working on for the class but it felt like a fair amount asked of us. It was a good amount that didn't drown us even though it required a little more from us.
   j. The labs took the most time, anywhere from 6-12 hours in a week. Everything else was very reasonable, consistent quizzes without any busy work.
   k. Most of time spent working on labs, much of which was in the TA help lab. Some spent on reading and quizzes [sic].
   l. Just working on the labs
   m. Time is spend on programming assignment, and online/in-class resources are very helpful for understanding and completing the work.
   n. About 3 hours on reading/quizzes per week and about 7 hours per lab
   o. Most of the time was spent doing the labs.
   p. Because the course material is easy and the Computer Science department doesn't let you skip any classes beside CS 140, this class took hardly any time.
   q. Appropriate workload
   r. CS is by nature time intensive, but I think this class did a good job of making our time outside of class very well spent.
   s. I did most of the readings during class... and then usually did the labs fairly quickly.
t. amount of time spent on homework depended on when labs were due.

4. Additional Comments
   a. Nate really knows his stuff. Sometimes, though, it was hard to follow the lecture when it involved 12+ step-by-step instructions.
   b. I really enjoyed this class. Beforehand, I had heard how awfully hard CS235 was, but over the course of the term I realized that with good instruction and materials all these labs were possible and understandable. Excellent teaching style too.
   c. Nate did a good job teaching the course. I felt I was able to complete each assignment without excessive difficulty.
   d. I quite enjoyed how this class was laid out. One thing I really liked was how he gave us options. For instance, we could get the textbook if we wanted to, but it wasn’t mandatory. We could go to class if we wanted to (I did, since it was worthwhile), but that too wasn’t mandatory.
   e. He did an excellent job teaching! I would definitely recommend him, if he were teaching again, to friends!
   f. Nathan is honestly one of the best professors I have ever had. He explains things very very well and has helped me to understand all the concepts in class. I was never overwhelmed with the course, but I was pushed to think through things and develop my programming skills. I was very impressed with him as a teacher and if he were teaching again I would recommend him to all my friends.
   g. He was a great teacher and really helped me learn.
   h. He was an amazing teacher. I feel like for the first time ever I was able to understand computer science and get the help I needed. I finally had hope in the idea that I could learn something hard. He was able to explain everything well and always kept the class engaged.
   i. Nate was a phenomenal computer science instructor because he focused on the broader concepts. He never made us guess at algorithmic logic only for the sake of making his course more difficult.
   j. excellent instructor, much preferred than the veterans who have been teaching for decades.
   k. Nathan Fox is the best professor I have had at this University.
   l. This course is well organized, encouraging students to learn through their out of class work and reinforcing those concepts in class.
   m. Nathan Fox was an awesome instructor. You should hire him when he's done with his dissertation.
   n. I just wish that we would’ve had some more hands-on learning, like starting class with the files all set up to be coded on, and then start coding on the subject we're learning (really similar to the labs, but not the same), but purposefully make the same mistakes that you think we'll make, and then try to run the program and when it fails, ask us what went wrong and have us solve it. That way you’re accelerating our learning process and showing us how to overcome the struggles we’ll face when coding, or at least that's the goal. I feel like we were just given 95% of the work (that wasn't really taught to us) and we just had to figure out the last 5% on our own.
   o. nothing
   p. Nathan was an awesome teacher. since the class was part of his masters thesis he did things a little different than other professors might have. He sent out lots of surveys asking students opinions on what worked in class and what didn't. after a
few weeks of class I feel like everything we did in class was efficient and effective in teaching us the material. Nate was also very willing to help whenever we needed it. 10/10 would recommend.

q. I very much enjoyed this class! I’m happy I signed up to take it from Nathan.

r. I loved the organization of the class.

In addition to student ratings, I had a couple of students who wrote messages to me at the end of the semester. Images of these messages are included below.

Figure 9.98. A written card from a student given to me at the end of the semester.
Hey Nathan!

I should have done this a long time ago, but I just wanted to write to thank you properly for last semester. It was really a magical turning point of sorts for me, and I feel like the luckiest guy in the world because I'm right on track to where I want to go! CS 235—a class that I withdrew from before, by the way—was right up there in the "courses that I was most afraid of right after the mission" list, but it ended up being one of the best ones.

In a weird way, you've probably done just as much for me in terms of web programming as you have leetcode/hackerrank programming, because all the advice you gave me has resulted in some seriously incredible domino effects. You won't believe what happened today! I finished the flexbox zombies thing ages ago, and I really wanted to do the next course on CSS grid, but Steve (the maker), was charging a ton for it. I knew that it was totally worth the price, but I ended up reaching out to him on twitter and told him my whole programming story and out of the blue he offered it to me for a tiny fraction of the price! I also just went to my second Vue.js meetup, where I'm making some really great friends and future job prospects.

