A Fully Flipped Active Learning Course

CELINE LATULIPE

In this chapter, I present a case study description of a large, fully flipped active learning (FFAL) class in computer science. The class, a sophomore-level core course that combines both theory and programming, is required for all students in our major. The FFAL section of this course has been taught for five semesters. Research on how this class has been designed for more inclusive success was presented at the Institute of Electrical and Electronics Engineers (IEEE) Frontiers in Engineering Education conference and is published and available for reference (Latulipe, MacNeil, & Thompson, 2018). There are many details on how to make such a large and critical class work, and this chapter will cover the details. I hope that by writing this chapter and describing how the whole class runs, others may be inspired to try something similar in their own large core courses.

Data Structures: A Core Course in Computer Science

The Data Structures class is a core course in computer science programs at most institutions, known as CS2. Data structures (lists, queues, arrays, graphs, and trees) are crucial building blocks in computer programming. Most applications and digital systems make use of at least one data structure to store information. At a minimum, computer science students need to learn how the most common data structures work, their relative efficiencies, and which data structures are appropriate for different types of problems or applications. This is the theory of data structures. In the application/implementation part of the Data Structures class, students learn to implement data structures from scratch, rather than just learning to use the data structures that are already built into high-level programming languages such as Java, Python, or C++. Students commonly complete at least some programming assignments as part of a CS2 curriculum.

CS2 courses vary widely in their focus on theory versus programming. Some professors focus on deep mathematical theory and ignore the applied programming aspects of the course. Instructors who put significant focus on the applied programming aspect of data structures vary in how much of the programming they require of students to be from scratch versus making use of built-in data structures. These variations occur across computer science programs at different institutions and sometimes across sections of the same class within a program.

Many CS2 curricula assume students already know how to program, and so the programming assignments are designed for students to complete outside of class without any help. Thus, many CS2 classes do not have a separate programming lab, and this is the case at UNC Charlotte.

Data Structures classes at UNC Charlotte have historically been taught in a traditional lecture format, with instructors sometimes adding code demonstrations to help show the programming application. Students are given homework assignments that may or may not involve programming, and they are assessed based on those assignments. This course is seen as a gateway course and typically has high rates of students who are not successful and either receive Ds, Fs, or withdraw (DFWs). dAt UNC Charlotte, this course is where we often lose students from groups underrepresented in computer science (students who are female and/or minorities).

Goals for Reworking the Class

In the summer of 2016, I worked to create a new FFAL section of the Data Structures course. My aim was to make the course a mix of theory and application but focused more heavily on the application. I treat the course as essentially the third programming course for our majors. My reason for doing this is based on data that show that the majority of students coming into our computer science program did not learn any computer programming prior to entering UNC Charlotte. This is not surprising, given that only 18% of high schools in North Carolina offer an Advanced Placement Computer Science class. Students who have taken two semesters of introductory programming usually are still not proficient or comfortable with programming. Like learning a foreign language, learning to program takes time and practice. Many of our students are economically marginalized and working multiple part-time jobs, so they struggle to find the time to practice programming outside of class, which often makes the learning process slow. That does not mean that these students are not capable, but it does mean that instructors need to have realistic expectations about what students are capable of doing in the first semester of their second year in a computer science degree.

It is also worth noting that the vast majority of our computer science undergraduate students get jobs in the industry right after graduation doing some type of software development or information technology (IT). Few go on to graduate school because there is high demand for skilled IT workers, and the salaries are very competitive. This points to the necessity to focus on the application of data structures more heavily than on the theory.

Given this context, I had a few specific objectives in mind when designing the fully flipped version of the Data Structures course. These are broken down into three professional development goals:

1. To help students continue on their journey of learning how to program.
2. To help students become resilient programmers, willing to work through bugs and use standard software testing methods to help discover issues with their own code.

3. To help students become collaborative team players and learn how to communicate effectively about computing and programming concepts because the software industry is very team-based.

I also had four subject-specific goals:

1. To help students learn which data structures to use for different types of problems.
2. To help students learn how to use the built-in data structures in Java.
3. To teach students how to build their own data structures from scratch.
4. To teach students to interpret data structure implementations built into a programming language by understanding abstract data types (ADTs).

FFAL Essential Class Components

I chose an FFAL structure to follow the restructuring I had already applied to the first two programming classes in our college. A flipped (or inverted) class is one in which the students consume the course content to gain facts, knowledge, and understanding on their own before coming to the class. These are the lowest order elements of learning in Bloom's taxonomy (Adams, 2015). When they attend class, they can actively engage with the material in higher-order cognitive learning activities that involve Bloom's theory of application, synthesis, evaluation, and creation. There are a variety of challenges to the flipped class approach (Maher, Latulipe, Lipford, & Rorrer, 2015), but most research shows significant benefits for student learning outcomes (Thai, De Wever, & Valcke, 2017).

The four main components to the overall structure of my FFAL class are:

1. **Structured Prep Work:** Students absorb content through watching videos and reading the textbook before coming to class, with a prep work quiz due before class starts.
2. **Lightweight Teams:** During class sessions, students sit at an assigned table with a team, and this seating plan and team stays the same all semester long.
3. **Active Learning:** During class sessions, students engage in peer instruction quizzes with their teams and pair programming activities.
4. **Varied Assessment:** Students are assessed across a wide variety of activities, including prep work, peer instruction quizzes, in-class programming labs, individual programming assignments, and individual tests.

These four components integrate together to form a very structured learning experience (Eddy & Hogan, 2014) that helps students stay on track as they learn to become programmers and learn the theories and application of the data structures content. I illustrate this case study with data collected from detailed, anonymous, end-of-class surveys that I give students time to complete on the last day of class every semester. Table 3.1 below shows various aspects of the course and how they have evolved over the first five semesters, including the enrollment, classroom setting, resources used, and the decreasing DFW rates, shown in the last row.

Table 3.1 Evolution Over Five Semesters of the FFAL Data Structures and Algorithms Class

	Fall 2016	Spring 2017	Fall 2017	Spring 2018	Fall 2018
Enrollment at census/ end	88/77	51/49	124/115	74/68	126/123
Classroom	KENN 236 Large active learning class. 14 large round tables seating nine students each	BINF 210 Medium active learning class. 14 small round tables seating four to five students each	KENN 236 Large active learning class. 14 large round tables seating nine students each	WOOD 135 Medium active learning class. 75 individual rolling chairs with under-chair storage	KENN 236 Large active learning class. 14 large round tables seating nine students each
Teaching team	Two professors (1F, 1M), two graduate students (1F, 1M)	One professor (F), one graduate student (M), one undergraduate student (M)	One professor (F), two graduate students (2F), three undergraduate students (1F, 2M)	One professor (F), one graduate student (F), three undergraduate students (1F, 2M)	One professor (F), one graduate student (M), six undergraduate students (2F, 4M)
Peer instruction quizzes	Turning Technologies clickers	Turning Technologies clickers	Poll Everywhere with word cloud warm-up questions	Poll Everywhere with word cloud warm-up questions	Poll Everywhere with word cloud warm-up questions
Discussion forum	Canvas Discussion Forum	Canvas Discussion Forum	Piazza, allowing anonymous posting	Piazza, allowing anonymous posting	Piazza, allowing anonymous posting
Textbook	Lewis & Chase, *Java Software Structures*	Lewis & Chase, *Java Software Structures*	zyBooks interactive textbook	zyBooks interactive textbook	zyBooks interactive textbook
Final course feedback response rate	43%	76%	90%	91%	92%
DFW rates	27%	24%	13%	13%	9%

Note. DFW = "D," "F" (fail), and withdraw; F = female; FFAL = fully flipped active learning; M = male. My final course feedback survey (given through Canvas) has higher response rates in later semesters because I give students time in class to complete it.

Structured Prep Work

In an FFAL class, the students are to consume the main content before class, typically through some combination of reading and watching videos. A major challenge with flipped classes is ensuring that students actually complete the prep work, which I called the forcing function. Both components (prep work content and prep work forcing function) are important.

Prep Work Content

I have assigned two different textbooks as reading for the Data Structures course. In the first few semesters, I assigned Lewis and Chase's *Java Software Structures*, a book that explained things well and focused on both the implementation and use of data structures and provided a lot of sample code. While I really like this book, I found that students were not always reading it.

For the last three semesters of the class, I have used an interactive, online textbook as part of the prep work in the class. This book intersperses the reading content with multiple-choice questions, animations, fill-in-the-blank questions, and other short activities designed to help students test their learning (Figure 3.1).

The zyBooks interactive textbook integrates with the Canvas platform. This enables me to assign sections or chapters of the textbook and grade for participation and challenge activities that the students do as they complete the reading. Because there is a grade associated with the reading, the grade transfer happens before they come to class each week, and this acts as a forcing function. The built-in activities make the reading active and engaging, as shown in Figure 3.2 and Figure 3.3. Results in these figures represent feedback from the fall 2017 semester and come from an end-of-semester anonymous feedback survey. Students were given time in class to respond, and the response rate was 93%.

Videos are another critical component of prep work. The current generation of students generally respond really well to learning by watching videos, and there are so many advantages to the format. Students can watch the videos as many times as they need to. They can pause, scroll back to things they have missed, and take a break and finish a video later. They also have the ability to slow down the video if they have trouble understanding the speaker or watch it at double-speed if they find the video goes too slow or if they just want a brief refresher on the topic. This level of flexibility just is not possible in a classroom lecture. But there are a number of challenges to address with respect to videos, specifically the challenge of creating them yourself or curating from the videos available online. I have done some of both, and I have some best practices to share.

If there are good videos available on a topic, then why reinvent the wheel? For a core topic like data structures, many videos are available on YouTube, and finding a few to use in my class is a massive curation task. One of the best ways to manage this is to involve some undergraduate students in independent studies. I set up a weekly meeting time and define a set of tasks related to the curriculum development for the class. One of the main tasks is video curation. I set up a Google sheet with all the topics and subtopics, and ask the students to add video links, along with comments and descriptions about the videos they find. During the weekly

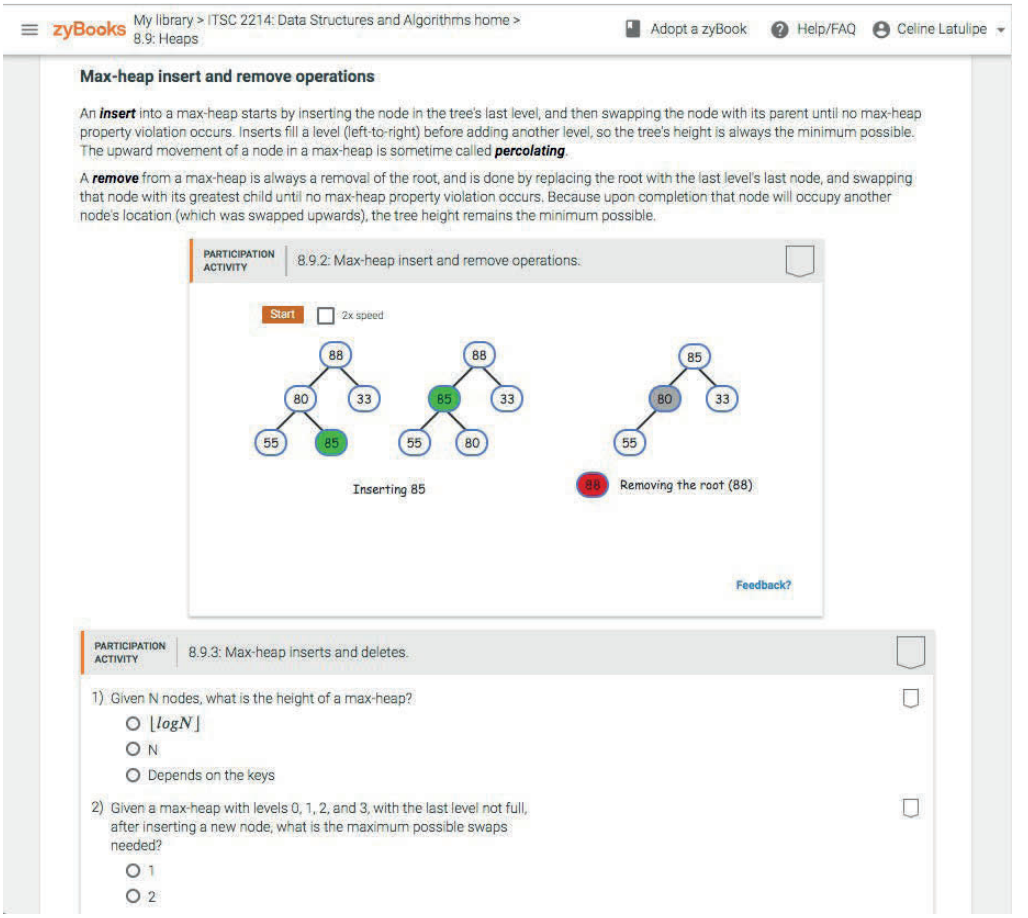

Figure 3.1. Active reading through zyBooks interactive textbook. Grades for participation and challenge activities are automatically submitted to the Canvas gradebook prior to class, so students have the incentive to complete the reading on time.

meetings, we go through the best videos and decide which ones are most suitable for the class. I also involve these students in creating weekly prep work quiz questions and testing out lab exercises or class assignments. The benefit to the students is that they get to revisit material and deepen their learning in the topic while earning credit. They also get exposure to the other side of the classroom—the prep work and planning that goes into course development.