I'm also about to clock in close to 100 hours of work over this winter break! I love the job to bits and Eric is a wizard. No joke the most intelligent programmer I've ever met for his age and above. I'm living the dream at the moment, and it's all thanks to you and all the other kind people on twitter that I not-so-creepily reach out to. I'll be wanting to switch over to web in two years or so, so keep me in the loop! I'll send you updates on what I'm up to and I'm sure I'll see you on campus or JS events in the future.

Thanks dude.

*Figure 9.99. An email from a student at the end of the semester.*

### 9.16.3 Assessments

See section 9.13 for an overview of assessment reports and instruments. This section includes some of that data in a side-by-side comparison with other CS 235 courses.

#### 9.16.3.1 Exams

Assessment data are primarily used for summative evaluation purposes to provide evidence related to the CS department's primary goal of promoting deeper learning. It is difficult to measure deeper learning, and my methods for doing so are not perfect. By defining deeper learning in terms of Bloom's taxonomy, one way of showing deeper learning is to design assessments that test more higher-level learning outcomes and then show that students performed well on the new assessments.

Tables 7.100 and 7.101 represent the exam specifications for the midterms for my new course and the flipped version of the course as taught by Dr. Clement in Winter 2018. My midterm has a drastic increase in application-level assessment items, so it is clear that the new midterm is better designed to measure deeper learning as defined in terms of Bloom's taxonomy. Figure X shows the distribution of scores for the new midterm. With a mean of 92%, a low of 72%, and a standard deviation of 3.93, the distribution of scores shows that students performed well on a midterm designed to measure deeper learning.

<table>
<thead>
<tr>
<th>Content</th>
<th>Know</th>
<th>Understand</th>
<th>Apply</th>
<th>Analyze/Evaluate</th>
<th>Total #</th>
<th>Percentage</th>
</tr>
</thead>
</table>
Table 9.100. The table of specifications for the midterm of my new CS 235 course.

Table 9.101. The table of specifications for the midterm of the flipped version of the course.

Figure 9.102. The distribution of student scores for the new midterm.
9.16.3.2 Final Grades

The following three figures show the grade distributions for my section of CS 235, the lecture-based version taught in Winter 2018, and the flipped version taught in Winter 2018. No two of these courses had the same assessment set, so the comparison does not provide much insight into how course design affects student learning. However, looking at grade distributions can give some insight into how well students were prepared for their assessments.

In general, grades from my section were higher than grades from the lecture-based version and lower than grades from the flipped version. This might be explained in part by the increased rigor in exams between the flipped version of the course and my version. It is also interesting to note that the course design that produced the most A grades was also the course design that weighed exams the least. My design weighs exams at 35%, the lecture-based design weighs them at 40%, and the flipped design weighs them at 30%. This suggests that students generally score better on labs and assignments than on exams, causing grades to fall when exams are more heavily weighted.

![Grade Distribution for My Section (n=56) - Fall 2018](image)

**Figure 9.103.** Grade distribution for the 56 students who took my version of CS 235 taught in Fall 2018.

Commented [7]: You could check this fairly easily, couldn't you?

Commented [8]: True. Comparing figure 7.31 to figure 7.36 shows that at least in my section, cumulative lab scores were higher than cumulative test scores.
9.16.4 Pedagogical Concepts from Literature Review

In my literature review, I identified the following pedagogical concepts that were most often mentioned in relation to teaching coding or computer science:

1. Paired programming and communities of learning,
2. Unplugged learning,
3. Modelling levels of abstraction,
4. Game design,
5. Use-modify-create,
6. Flipped classroom,
7. Live coding,
8. Think-pair-share,  
9. Code tracing, and  
10. Subgoal (problem decomposition) modelling

These pedagogical concepts relate to the CS department’s goal of promoting deeper learning. In an effort to meet this goal of promoting deeper learning, I relied on some of these pedagogical concepts in my course design (cf. Table 9.106).