Sometimes there may not be a suitable video on a topic, and that is when I create my own. I have found that students respond well to videos that I have made, and it sends the message that they are learning from me. (Some students may feel that if they only watch videos on YouTube made by other people, they are not getting their money's worth.) I am not capturing videos of me, but rather videos of my screen as I do voice-over and explain things. Some of my videos are of me explaining concepts using animated PowerPoint slides. I try to keep these videos under

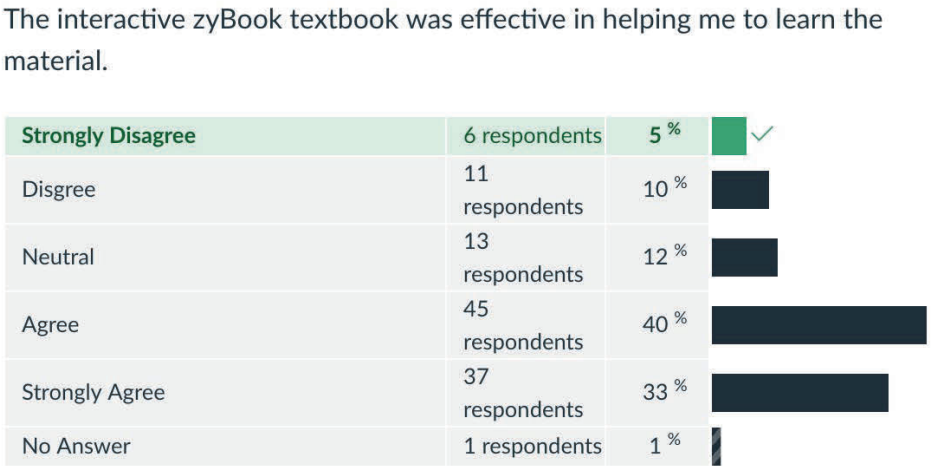

Figure 3.2. Students' perceptions about the effectiveness of the zyBooks.

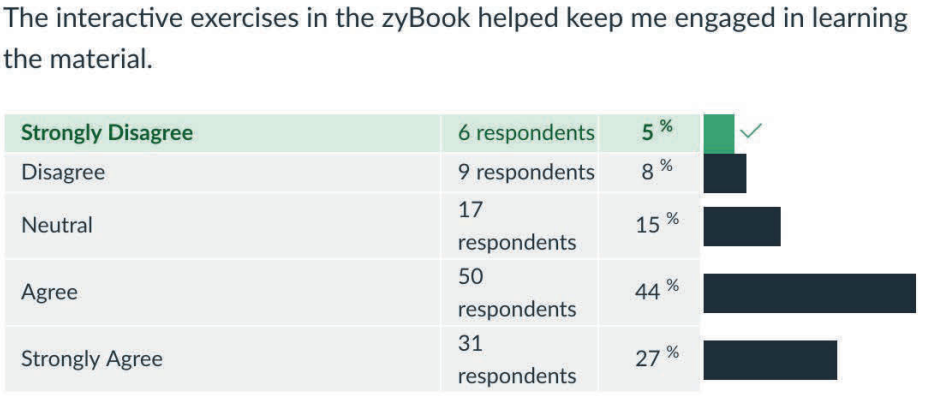

Figure 3.3. Students' perceptions about how well zyBooks keeps them engaged in the reading prep work.

10 minutes, as research shows students do not finish long videos. Some of my videos show me programming something by typing in an editor and running the code. These videos are slightly longer (15 to 20 minutes), but they are intended as reference rather than something to watch for conceptual understanding. I also have used a program on my iPad called ShowMe that is a recorded whiteboard session. With ShowMe, I might create some diagrams ahead of time and load them on different pages, and then walk through annotating and illustrating, while the software records my drawings and my voice (see Figure 3.4).

Two major notes: First, I have never had someone come in and record me giving a full-length class lecture and then just put that online. Research shows that is not effective. A 75-minute lecture tends to put students to sleep, even when they are physically in the same room as the lecturer. We fill those 75 minutes with many examples, and we ramble on trying to relate the

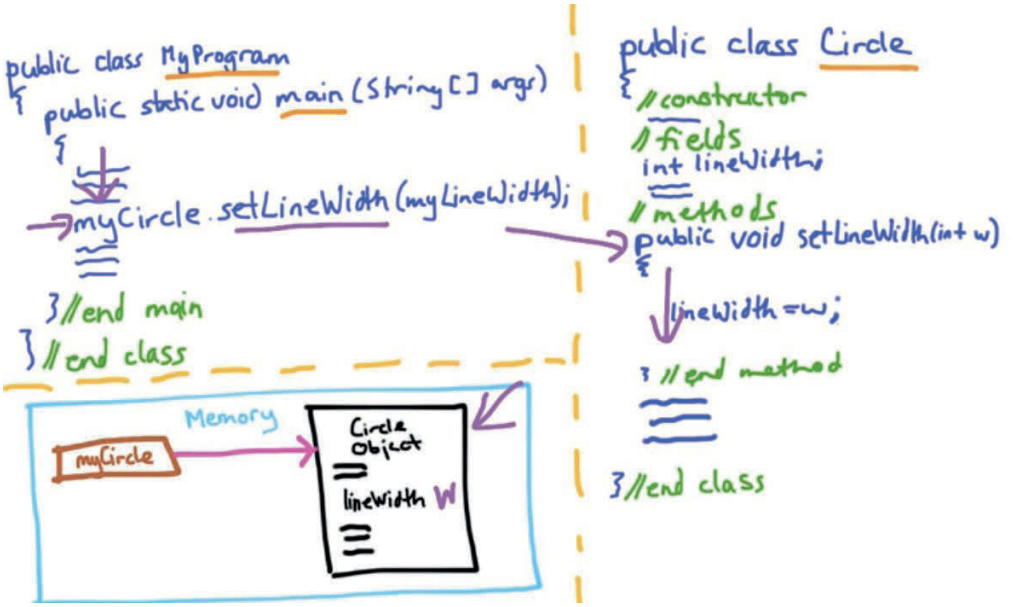

Figure 3.4. Screenshot of a ShowMe video created for the class using the iPad software.

concept to other concepts. Lectures are not typically efficient in terms of content. So breaking up what was communicated into smaller chunks makes way more sense when people are going to be watching the content on video. Second, I do not fret about production quality. It is just me talking to the students, and I think it is more personal that way.