<table>
<thead>
<tr>
<th>Concept</th>
<th>Application</th>
</tr>
</thead>
</table>
| Communities of Learning| 1. I allowed students to work on labs in groups and submit the same code as long as each student understood the code  
                           2. I used Campuswire as a Q&A tool to allow students to answer each other’s questions                                           |
| Flipped classroom      | 1. I expected students to learn new material from the reading assignments so that we could focus on writing code and working on labs in class.  
                           2. When lab due dates drew near, I dedicated a portion of lecture time (or sometimes an entire lecture block) to helping students with their labs. Students wrote their names in a list on the board and I helped them through their code one by one. |
| Live coding            | 1. I created programming practices that shared some similarities with the labs and coded the solution in class to model how to approach computational problem solving. |
| Think-pair-share        | 1. I created a think-pair-share web application and used it for think-pair-share activities in the classroom.                            |
| Code tracing           | 1. Many of the questions on reading assignments and exams asked students to look at code and determine the output. Like language literacy, computer literacy can be gained by learning how to read before learning how to write.  
                           2. For each lab, students were given some initial code to start working with. Students had to read through and understand the provided code in order to add to it to get the labs working. |
| Subgoal modelling      | 1. An important part of live coding is modelling the entire problem solving process from beginning to end. As part of my live coding lectures, I made sure to explicitly break down the goal into smaller subgoals, then tackle each subgoal one at a time.  
                           2. Labs were divided into multiple parts, each part having its own description, point value, and test cases. This helped break each lab down into subgoals, but students still had to decompose each of these parts into smaller subgoals in order to succeed. |

*Table 9.106: Pedagogical concepts from literature that I relied on and how I applied each in my course design and implementation.*
9.16.5 Instructor Experiences

In a typical design and development project, I would have interviewed the instructors during and after the implementation to get an idea of how they felt about the course as they were teaching it. In this somewhat unique situation in which I am designer, developer, instructor, and evaluator, there is no instructor from which I can get unbiased feedback. The situation being what it is, the best I can do is to write about my own experiences. Having little teaching experience prior to this, I do not have much to compare this teaching experience against. This is a limiting factor of my evaluation, and can only be remedied in the future when a less biased and more experienced instructor teaches this version of the course. In this section, I discuss my experiences working with some of my primary deliverables as outlined in section 9.11.

9.16.5.1 Reading Assignments

When I met with Dr. Clement (the CS professor who taught the flipped version of the course in Winter 2018) and asked him about what he thought would be most helpful for this course, he said that he wanted more reading assignments. Near the end of Winter 2018 he started making interactive reading assignments using Canvas quizzes to keep students accountable for the reading. Prior to that, he put links to reading materials in Canvas but had no mechanism for holding students accountable for the reading. Without that accountability for the reading, it made it difficult to use the flipped classroom approach effectively.

Using the required reading assignments made it easy for me to use the flipped classroom approach. Instead of designing lectures to cover all course content, I was able to arrive in class and just ask students what questions they had from the reading assignments. When they didn’t have questions, I could launch right into whatever I had prepared, be it a deeper dive into the material, live coding activities, or helping students with labs. My experience with the reading assignments was very positive overall, and if I were to teach this course again I would certainly use the reading assignments again.

That being said, there are some changes that need to be made to the reading assignments. In order to stay ahead, I often wrote the reading assignments 3-6 weeks before they were due. In some cases, I wasn’t yet sure how I wanted to cover the material in class or how I would connect it to other content. In some cases, such as the reading assignments related to AVL trees, my own understanding of the content developed more as I prepared lectures on it, so in those cases the lectures were much more clear and effective than the reading assignments. (After teaching the class, I rewrote the AVL Add/Remove assignment to match my lectures before turning in my design to the CS department). Also, none of the reading assignments have been edited, so having an editor go through them to fix grammar errors, make the voice consistent, etc. would certainly improve them.

9.16.5.2 Exams

My initial worry with using objectively-scored online exams was that the exams would be too easy. It’s easy for students to get away with cheating on unproctored online exams and it can be difficult to write objectively-scored questions that effectively measure learning outcomes.
However, looking at the exam scores (see section 9.13) gave me confidence that the exams were an appropriate difficulty level.

9.16.5.3 Live Coding Activities

The live coding activities were very effective over the first half of the semester when we were reviewing CS 142 content and learning about how to use data structures from the C++ STL. It was relatively easy to come up with live coding activities that related to the labs enough to be helpful, but not so much that students could do the labs without thinking about it.

In the second half of the semester, the focus shifted from using data structures to implementing data structures. Labs 5-9 all dealt with implementing a common algorithm or data structure. In these cases, it was a lot more difficult to design live coding activities that related to the labs enough to be helpful without giving away code for the labs. So in the second half of the semester, live coding activities became more sparse. When I did live coding activities for these labs, they were essentially lab help sessions where we wrote some of the code for the lab together as a class.