Prep Work Forcing Function

Because the in-class active learning activities will assume the students have been exposed to (and hopefully have absorbed) the material, it is critical to ensure the prep work is being completed. The class activities are meant to actively engage the students with the material, tease out any misunderstandings, and help apply the content. None of that can happen if students have not done the prep work. The most effective way that I have found to get students to do the prep work is to have a quiz completed before class. The quiz has to be challenging enough that they cannot just guess the answers. It also needs to be worth points toward their final grade. I typically make prep work quizzes worth about 10% of the grade. Each quiz is worth less than 1%, so it is not enough to incentivize cheating, but it is enough to incentivize them to do the work. It is not 100% perfect. The impact of social structure of the class also tends to act as an implicit forcing function. That is one of the benefits of the lightweight teams structure, which is the second major component of the class and the topic of the next section.

Lightweight Teams

FFAL classes provide an excellent opportunity to leverage social learning. While you could have a fully flipped classroom where students do the prep work and then come to class to silently work alone on active learning activities, this would offer little benefit over the non-flipped classroom. The only benefit in this case would be that the teaching assistants (TAs) and professors are available to help students if they struggle. The real benefit of the fully flipped classroom stems from the social learning that happens when the classroom is structured so that student–student interaction is the expected norm. While this could happen to some extent naturally, the lightweight teams approach (Latulipe, Long, & Seminario, 2015; Latulipe, MacNeil, et al., 2018; Latulipe, Rorrer, & Long, 2018) is designed to promote and facilitate social learning, and it is particularly useful in medium and large classes.

Lightweight teams are teams of four to nine students who sit together all semester long and work on activities that are worth very little toward their final grades. This lack of emphasis on graded work is what makes these teams lightweight and very different from traditional student project teams. I devise a seating plan for the classroom, specify where each team sits, and display this on the first day of classes. I do not care which seat a student sits in, as long as the student sits with their team. The main idea with lightweight teams is that the students get comfortable with each other and are given activities that will benefit them from discussing the work with each other.

Team Logistics

The size of the teams is somewhat dependent on the classroom layout, but I believe that five in a group is the best team size. In the Kennedy active learning classrooms, the round tables seat nine students, so when I first taught there, I created teams of nine. However, a team of nine students leaves a lot of room for students to fade into the background and not participate. In that classroom, I now create teams of four or five students, with two teams at each table. In classrooms that have really small tables, I create teams of four. In more traditional classrooms with rows of tables and chairs, I go with teams of five and use a layout that alternates two in front, three behind with three in front, two behind, as shown in Figure 3.5 for one of the CHHS classrooms:

There are various ways to form lightweight teams, but the general approach is to let the learning management system (LMS) randomly create them, only making adjustments to fit the situation. In computer science, there is a lack of female students. I do not allow a lone female to be on a team with four males, so I will pair the female students up. This means there are a bunch of teams with no female students at all, but it also means the female students are less likely to feel isolated. I also try to do the same with underrepresented minority students. Determining student gender identity and racial identity is tricky and typically requires downloading institutional reports as this information is not available in Canvas or Banner. In more recent semesters, I have asked students to provide their preferred pronouns on an introductory background survey and use that to help with the team formation.

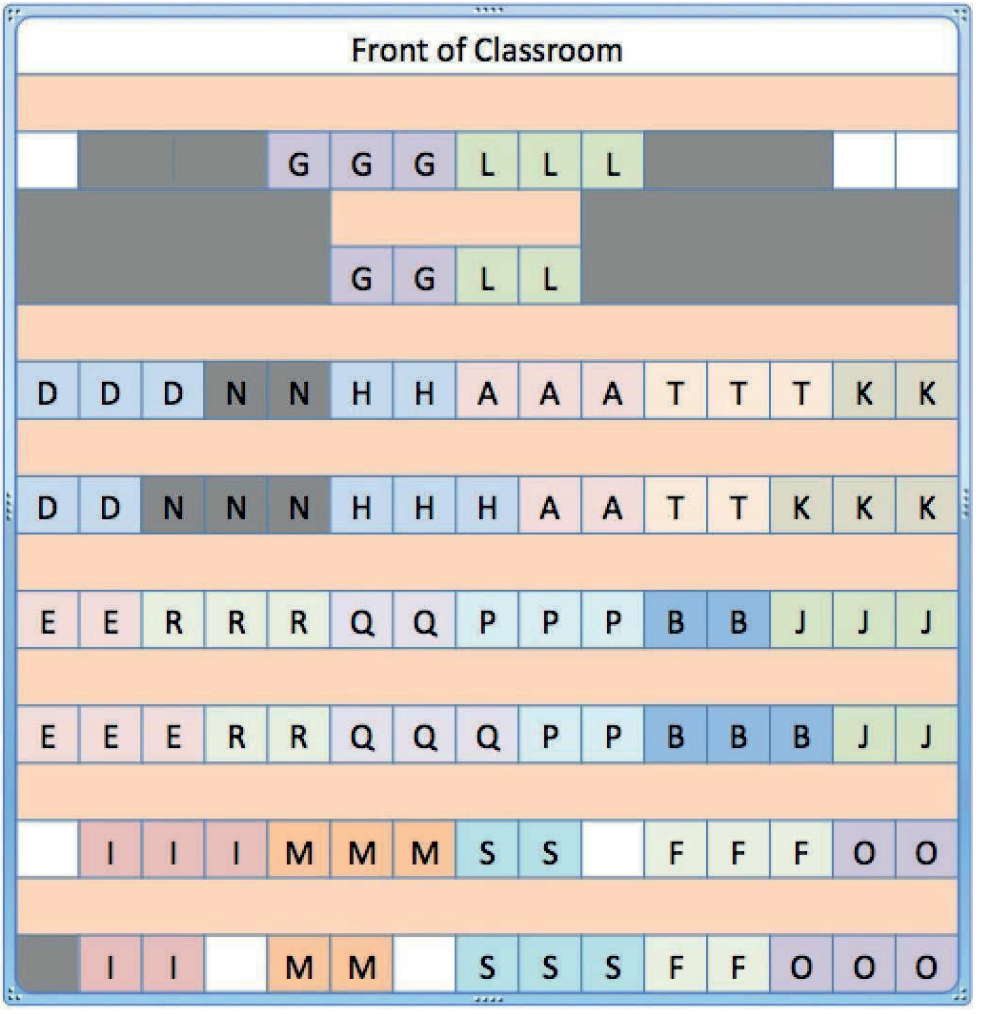

Figure 3.5. A classroom with rows of tables and chairs, set up to accommodate 18 teams with five students each (total of 90 students). Teams are denoted by letters, with Teams G and L at the front of the classroom.

Another consideration might be skill or background level. In classes with heavyweight projects, this type of balance is critical. In first- and second-year classes with lightweight teams, skill balance is somewhat less critical. However, I have experienced semesters where there are some teams full of great students and other teams full of students with much less background preparation, and mixing these students up ensures that there are well-prepared students on every team. In recent semesters, I have given students a programming assessment test on the first day of class so that I can plan for there to be a mix of levels of programming experience on each team.

It sometimes feels like a lot of effort to try to balance teams across gender, race, and back-

ground preparation. And even after all this balancing, there are occasionally teams that just do not work well together. It is amazing how much one or two extraverts at a table can make a difference, and I have not yet found a way to ensure that I do not end up with an entire team of extreme introverts. There are a lot of introverts in the computing major, so this does happen. Introverts often end up becoming great team members once they get comfortable. But I have seen a team full of introverts stay fairly quiet all semester long because there is not someone on the team to get them started talking.

Benefits of Lightweight Teams

Lightweight teams enable us to really leverage the power of social learning. When a student hears other students' perspectives on a concept, it gives that student new ways to think about and relate to that concept. By working and talking with others, students get to practice all of the crucial skills needed in the professional world: socializing, talking, working together, and communicating within the discipline. It also works to reinforce their own knowledge by explaining what they know to other students. By having students communicate with each other, we are allowing them to become more engaged in the topic than they would be if they sat silently listening to a faculty member lecture.

In sociology, the contact hypothesis (McKeown & Dixon, 2017) explains that working together with others who are different from us actually helps mitigate prejudice. This is important in computer science where there are domain-specific prejudices, namely that boys are better at programming and technical work, that layer on top of the standard prejudices prevalent in U.S. society. In addition to helping students realize that they have more in common with "others" than they think, I have come to believe that the lightweight teams approach is also beneficial for moderating student confidence. Students who come in thinking that they know nothing find out that there are others who also do not have a lot of previous experience, and over time, they realize that they can, in fact, contribute to the conversation. On the other hand, students who come in thinking they know everything will soon realize through the team-based peer instruction quizzes, that they do not know everything. Those who have significant prior experience (such as taking programming classes in high school) may come to realize that not everyone was as privileged and that in fact many high schools do not offer any programming courses at all. None of these insights can be gained by students who sit silently in a lecture hall.

Lightweight Team Results

Every semester I survey students anonymously to get feedback about their experience in this class, and I find that most students in Data Structures are generally happy to be on a team. When I first started teaching this way, I thought that students put on a team without choosing their own teammates would hate being told where to sit. But results show that the students do see the benefits of this approach, as demonstrated by Figure 3.6, Figure 3.7, and Figure 3.8. I am forcing them to meet new people and make new friends while also allowing them to learn together and from each other. Creating a social environment in the class makes the learning more enjoyable.

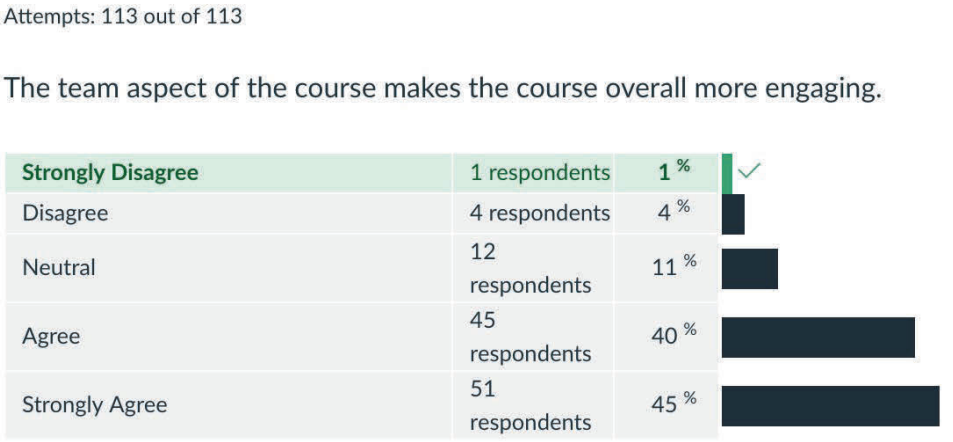

Figure 3.6. Students' perceptions about how being on a team impacts engagement with the course.

Figure 3.7. Students' perceptions about being on a team and the fixed seating plan.

I also asked students how many new friends they made as a result of taking this class. The results can be seen in Figure 3.9. Now, this may seem like a bizarre thing to measure because, after all, are they not here to learn? But this is when we must think about the bigger picture: retention within the program and within the university and helping our students become better global citizens. Helping students to develop interpersonal relationships is an important aspect of the college experience. Many of our students have part-time jobs, which means that they do not have a lot of time to spend on campus engaged in extracurricular activities. For these students, the socioacademic integrative moments (Deli-Amen, 2011) that we can provide in the classroom are critical in helping them feel like they are part of the campus community.

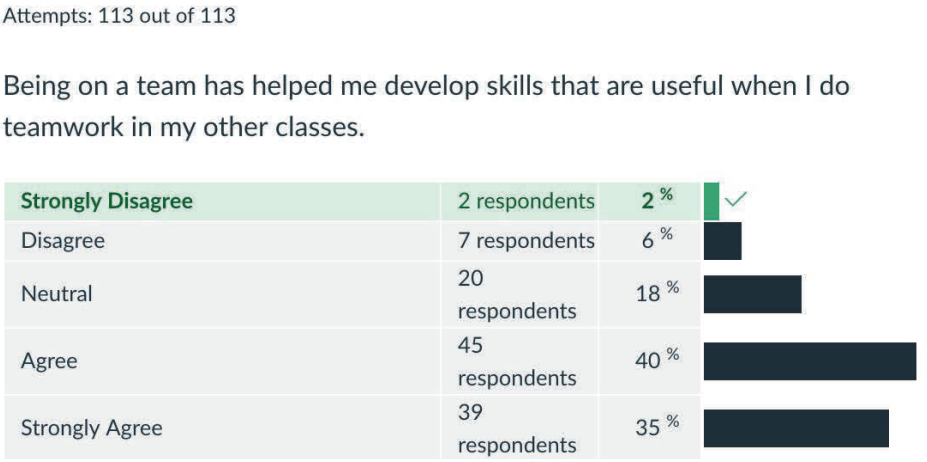

Figure 3.8. Students' perceptions about how being on a team impacts their teamwork skills.