9.16.5.4 Think-Pair-Share

I used think-pair-share questions regularly in the first 6 weeks of the semester. I often started class with a think-pair-share question that was related to the reading assignment that students should have had completed before class. In my experience, it seemed like it wasn’t worth the time that it took. 50 minutes is not a long time, and taking 5 minutes to go over a think-pair-share question ate up too much of that time. In many cases, the vast majority of the students got the question right the first time, so I just skipped the pair-share and launched into an open discussion about the question. In other cases where I did ask the students to discuss their answers with a neighbor, some students (often the ones who did need help to get the correct answer) did not participate. I designed my think-pair-share web application for quick use so that students did not have to authenticate to answer questions, but the downside to that is that the data I collected are not tied to individual students, so there is not much use for the data. After the first 6 weeks I stopped doing think-pair-share activities and just did class discussions instead.

9.16.5.5 Lab 9

Lab 9 was a lot of fun to design. Instead of just having students implement a hashmap, lab 9 tests their hashmap to solve an interesting real-world problem. On top of that, lab 9 has students use heapsort, which is covered in reading assignments but is not typically included in labs. However, I didn’t like assigning lab 9 out so close to the end of the semester and having it due the last day of class. I really like lab 9, but I’m not sure how I would use it in the future. I might consider making lab 8 (the AVL lab) extra credit and then assigning lab 9 a week earlier to give students more time to get it done. I think that the learning outcomes of lab 9—such as understanding the nature and purpose of hash functions—are more important throughout the rest of the CS program than the learning outcomes of lab 8. The lecture-based version of the class already uses the AVL lab as an extra credit lab, so there is established precedent for not requiring it.

9.16.5.6 Cloud9 IDE
Overall I liked teaching with the Cloud9 IDE, but there were some pain points. I liked that when students asked for help, they could just give me their workspace URL and I could help them remotely. Cloud9 allows for real-time collaboration and has a messaging feature, which made remote help easy and effective. For students who did choose to work on labs in pairs or groups, they could collaborate easily in the online environment. Having everyone on a virtual Linux machine made it easier to design labs, and it made programs like GDB and Valgrind available to everyone. With the virtual Linux shell provided in Cloud9, students could easily use command-line git to download code for live coding activities and for labs.

One of the biggest pain points of Cloud9 was debugging. In CS 142 students learn how to use the debugging tools in Visual Studio. Cloud9’s debugging tools for C++ were severely lacking and did not work very well with the way labs are configured. Many students ended up copying their code into Visual Studio for debugging, then uploading their code back to Cloud9 once they were done. Unfortunately, in some of these cases their code worked in Visual Studio but not in Cloud9 because the C++ compiler is different, with the Visual Studio compiler being more forgiving.

For many students, another pain point was learning how to work in the Linux environment. Once I identified that pain point, I was able to respond to it and develop training materials to help students get started. Many students felt overwhelmed at first, but were later grateful for the opportunity to become familiar with Linux. Some even said that learning Linux was one of the most practical skills they learned from the class, and I don’t disagree.

There were also rare instances where Cloud9 lost files. It happened to me in class one time in the middle of a live coding activity. I had code files open in the editor, but when I used the Linux terminal to try to compile my code, it said that the files didn’t exist. I had to make a backup and restore it to get it working again. This happened to a couple of students throughout the semester, but fortunately they had backups and were able to restore lost code.

The version of Cloud9 IDE that I used in Fall 2018 was taken offline at the beginning of 2019. Now the only option is to use the version in AWS, which might have resolved some of these issues.

9.16.5.7 Campuswire

Just before the first week of teaching, I was trying to figure out which communication tool would be best to use for general Q&A outside the classroom. I have had classes use Google Groups or Piazza, so I was looking into using those. As I was looking into it, I got an email from a brand new product called Campuswire that was looking for beta testers. I decided to go for it. Because it is in beta, it did change a lot over the semester, but always for the better. It allows students to ask questions, answer questions, and upvote answers. Instructors and TAs can also answer questions and can post notes for general communication. There are also private discussion rooms where instructors and TAs can have discussions that students can’t access.

My experience with Campuswire was positive, but it wasn’t used as much as I thought it would be. In almost every case, I was the only one to provide answers to questions. I had hoped that students would be more engaged in answering each other’s questions. I was asked questions in class or in lab help sessions that would have been better asked and answered in Campuswire. If I were to teach again, I would use Campuswire again but I would design better training materials.
for it so that students can quickly jump right into it. I might even require students to post both a question and an answer on Campuswire near the beginning of the semester, just to make sure that everyone knows how to.