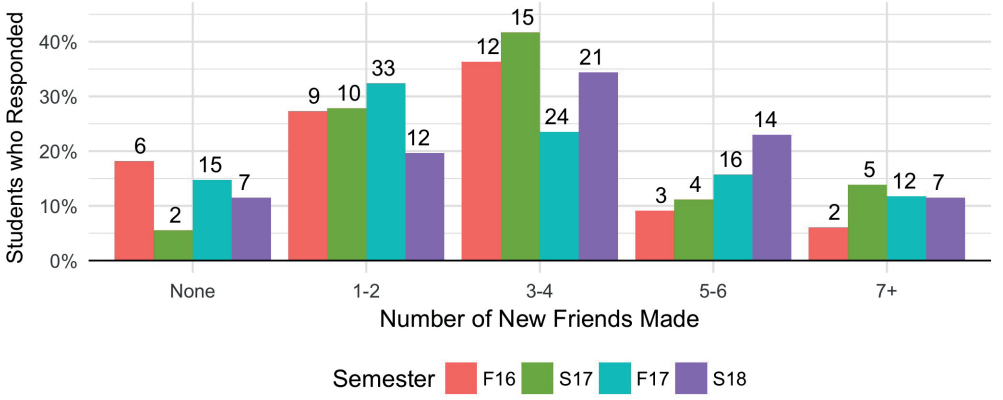

Figure 3.9. Histogram showing how many new friends students made each semester as a result of taking this class. For example, in the spring 2018 semester (purple), 14 students reported making five or six new friends.

Active Learning

In the classroom, my aim is to engage students in active learning activities that get them to move up Bloom's taxonomy from simple information recall to understanding, synthesis, application, and evaluation activities. With three hours per week of instruction, I divide the time between peer instruction quizzes and programming activities. Peer instruction quizzes focus on understanding, synthesis, and evaluation activities while programming activities require those types of cognition as well as application.

Peer instruction quizzes with Poll Everywhere are the anchor activity in my FFAL classes.

1. What does **extends** do in the first parameter type in this method?

```
public int myMethod(<T extends Number<T>> t1, int i) {…}
```

A. It allows any object passed in to be treated as a Number
B. It ensures only objects from classes that implement/extend Number can be used for this parameter
C. It ensures that all the following parameters in the method header are class types that extend Number
D. It allows returns of generic type

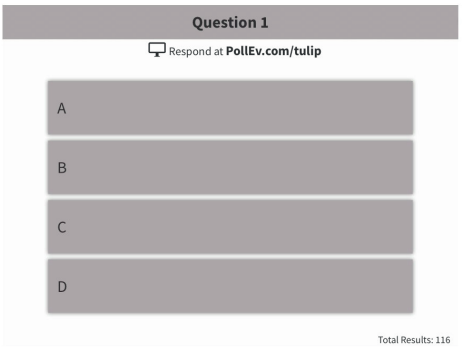

Figure 3.10. A sample PowerPoint slide showing a question and the Poll Everywhere poll. Student responses are hidden, although the number at the bottom right indicates 116 students have answered.

I have created PowerPoint slide decks with my questions that allow me to type in formatted code, show screenshots of programming environments, etc. Each slide has the question and the Poll Everywhere poll inserted (see Figure 3.10), and the students use their smartphones or laptops to log in to respond.

I encourage students to talk to each other before answering to stimulate communication between students. I do not consider this an assessment activity, so I am not worried that students might get the answer from others. Instead, I treat this as a learning activity and continually remind them that they need to not just say to their teammates, "the answer is B," but also to explain to each other *why* they think that B is the correct answer. I repoll questions if the histogram of responses shows that students are not in agreement about the correct answer (as in Figure 3.11).

When these disagreements arise, I encourage students to talk to students on other teams before taking a second shot at answering. My instruction in this situation is "Get up and walk around—find someone who has a different answer than you and convince them that you are right."

When there is this level of confusion about a topic, as evidenced by the answer distribution, I take this as a sign of a general misconception and will use this time to launch into an impromptu minilecture to clear up the misconception. This typically involves me using Sharpie pens and a sketchbook under the doc cam so that my explanation can be broadcast to screens around the room. The beauty of such impromptu, on-demand explanations is that the students are fully tuned in. They have just lost points in the Poll Everywhere quiz and know there is something that they do not understand, so they are fully primed to listen to the explanation.

10. What is the efficiency of the following method:

```
public String[] getNames(int n) {
    String s[] = new String[n];
    for (int i = 0; i < n; i++) {
        try {
            s[i] = myStack.pop();
            System.out.println("Just got name: " + s[i] + " from the stac
        } catch (EmptyCollectionException ex) {
            Logger.getLogger(LinkedStackTest.class.getName()).log(Level.S
        }
    }
    return s;
}
```

A. O(n)
B. O(4)
C. O(1)
D. O(4n)

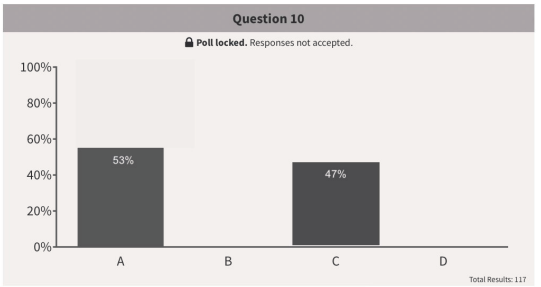

Figure 3.11. In this slide, student responses are shown and indicate that there is a split, with half the students thinking A is correct and half the students thinking C is correct. This shows a misconception and would lead to a five-minute minilecture explanation.

These short explanations usually last no more than four minutes, so the students do not have a chance to get bored and tune out.

Varied Assessment

One of the most important aspects of the FFAL class is that the grading is as structured as the rest of the course. In a traditional Data Structures class, students might be assigned a few programming assignments and have a midterm and final exam. This format gives students no feedback on their understanding of the material or their mastery of requisite skills until the midterm or first assignment is graded. It gives students the impression that nothing major is required of them until the midterm or the first assignment deadline. In FFAL classes, especially at the freshman and sophomore levels, providing a lot of structure to help students do all the things required for successful learning is key. Research has shown that providing such structure is especially beneficial to students without a lot of social capital—students coming from lower socioeconomic status (SES) backgrounds, who are first-generation college students, etc. These students often do not get a lot of help or guidance from their families and home communities about what it means to succeed in such an environment. Providing significant structure helps scaffold these students toward success while also helping them understand what activities they need to be successful at university classes. By encouraging early successes, we can build up their social capital and self-efficacy so that they have a better chance of succeeding in later

Table 3.2 Weighted Grading Scheme

Activity	Grade weight
Prep work quizzes	10%
zyBooks interactive textbook reading	10%
Poll Everywhere peer instruction quizzes	5%
In-class programming labs	10%
Tests	20%
Individual programming assignments	40%
Reflections and sketchbooks	5%
Each unexcused absence	(–2%)
Total	100%

Note. In this class, almost everything counts toward a student's final grade.

classes even when the structure is reduced. One form of structure is the forced prep work discussed in the last section. Another form of structure is in varied grading, which is discussed next.

In my Data Structures class, everything a student does counts toward their final grade in some way. Table 3.2 shows the weighted grading scheme for the class.

Some of the components that are graded are items that students do in groups or pairs such as the Poll Everywhere quizzes and in-class programming labs, but these are only worth 15% of the final grade. Yes, it is possible some students could completely rely on other students to do well in these portions, but it would not be enough to pass a student who is otherwise completely failing to do any work because 60% of the final grade is based on tests and assignments that each student must complete individually.

The effect of this varied grading scheme is that it forces students to put in the work that is necessary for success. They get points for reading the interactive textbook and completing the questions that are embedded in the book; these questions are designed to help students check their understanding while forcing them to be actively engaged in the reading. Getting points for completing their prep work and the associated quiz before coming to class means they are more likely ready to be engaged for the in-class active learning. Because they have put in this study time and are prepared for the in-class work, they get more out of the in-class active learning activities, which helps prepare them for their tests and individual assignments.

I apply the low-stakes testing philosophy to this class. Rather than two high-stakes exams, I have students take four noncumulative tests throughout the semester. This reduces the amount of test anxiety and allows students to focus more on learning a smaller amount of material rather than trying to simply cram in a large amount of material. High-stakes exams typically lead to surface learning.

There is a negative grade weight associated with unexcused absences in the class. While I feel that it does not make sense to give a student points "just for showing up," missing class means they are losing valuable learning opportunities. Thus, I impose a –2% penalty for unexcused absences. This ensures that students are punctual. If a student has a medical excuse, reasonable documentation, or an excuse approved by the dean of students, I do not deduct these points.

Finally, there is also a hidden performance incentive related to attendance and tests. If a student has no unexcused absences at the end of the semester, they are given the opportunity to retake one of the first three tests after taking Test 4 during the final exam period session. This is a win–win situation; students go back and revisit the material that they struggled with the most, thereby enhancing their learning. The chance to raise their grade provides students with even more incentive to make sure they attend every class.

Supporting Technologies

In addition to the four structural elements I have covered, it is also important to note that I made heavy use of various technologies to run this FFAL class. While some of those technologies have been explained in the sections above (Poll Everywhere, zyBooks), there are other technologies that played an important supporting role and are worthy of mention. Below I briefly describe how and why I consider each to be critical.

Canvas

The Canvas LMS is my organizational go to. I structure the Canvas page using weekly modules that show students the flow of work in the class. Figure 3.12 shows what a typical week looks like.

I often make use of the dependencies built in to Canvas, which allow you to specify that certain elements are not accessible to students until they complete other items. This ensures students cannot have access to in-class lab materials until they have completed prep work, forcing them to stay on top of the work.

Piazza

In an FFAL class, where there are many moving parts, students may have lots of questions about expectations and the varying tools being used. In order to prevent my email inbox from being overwhelmed, I have a strict communications policy. Questions about the class, content, assignments, etc. must be posted on the class discussion forum on Piazza. I have chosen to use Piazza for a number of reasons. First, Piazza is a much faster system than the built-in discussion forum in Canvas, which has a poor user interface and is typically slow to load. Second, Piazza allows students to post questions anonymously (though the posts are only anonymous to other students, not to the TAs or professors). This is very important because many students are afraid that if they ask a question they will "look dumb" in front of their peers. Allowing anonymous posts gives them the opportunity to post without unnecessary anxiety. Third, Pi-

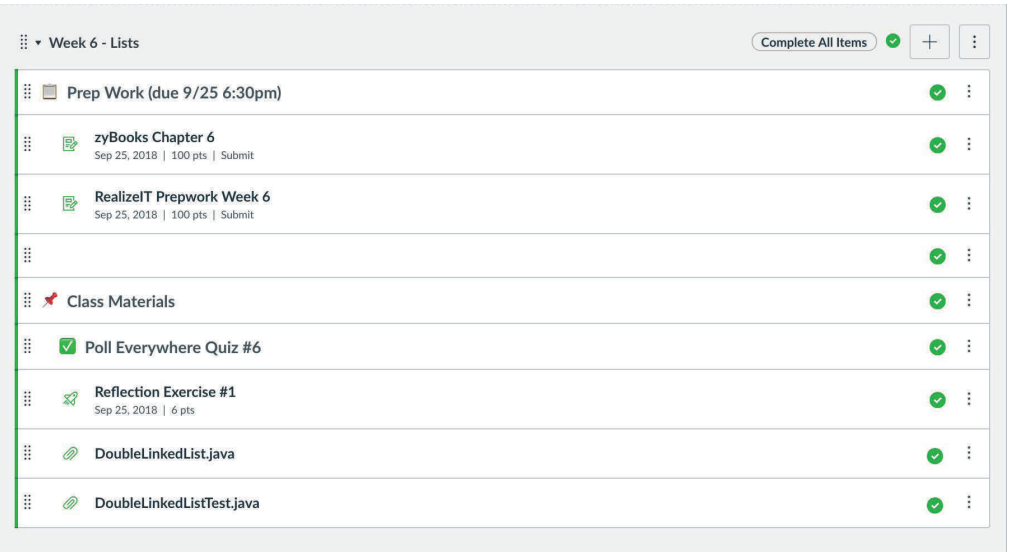

Figure 3.12. A typical weekly module in Canvas, with separated sections for prep work and class materials.

azza allows up-voting of questions so the teaching team can see which questions or comments the students agree are important. Finally, the homepage of the Piazza forum tells me at a glance what is happening in the course discussion (see Figure 3.13).

To come back to the communications policy, this ensures that all students have access to information provided in response to questions about course content, policies, assignment clarifications, etc. This information transparency helps to ensure equity, so that the information is available even to those students who do not have the social capital to know that they could be asking those questions at all. The only exceptions to this communication policy are for when students need to email me about their personal grades, an illness, or other personal issues.

RealizeIT

In the fall 2018 semester, I embarked on a new adventure with the class by moving most of the prep work content into an adaptive learning platform called RealizeIT. In the RealizeIT platform, students enter each week's module and immediately take a "determine knowledge" quiz. This then populates a learning map with varying levels of mastery, depending on how much a student already knows about the concepts covered in the module. The learning map not only provides a visual indication of how the concepts they are learning link together, but also shows students how much they have grasped each concept and what the mastery level of everyone else in the class is. Based on the determine knowledge quiz results and the student's prior learning, the system personalizes the learning by pointing students to appropriate starting points. From there, they work through each node consuming material through video or text content. The learning map could be considered a gamification of the learning experience as it encourages students to work hard to get all green stars across the learning map.

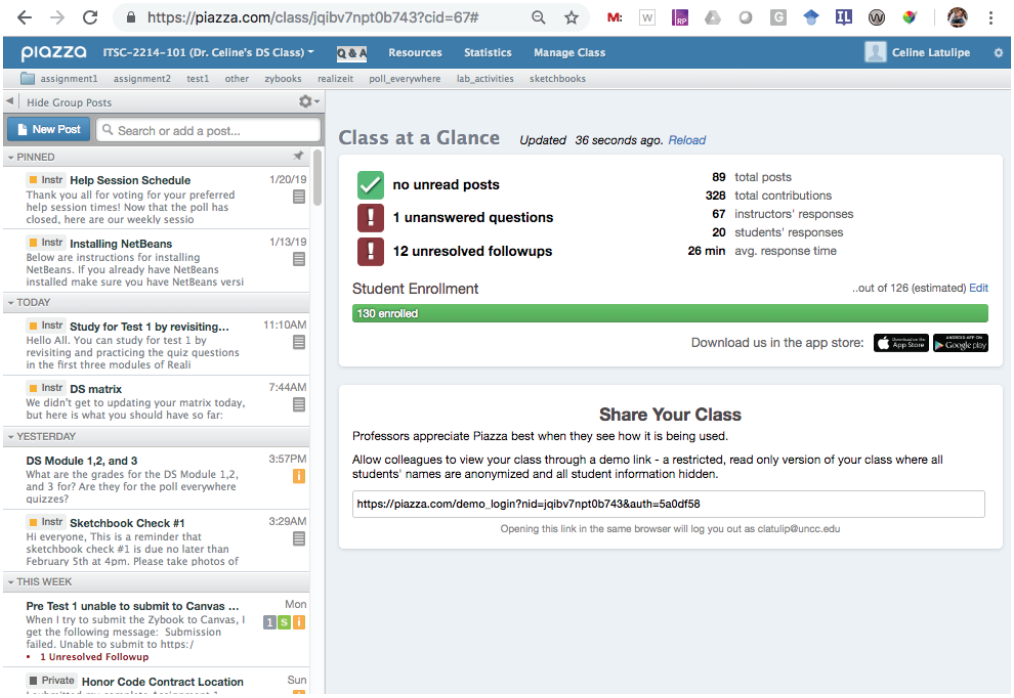

Figure 3.13. The Piazza course discussion interface. The homepage shows me at a glance the number of new posts and unresolved questions. Each post also has icons showing whether other students or instructors have responded.

There were some bumps in the road during the first semester use of RealizeIT and mixed feedback from the students, but it does appear that students are spending more time on their prep work in order to reach mastery. I have even been able to increase the difficulty of the in-class Poll Everywhere questions as a result. In the spring 2019 semester, the RealizeIT platform provided integration with zyBooks, and the adaptive learning experience is something that we will continue to expand on. Measuring the effectiveness of this platform and how it impacts student learning is important. The DFW rates dropped from 13% to 9% when the course switched to the adaptive learning platform (see Table 3.1), and while we cannot say for sure that it was because of RealizeIT, the delta shows a positive correlation.

In addition to the three technologies described above, I also made use of numerous domain-specific technologies to help teach programming and data structures concepts. The Web-Cat system for autograding programming assignments proved to be a crucial tool for programming classes. Students submit their code that is then run against a set of test cases. Students receive feedback on what test cases their code did not pass and have the chance to resubmit their programs up to 30 times for each assignment. This allows them to learn how to make robust code that does not break when odd inputs are applied.

Other programs I use are the MOSS system, which is like plagiarism detection for code. I

Figure 3.14. The learning map shown to students for a module in the RealizeIT adaptive learning platform.

make use of a Code Workout server that provides students with small practice programming assignments, and CodingBat, which similarly provides programming practice assignments. And finally, the website visualgo.net provides step-through animations of data structures and online quizzes to test student understanding.

Conclusion

In this chapter, I have described in detail the workings of an FFAL class in the College of Computing and Informatics. Every detail of this class has been engineered, designed, and tweaked in an attempt to create a rich learning experience for students. The goals are not only to make

sure students learn about data structures and develop their programming skills, but also to ensure that they develop their social, communication, and teamwork skills by developing relationships with other students in the computing major. There are many moving parts, and as seen by the staffing row in Table 3.1, it takes more than just one person to run this course. The course depends on a dedicated set of TAs who care deeply about the class and the learning experiences of the students coming up behind them. A later chapter in this book focuses on how a stepping stone mentorship model is an important element in making such complex classes work well.

References

Adams, N. E. (2015). Bloom's taxonomy of cognitive learning objectives. *Journal of the Medical Library Association*, *103*(3), 152–153. https://doi.org/10.3163/1536-5050.103.3.010

Deil-Amen, R. (2011). Socio-academic integrative moments: Rethinking academic and social integration among two-year college students in career-related programs. *The Journal of Higher Education*, *82*(1), 54–91. https://doi.org/10.1353/jhe.2011.0006

Eddy, S. L., & Hogan, K. A. (2014). Getting under the hood: How and for whom does increasing course structure work? *CBE—Life Sciences Education*, *13*(3), 453–468.
https://doi.org/10.1187/cbe.14-03-0050

Latulipe, C., Long, N. B., & Seminario, C. E. (2015). Structuring flipped classes with lightweight teams and gamification. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 392–397. https://doi.org/10.1145/2676723.2677240

Latulipe, C., MacNeil, S., & Thompson, B. (2018). Evolving a data structures class toward inclusive success. *2018 IEEE Frontiers in Education Conference (FIE)*, 1–9. https://doi.org/10.1109/FIE.2018.8659334

Latulipe, C., Rorrer, A., & Long, B. (2018). Longitudinal data on flipped class effects on performance in CS1 and retention after CS1. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 411–416. https://doi.org/10.1145/3159450.3159518

Maher, M. L., Latulipe, C., Lipford, H., & Rorrer, A. (2015). Flipped classroom strategies for CS education. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 218–223. https://doi.org/10.1145/2676723.2677252

McKeown, S., & Dixon, J. (2017). The "contact hypothesis": Critical reflections and future directions. *Social and Personality Psychology Compass*, *11*(1), e12295. https://doi.org/10.1111/spc3.12295

Thai, N. T. T., De Wever, B., & Valcke, M. (2017). The impact of a flipped classroom design on learning performance in higher education: Looking for the best "blend" of lectures and guiding questions with feedback. *Computers & Education*, *107*, 113–126. https://doi.org/10.1016/j.compedu.2017.01.003